Analysis

Serafina

4/22/2019

Problem

Given a movie rating historical data, Our task is to create a recommendation system using predictive approach which aims to predict the rating value for a user-movie combination. We'll study and analyzed using Regression and Matrix Factorization with Stochastic Gradient Boosting (SGD).

RMSE will be used for the evaluation of the model on validation dataset (10% of the historical dataset). The rating column will used to evaluate the prediction result from the model.

Exploratory Analysis

Load libraries and the edx dataset

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.1
```

```
## -- Attaching packages ---------------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  2.1.1     v dplyr   0.8.1
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

```
## Warning: package 'tibble' was built under R version 3.5.3
```

```
## Warning: package 'tidyr' was built under R version 3.5.1
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
## Warning: package 'stringr' was built under R version 3.5.1
```

```
## -- Conflicts ------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.5.1
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(Matrix.utils)
```

```
## Warning: package 'Matrix.utils' was built under R version 3.5.3
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
library(DT)
```

```
## Warning: package 'DT' was built under R version 3.5.3
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 3.5.1
##
## Attaching package: 'lubridate'
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year
## The following object is masked from 'package:base':
##
##     date
```

```
library(irlba)
```

```
## Warning: package 'irlba' was built under R version 3.5.1
```

```
library(recommenderlab)
```

```
## Warning: package 'recommenderlab' was built under R version 3.5.1
## Loading required package: arules
## Warning: package 'arules' was built under R version 3.5.1
##
## Attaching package: 'arules'
## The following object is masked from 'package:dplyr':
##
##     recode
## The following objects are masked from 'package:base':
##
##     abbreviate, write
## Loading required package: proxy
## Warning: package 'proxy' was built under R version 3.5.1
##
## Attaching package: 'proxy'
## The following object is masked from 'package:Matrix':
##
##     as.matrix
```

```
## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry
```

```
library(recosystem)
```

```
## Warning: package 'recosystem' was built under R version 3.5.3
```

```
#library(h2o)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.1

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked from 'package:recommenderlab':
##
##     MAE, RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 3.5.3

##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 3.5.3

## Loading required package: RColorBrewer
```

```
library(RColorBrewer)
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 3.5.1
```

```
load("~/harvard.RData")
```

Originally, The dataset consist of 7 features for a total of 9000055 observations, where each row represents a rating per user per movie. There some issues such as, most of the movies have their debut year added to their names - we want to extract this into separate columns. Also, Genres columns contains multiple categories per row - we want to have them separated into one category per row. We're using below piece of code to solve above issues.

```
edx <- sample_n(edx, 4000055)
glimpse(edx)
```

```
## Observations: 4,000,055
## Variables: 7
## $ userId    <int> 50805, 42011, 45985, 19660, 46073, 13268, 6910, 3026...
## $ movieId   <dbl> 1095, 6303, 593, 4210, 186, 485, 596, 2402, 913, 623...
## $ rating    <dbl> 4.0, 3.5, 4.0, 4.0, 4.0, 2.0, 2.5, 2.0, 3.5, 3.0, 1....
## $ timestamp <int> 952366935, 1121891132, 854490846, 1207988321, 112368...
## $ title     <chr> "Glengarry Glen Ross", "Andromeda Strain, The", "Sil...
## $ year      <int> 1992, 1971, 1991, 1986, 1995, 1993, 1940, 1985, 1941...
## $ genres    <chr> "Drama", "Mystery|Sci-Fi", "Crime|Horror|Thriller", ...
```

Due to my limited memory, I will take sample from the edx dataset, i.e. 4000055. Then the validation set will about 10% from these new dataset.

Data Cleaning

```
edx <- edx %>%
  # generic function to turn (no genres listed) to NA
  mutate(genres = if_else(genres == "(no genres listed)", `is.na<-`(genres), genres))
glimpse(edx)
```

```
## Observations: 4,000,055
## Variables: 7
## $ userId    <int> 50805, 42011, 45985, 19660, 46073, 13268, 6910, 3026...
## $ movieId   <dbl> 1095, 6303, 593, 4210, 186, 485, 596, 2402, 913, 623...
## $ rating    <dbl> 4.0, 3.5, 4.0, 4.0, 4.0, 2.0, 2.5, 2.0, 3.5, 3.0, 1....
## $ timestamp <int> 952366935, 1121891132, 854490846, 1207988321, 112368...
## $ title     <chr> "Glengarry Glen Ross", "Andromeda Strain, The", "Sil...
## $ year      <int> 1992, 1971, 1991, 1986, 1995, 1993, 1940, 1985, 1941...
## $ genres    <chr> "Drama", "Mystery|Sci-Fi", "Crime|Horror|Thriller", ...
```

```
#check if some movies don't have debut year
edx %>%  filter(is.na(title) | is.na(year))
```

```
## [1] userId    movieId   rating    timestamp title     year      genres
## <0 rows> (or 0-length row.names)
```

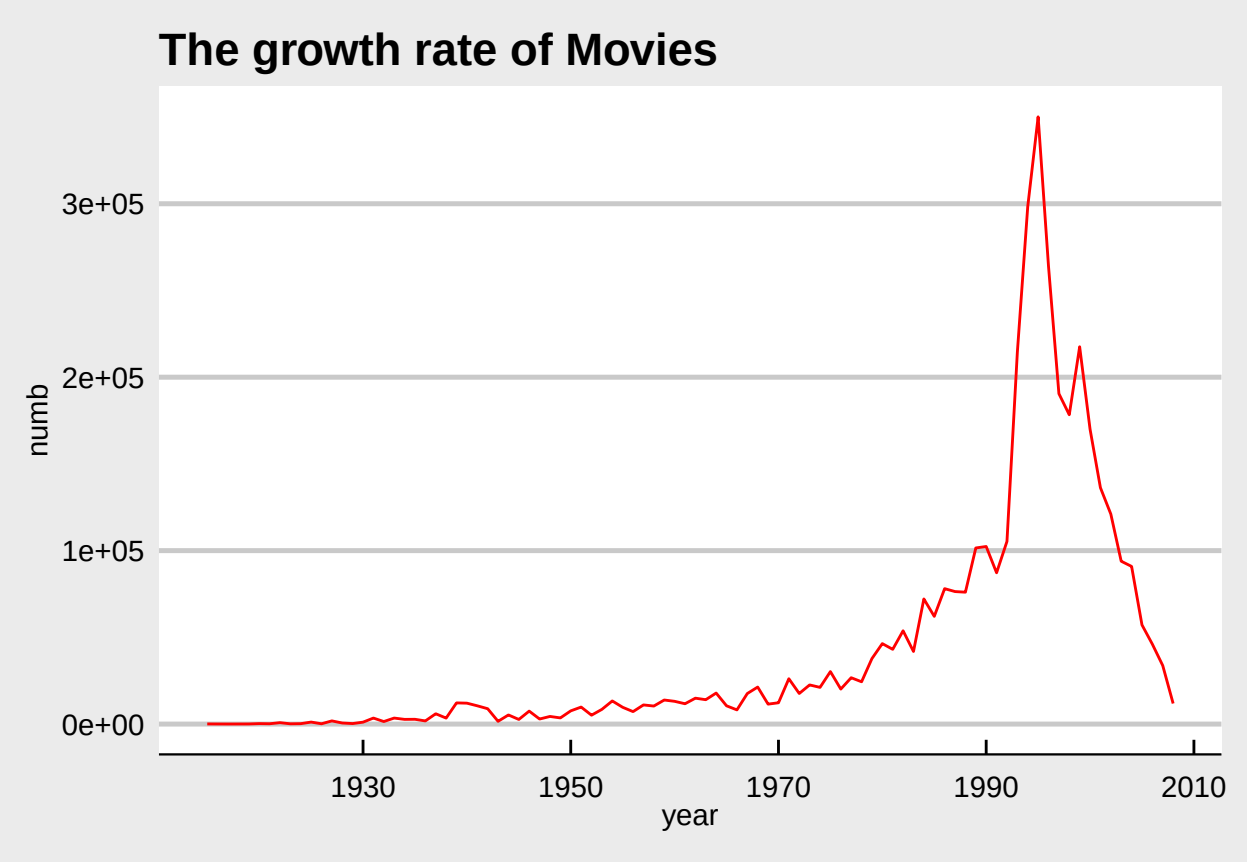
All the movies listed have their debut year. It's great.

Data Exploratory

How was the trend of growth of movies?

```
edx %>%
  na.omit() %>% # omit missing values
  select(movieId, year) %>% # select columns we need
  group_by(year) %>% # group by year
  summarise(numb = n())  %>% # number of produced movies per year
  arrange(desc(year)) %>%
  complete(year = full_seq(year, 1), fill = list(numb = 0)) %>% # in case there are issue where there w
  ggplot(aes(x = year, y = numb)) + #plot
```

```
  geom_line(color="red")+theme_economist_white() +
  ggtitle("The growth rate of Movies")
```



There's an exponential growth of the movie business and a sudden drop in 2000. The latter is caused by the fact that the data is collected until October 2000 so we don't have the full data on this year.Growing popularity of the Internet must have had a positive impact on the demand for movies. That is certainly something worthy of further analysis.

Which Genres get the most Rating?

Let's see which genres is the most popular based on their rating frequency

```
library(DT)
gen <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(freq = n()) %>%
  arrange(desc(freq))

head(gen)
```

```
## # A tibble: 6 x 2
##   genres       freq
##   <chr>       <int>
## 1 Drama     1737624
## 2 Comedy    1573933
## 3 Action    1136816
## 4 Thriller  1033503
```

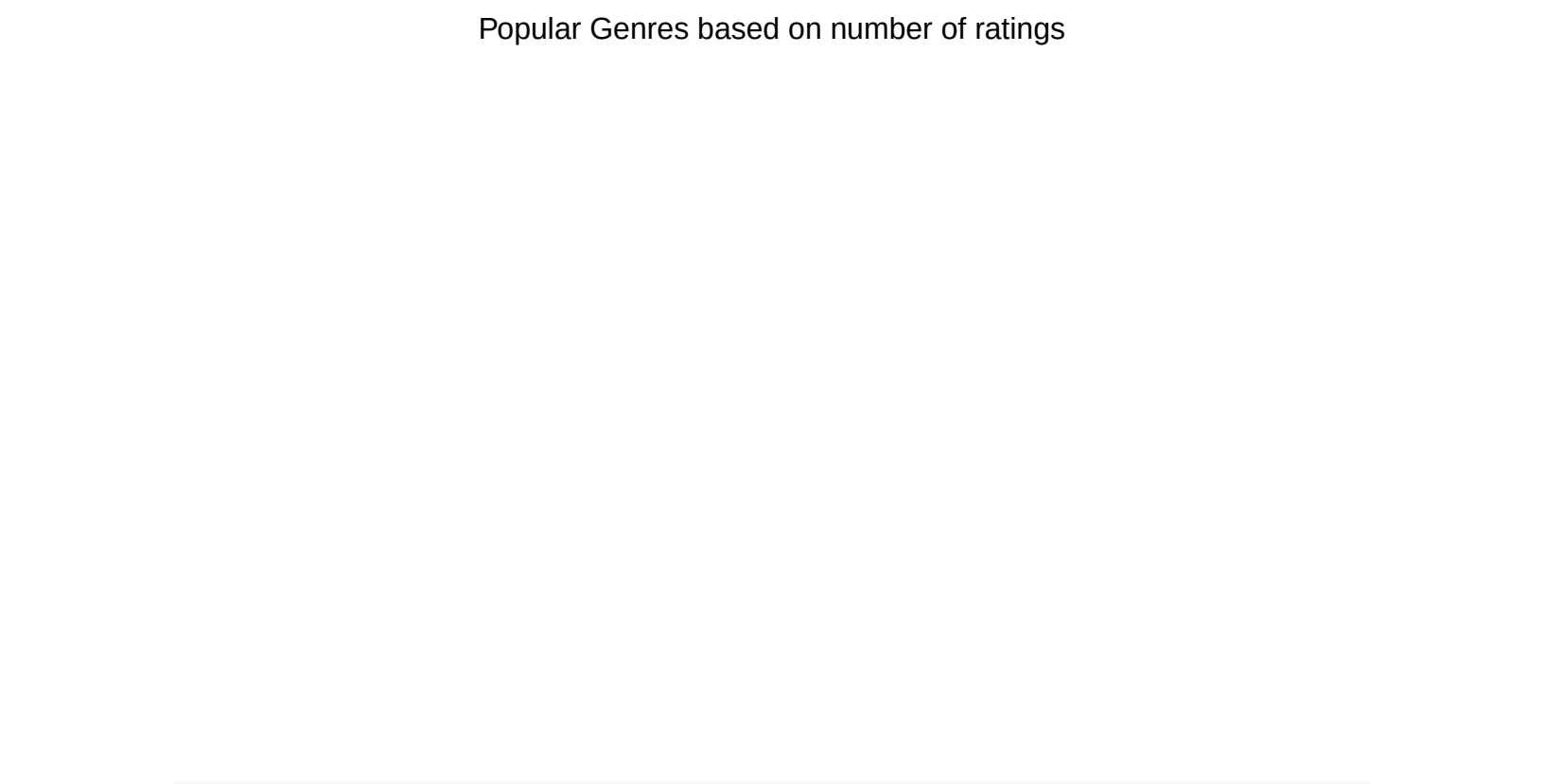
```
## 5 Adventure  847435
## 6 Romance    760748
```

It's not surprise there, the majority of people love drama

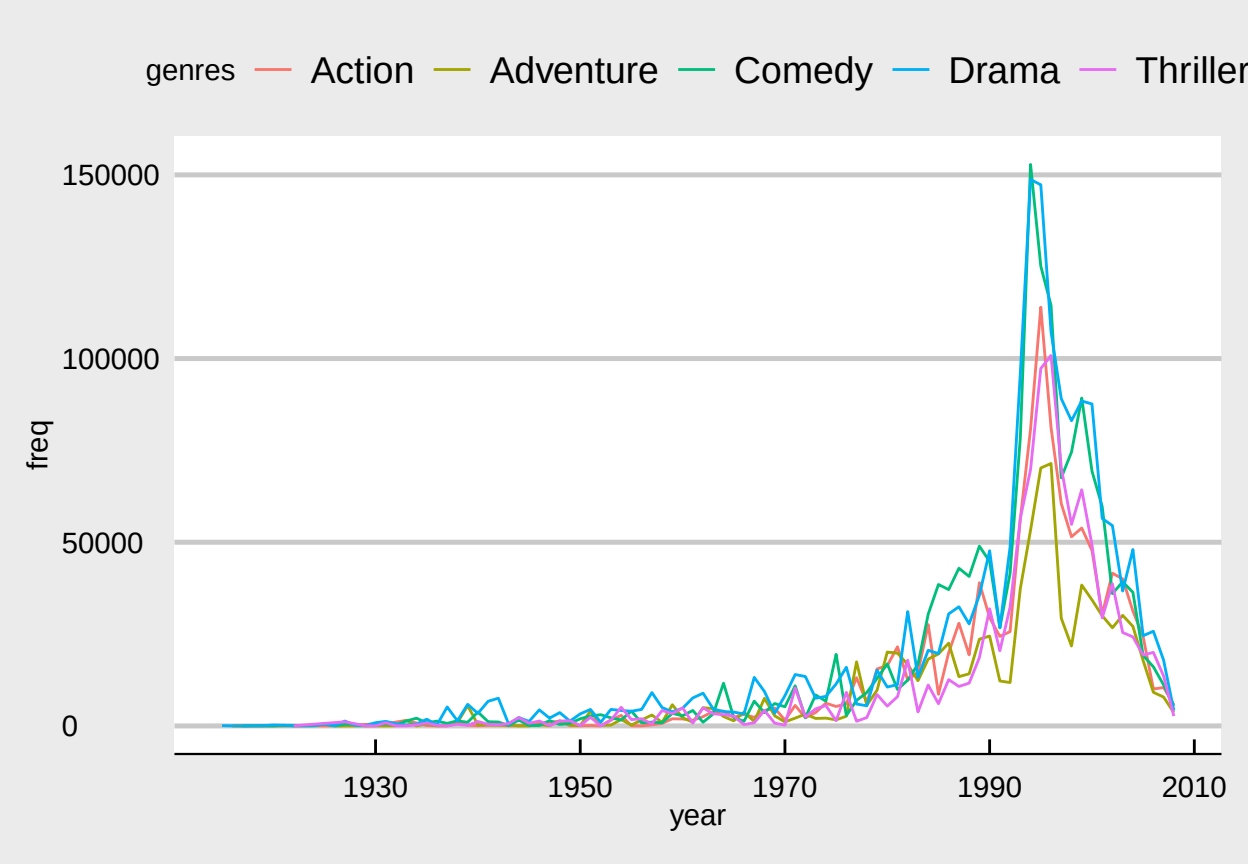
```
library(wordcloud2)
```

```
## Warning: package 'wordcloud2' was built under R version 3.5.3
```

```
layout(matrix(c(1,2), nrow =2) , heights = c(1,4))
par(mar=rep(0,4))
plot.new()
text(x=0.5,y=0.5, "Popular Genres based on number of ratings")
print(wordcloud2(data = gen))
```




```
tmp <- edx %>%
  na.omit() %>% # omit missing values
  select(movieId, year, genres) %>% # select columns we are interested in
  separate_rows(genres, sep = "\\|") %>% # separate genres into rows
  mutate(genres = as.factor(genres)) %>% # turn genres in factors
  group_by(year, genres) %>% # group data by year and genre
  summarise(freq = n()) %>% # count
  complete(year = full_seq(year, 1), genres, fill = list(number = 0))  # add missing years/genres
tmp <- tmp %>% na.omit() %>% filter(genres %in% c("Drama", "Action", "Comedy", "Thriller", "Adventure"))
  ggplot(aes(x = year, y = freq)) +
    geom_line(aes(color=genres))+scale_fill_brewer(palette = "Paired")+theme_economist_white()
print(tmp)
```



There are multiple things to observe: 1. There is rapid growth of Drama movies after 1990, this is modernism area where Hollywood emerged. Child-oriented drama also become more popular in Disney movies.

2. There is a rise of popularity of Action movies, the most probable reason might be the growth of development in computer technology advancement which made the production much easier.
3. Although the Adventure movie has reached its peak at the same time as other genres, it seems people tend to like other genres more.
4. Although thriller genres reached its peak in 1995, it still has the lowest peak among the other genres.
5. As we notice Comedy and Drama movies almost have the same rise and peak, it probably an afteraffect of by the end of the 1970s, Comedy clubs were beginning to spring up everywhere.The comics that had gotten famous in the '70s were now the veterans as a flood of new faces came onto the scene.

What were the best movies of every decade (based on users's ratings)?

First, we need to find what is the highest rated movies in every decade by calculating the average score for each movie

```
#free memory
rm(tmp)
rm(validation, na_df, t)
```

```
avRating <- edx %>%
  na.omit() %>%
  select(title, rating, year) %>%
  group_by(title, year) %>%
  summarise(count = n(), mean = mean(rating), min = min(rating), max = max(rating)) %>%
```

```
  ungroup() %>%
  arrange(desc(mean))
print(head(avRating))
```

```
## # A tibble: 6 x 6
##   title                                    year count  mean   min   max
##   <chr>                                   <int> <int> <dbl> <dbl> <dbl>
## 1 Along Came Jones                         1945     1     5     5     5
## 2 Blue Light, The (Das Blaue Licht)        1932     1     5     5     5
## 3 Class, The (Entre les Murs)              2008     1     5     5     5
## 4 Fighting Elegy (Kenka erejii)            1966     1     5     5     5
## 5 Games People Play: New York              2004     1     5     5     5
## 6 Human Condition III, The (Ningen no joken ~  1961     1     5     5     5
```

We have a problem here, if we sort by average score our ranking will be polluted by movies with low count of rating. To address this problem we'll use weighted average IMDB website for their Top 250 ranking[https://districtdatalabs.silvrback.com/computing-a-bayesian-estimate-of-star-rating-means]

```
# A = average for the movie (mean) = (Rating)
# V = number of votes for the movie = (votes)
# v = minimum votes required to be listed in the Top 250
# m = the mean vote across the whole report
weighted<- function(A, V, v, m) {
  return (V/(V+v))*A + (v/(V+v))*m
}

avRating<- avRating %>%
  mutate(w = weighted(mean, count, 500, mean(mean))) %>%
  arrange(desc(w)) %>%
  select(title, year, count, mean, w)

print(head(avRating))
```

```
## # A tibble: 6 x 5
##   title                      year count  mean     w
##   <chr>                     <int> <int> <dbl> <dbl>
## 1 Pulp Fiction               1994 13911  4.15 0.965
## 2 Forrest Gump               1994 13680  4.01 0.965
## 3 Silence of the Lambs, The  1991 13600  4.20 0.965
## 4 Jurassic Park              1993 12978  3.66 0.963
## 5 Shawshank Redemption, The  1994 12472  4.45 0.961
## 6 Braveheart                 1995 11577  4.08 0.959
```

It's seem better than before. Movies with more good reviews got higher score.

```
# find best movie of a decade based on score
# heavily dependent on the number of reviews
movDecade <- avRating %>%
  mutate(decade = year  %/% 10 * 10) %>%
  arrange(year, desc(w)) %>%
  group_by(decade) %>%
  summarise(title = first(title), w = first(w), mean = first(mean), count = first(count))
head(movDecade)
```

```
## # A tibble: 6 x 5
##   decade title                                          w  mean count
```

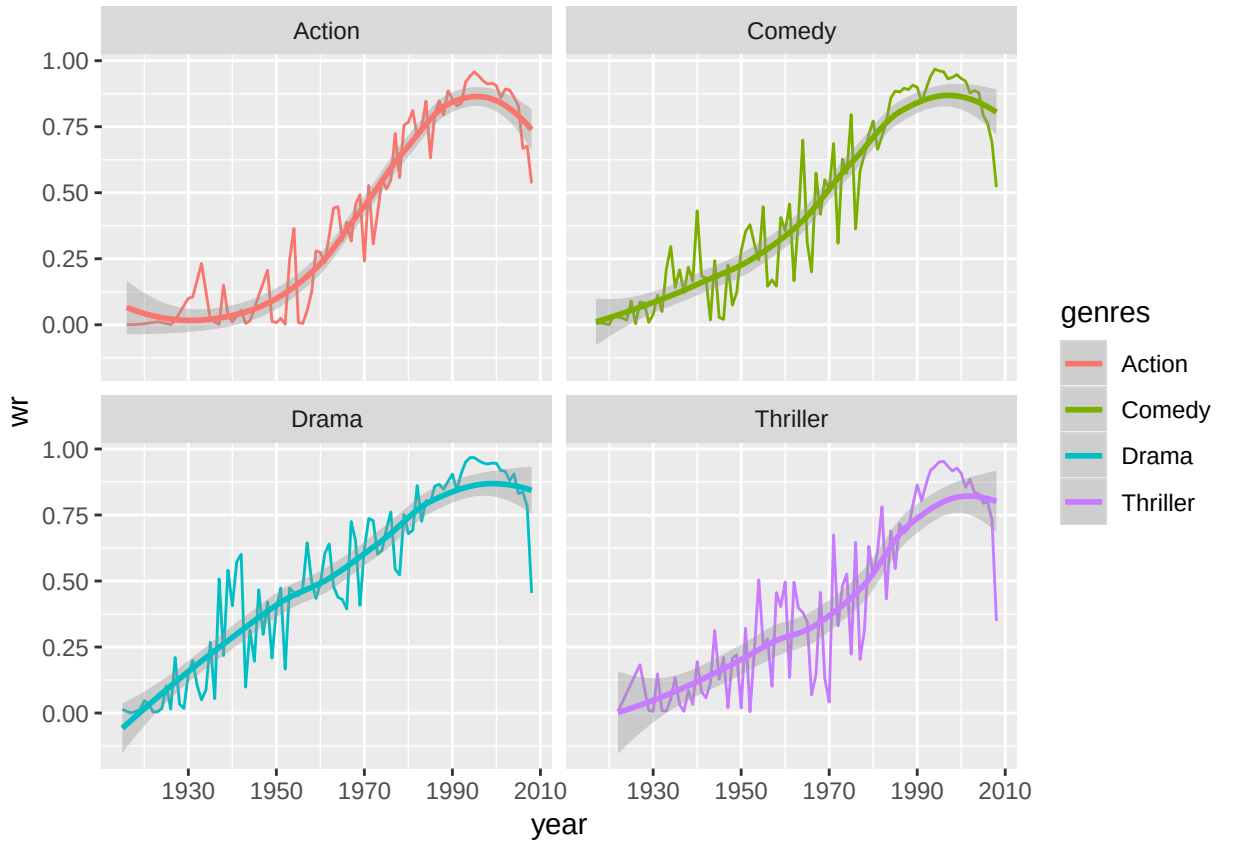
```
##    <dbl> <chr>                                                <dbl> <dbl> <int>
## 1   1910 Birth of a Nation, The                               0.127  3.28    73
## 2   1920 Cabinet of Dr. Caligari, The (Das Cabinet des D~ 0.322  4.04   238
## 3   1930 All Quiet on the Western Front                       0.528  3.99   559
## 4   1940 Pinocchio                                            0.855  3.52  2937
## 5   1950 Cinderella                                           0.782  3.59  1791
## 6   1960 Psycho                                               0.892  4.10  4144
```

We notice the disadvantage of weighted ratings - low score for old movies. That's not necessarily caused by movies quality, rather small number of viewers.

What were the best years for a genre (based on users's ratings)?

```
genresRate <- edx %>%
  na.omit() %>%
  select(everything(), -c(timestamp,userId)) %>%
  mutate(decade = year  %/% 10 * 10) %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(year, genres) %>%
  summarise(count = n(), avg_rating = mean(rating)) %>%
  ungroup() %>%
  mutate(wr = weighted(mean, count, 5000, mean(mean))) %>%
  arrange(year)

genresRate %>%
  filter(genres %in% c("Action", "Comedy", "Drama", "Thriller")) %>%
  ggplot(aes(x = year, y = wr)) +
    geom_line(aes(group=genres, color=genres)) +
    geom_smooth(aes(group=genres, color=genres)) +
    facet_wrap(~genres)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Notice the fact that the rating of the movies across these genres is increasing from year to year.

Does the user has rating consistency when they rate movie? Is there any observeable pattern?

Here we take the top 20 most active user and analyze the trend of their rating

```
#Find the top N most active user
top <- edx %>% select(userId, rating) %>% group_by(userId) %>% summarize(numofrate = n()) %>% arrange(d
```

```
## Selecting by numofrate
```

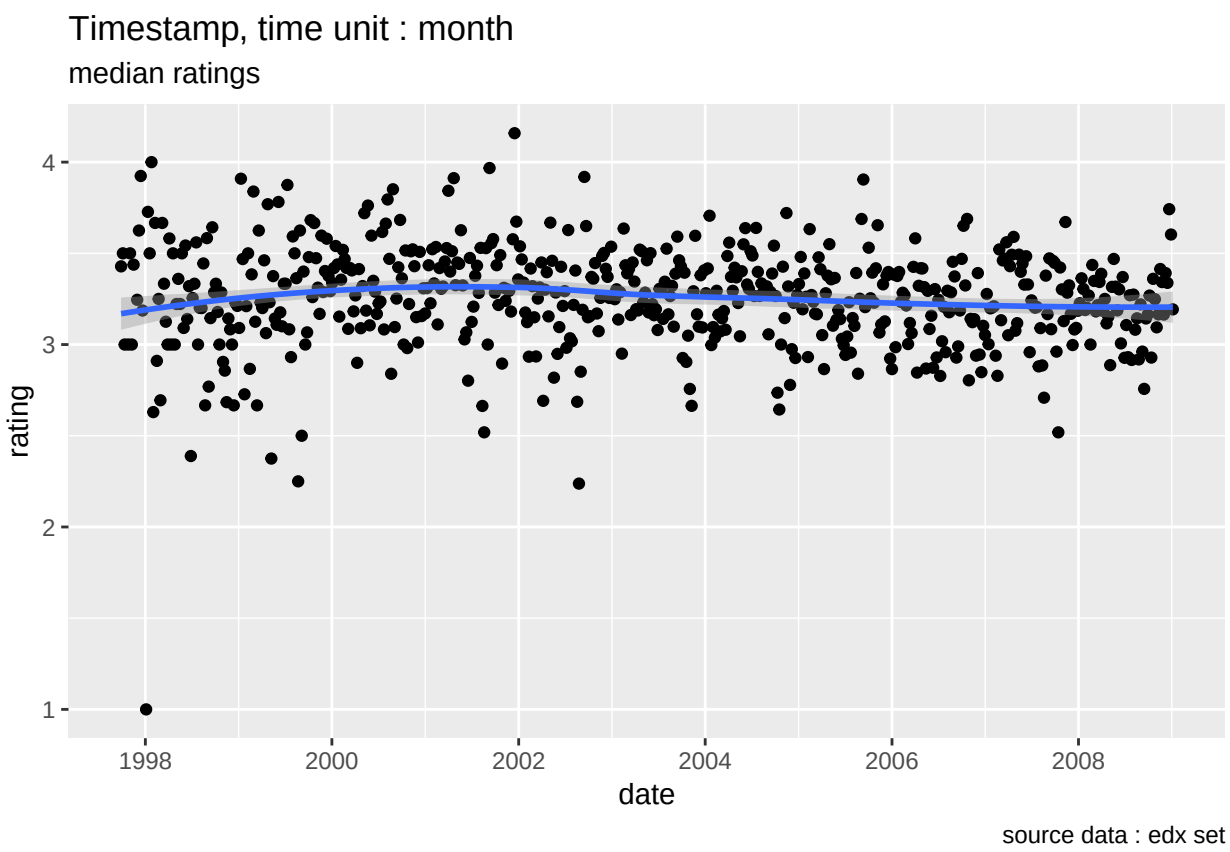
```
print(head(top))
```

```
## # A tibble: 6 x 2
##   userId numofrate
##    <int>     <int>
## 1  59269      2951
## 2  67385      2859
## 3  14463      2043
## 4  68259      1798
## 5  27468      1793
## 6   3817      1670
```

```
edx %>%
  filter(userId %in% top$userId) %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
```

```
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : month")+
  labs(subtitle = "median ratings",
       caption = "source data : edx set")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



As we see from above figure, clearly there's a consistency on rating.It doesn't show a strong effect of time. Notice here instead using the mean we're using the median to calculate the rating consistency.

```
library(stringi)
```

```
## Warning: package 'stringi' was built under R version 3.5.1
```

```
#Extract and add the premier date to the df
edx1 <- edx %>% mutate(premier_date = as.numeric(stri_extract(edx$title, regex = "(\\d{4})", comments =
```

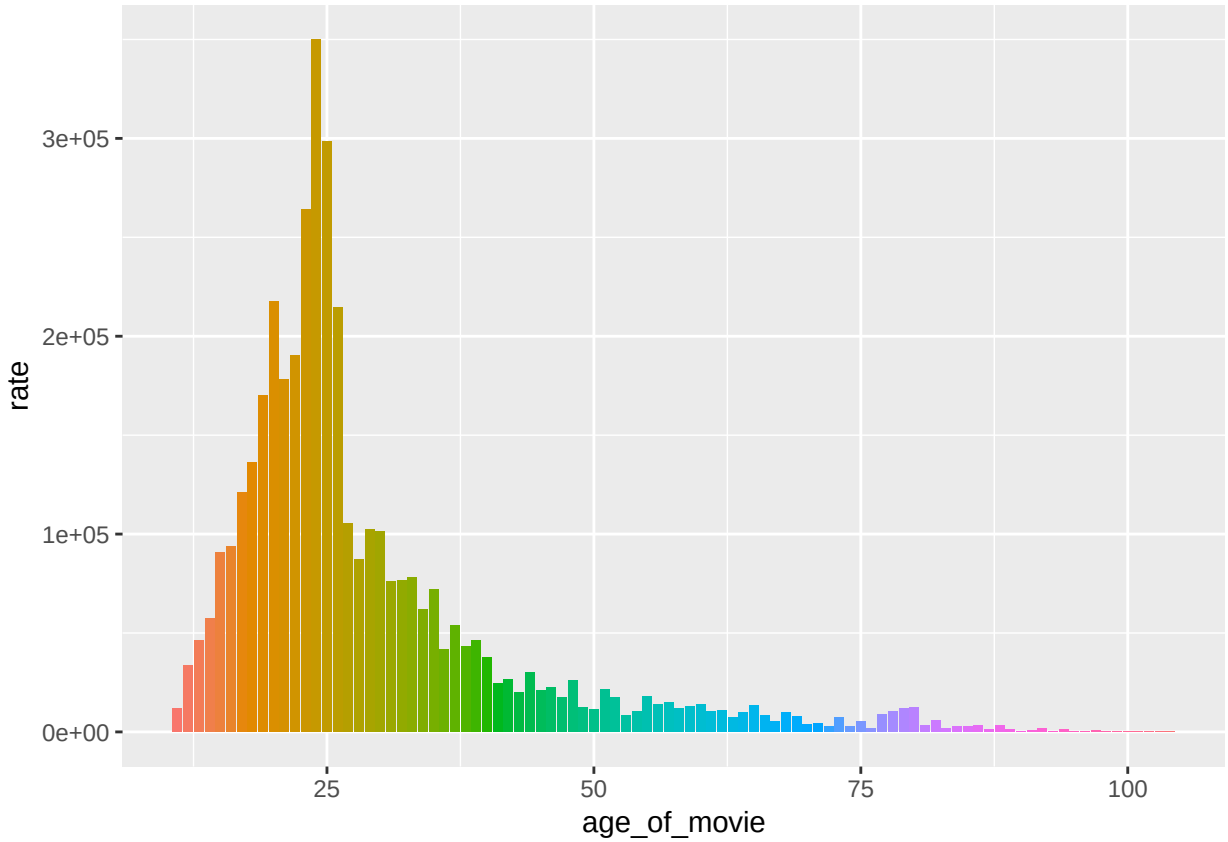
Does the age of the movie affect the given rating?

```
#calculate the age of the movie using this year as pivot as well as the rating date using premier date
edx1 <- edx %>% mutate(yearRated = year(as_datetime(timestamp)))
```

```
edx1<- edx1 %>% mutate(age_of_movie = 2019 - year,
ratingRangeofDate = year - yearRated)
head(edx1)
```

```
##   userId movieId rating  timestamp                    title year
## 1  50805    1095    4.0  952366935       Glengarry Glen Ross 1992
## 2  42011    6303    3.5 1121891132     Andromeda Strain, The 1971
## 3  45985     593    4.0  854490846 Silence of the Lambs, The 1991
## 4  19660    4210    4.0 1207988321                 Manhunter 1986
## 5  46073     186    4.0 1123689113              Nine Months 1995
## 6  13268     485    2.0  923662409          Last Action Hero 1993
##                               genres yearRated age_of_movie
## 1                              Drama      2000           27
## 2                      Mystery|Sci-Fi      2005           48
## 3               Crime|Horror|Thriller      1997           28
## 4 Action|Crime|Drama|Horror|Thriller      2008           33
## 5                      Comedy|Romance      2005           24
## 6    Action|Adventure|Comedy|Fantasy      1999           26
##   ratingRangeofDate
## 1                -8
## 2               -34
## 3                -6
## 4               -22
## 5               -10
## 6                -6
```

```
# group by the age of movie, and calculate its distribution and plot
edx1 %>% select(movieId, age_of_movie, rating) %>% group_by(age_of_movie) %>% summarize(rate = n()) %>%
```



The majority of rating data lies on 25th quartile. It's imply that people tend to rate a movie which has been premiere for less and equal to 25 years, old age movie is left behind. There's an effect caused by the age of movie.

There is some debate about is "Are The Old Movies Better Than The New Ones ?".

We will doing some analysis to answer this questions using question such as does old movies get better ratings than new movies?

```
movAvg <- edx1 %>% group_by(movieId) %>% summarize(avgMovRate = mean(rating))

userAvg <- edx1 %>% group_by(userId) %>% summarize(avgUserRate = mean(rating))

# when is the movie being rated?
yearAvg <- edx1%>% mutate(yearRated = year(as_datetime(timestamp)) ) %>% group_by(yearRated) %>% summari
#age of movie
ageAvgs <- edx1 %>% group_by(age_of_movie) %>% summarize(avgbyAge = mean(rating))
```

```
print(head(ageAvgs))
```

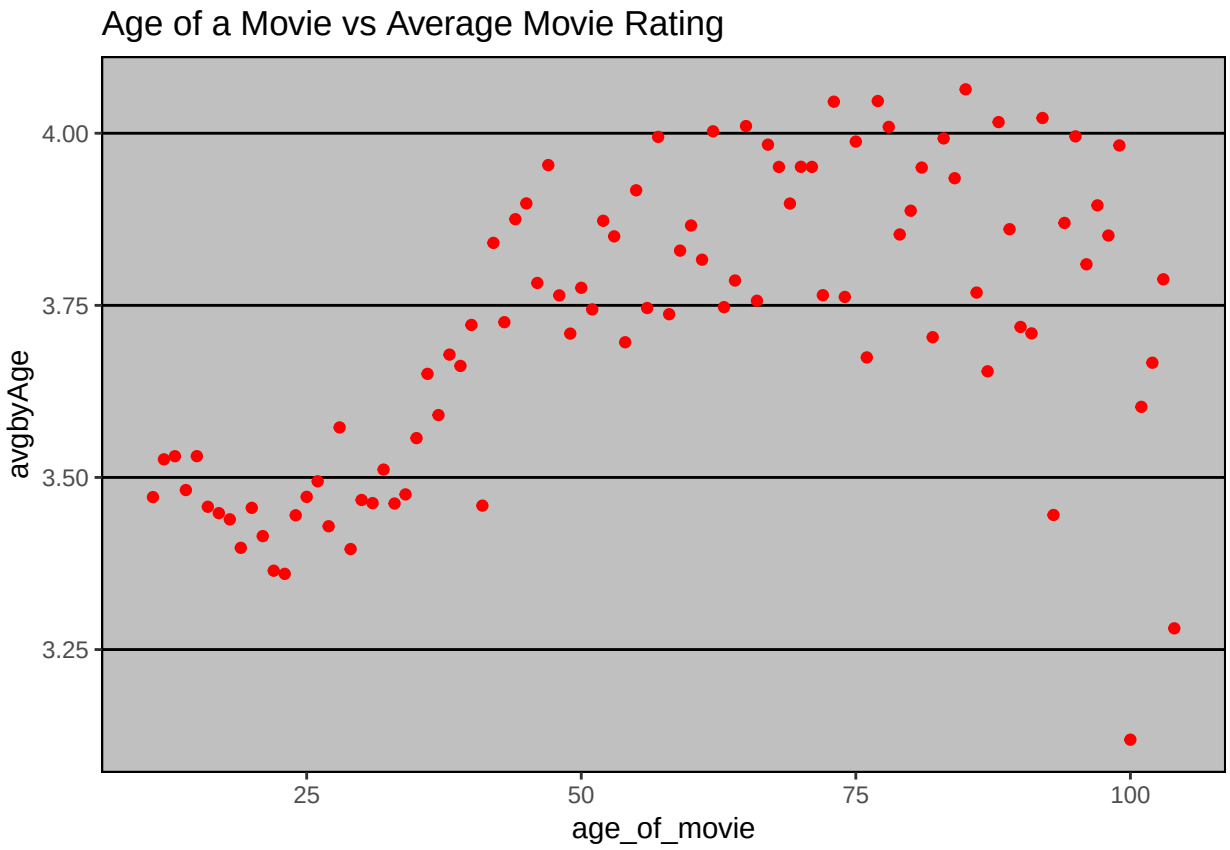
```
## # A tibble: 6 x 2
##   age_of_movie avgbyAge
##          <dbl>    <dbl>
## 1           11     3.47
## 2           12     3.53
## 3           13     3.53
```

```
## 4           14     3.48
## 5           15     3.53
## 6           16     3.46
```

```
print(head(userAvg))
```

```
## # A tibble: 6 x 2
##   userId avgUserRate
##    <int>       <dbl>
## 1      1        5
## 2      2        3.25
## 3      3        4.29
## 4      4        4
## 5      5        4
## 6      6        3.95
```

```
library(ggthemes)
ageAvgs %>%
  ggplot(aes(age_of_movie, avgbyAge)) +
  geom_point(colour="red") +
  theme_excel()+
  ggtitle("Age of a Movie vs Average Movie Rating")
```



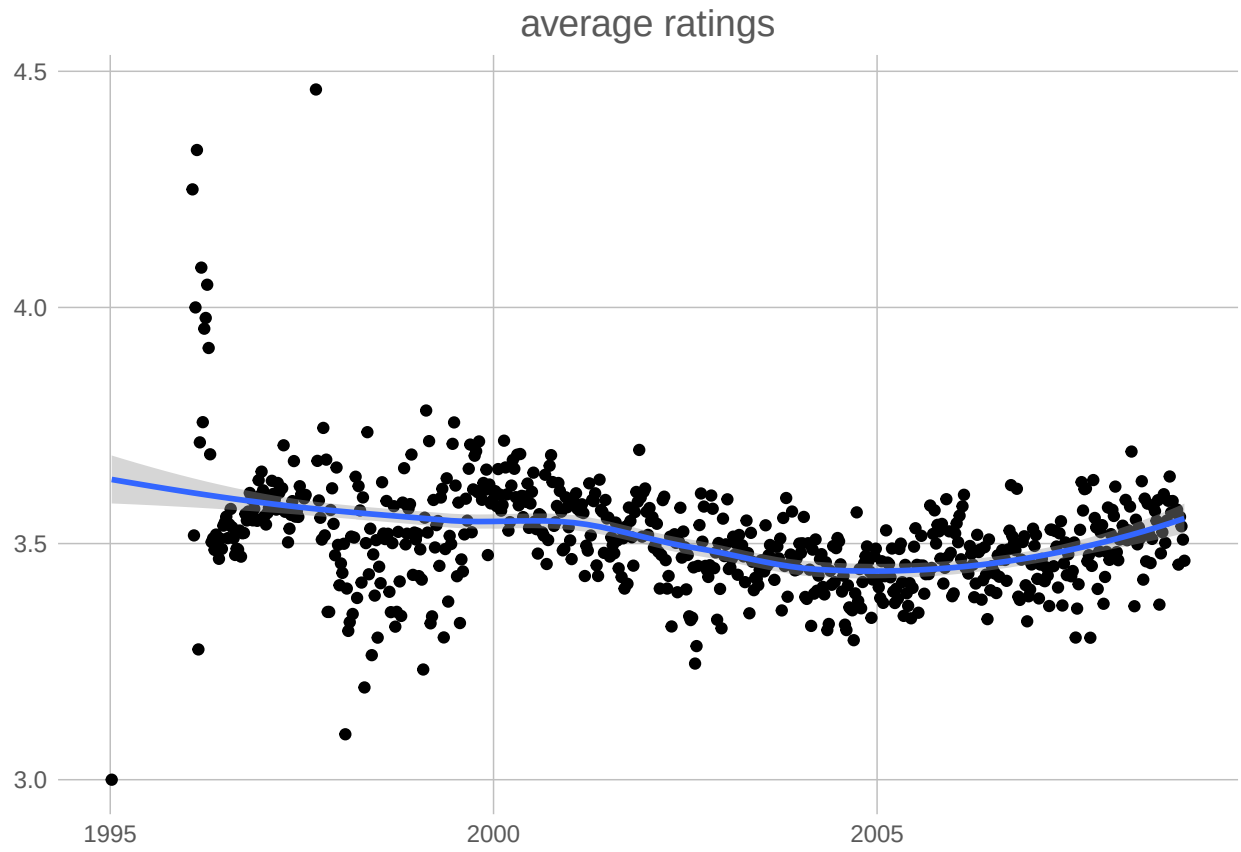
Notice from above, old movies have a higher rating than the new movies, maybe user who doing the rating thinks old movies are more creative than new movies.

Data Processing

Is there any observeable linear pattern from the ratings dataset?

```
p <- edx1 %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point(show.legend = FALSE) +
  geom_smooth(show.legend = FALSE) +
  theme_excel_new()+
  ggtitle("Timestamp, time unit : week")+
  ggtitle("average ratings")

print(p)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
#free memory
rm(edx1)
rm(na_df,t, validation)
```

```
## Warning in rm(na_df, t, validation): object 'na_df' not found
## Warning in rm(na_df, t, validation): object 't' not found
## Warning in rm(na_df, t, validation): object 'validation' not found
```

Is the data sparse?

```
#Calculate how many unique user and movie
edx %>% summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10516
```

It show that there are less movies provided for ratings than users that rated them. From matrix perspective where each row and column represent user and movie respectively, this surely will affect the sparsity of the data since many cells will end up being empty.Somehow, we need to address the fact that not every user has rated every movie which will affect the sparsity of the data.

```
set.seed(12453)
id <- createDataPartition(edx$rating, p = 0.9, list=FALSE)
train <- edx[id,]
validation <- edx[-id,]
```

```
library(dplyr)
edxcopy <- train %>% select(userId, movieId,rating)
#treat movieID and userID as factor  then convert the variables into numeric since sparseMatrix only wo
edxcopy$movieId <- as.factor(edxcopy$movieId)
edxcopy$userId <- as.factor(edxcopy$userId)
edxcopy$movieId <- as.numeric(edxcopy$movieId)
edxcopy$userId <- as.numeric(edxcopy$userId)

# convert to uxm matrix where each row represent user, each column represent movie
sparseRate<- sparseMatrix(i = edxcopy$userId,
                          j = edxcopy$movieId ,
                          x = edxcopy$rating,
                          dims = c(length(unique(edxcopy$userId)),
                                   length(unique(edxcopy$movieId))),
                          dimnames = list(paste("u", 1:length(unique(edxcopy$userId)), sep = ""),
                                          paste("m", 1:length(unique(edxcopy$movieId)), sep = "")))

#free some memory
rm(edxcopy)

sparseRate[1:10,1:10]
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"
##    [[ suppressing 10 column names 'm1', 'm2', 'm3' ... ]]
##
## u1  . . . . . . . . . .
## u2  . . . . . . . . . .
## u3  . . . . . . . . . .
## u4  . . . . . . . . . .
## u5  1 . . . . . . . . .
## u6  . . . . . . . . . .
## u7  . . . . . . . . . .
## u8  . . . . . . . . . .
## u9  . . . . . . . . . .
## u10 . . . . . . . . . .
```

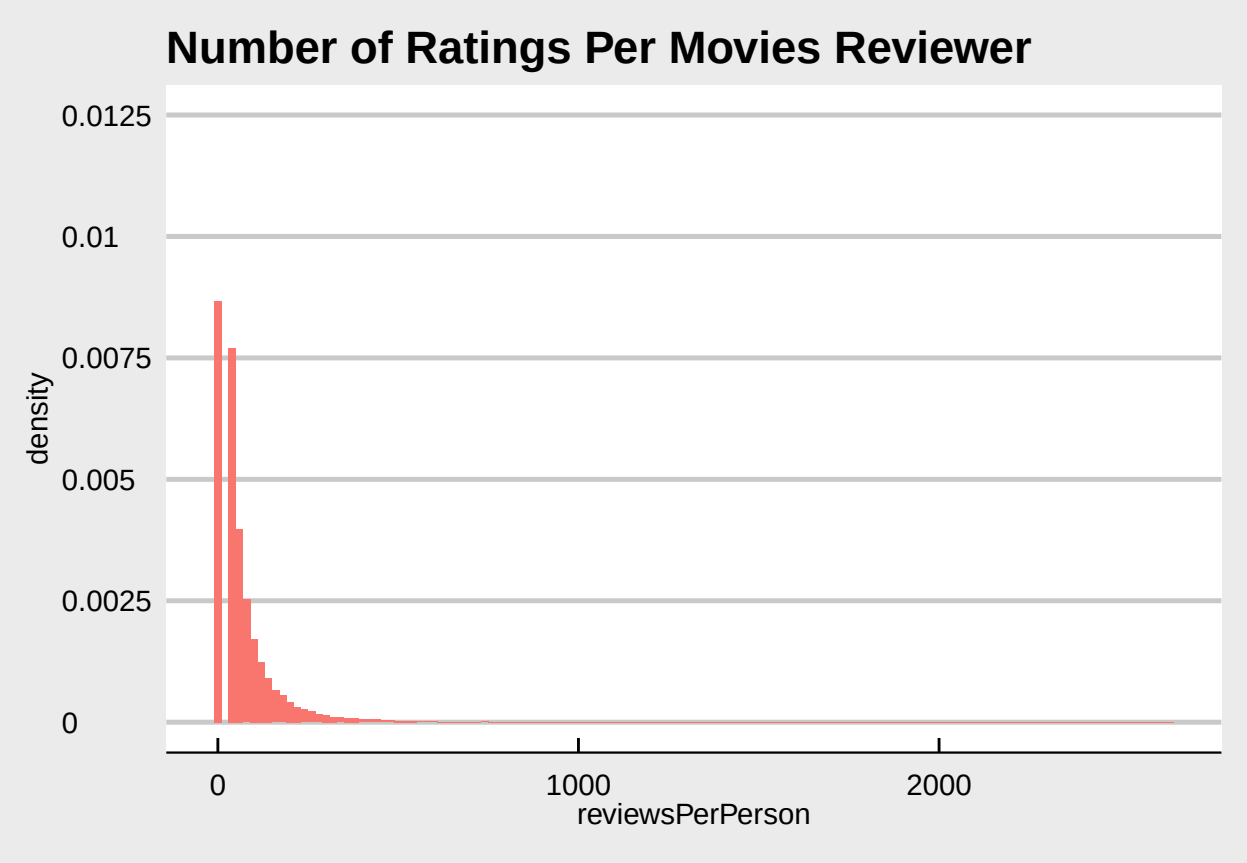
It's necessary to check the distributions of the number of reviews given by each user.The density is plotted along the y-axis instead of the raw counts, to give an idea of the the proportional frequency of each unit of each discrete bin in relation to the whole data set. The overall left-skewed distribution is indicative that most reviewers give very few overall reviews.

```
#convert to matrix for easier calculation
matxrat <- new("realRatingMatrix", data = sparseRate)
rpp <- rowCounts(matxrat) %>%
  data.frame(reviewsPerPerson = .)
p1 <- rpp %>%
  ggplot(aes(x = reviewsPerPerson, fill='#FFA07A')) +
    geom_histogram(aes(y = ..density..), binwidth = 20, show.legend = FALSE) +
    scale_y_continuous(limits = c(0,.0125),
                       breaks = seq(0, .0125, by = 0.0025),
                       labels = seq(0, .0125, by = 0.0025)) +
  theme_economist_white()+
    ggtitle('Number of Ratings Per Movies Reviewer')
print(summary(rowCounts(matxrat)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   13.00   25.00   51.52   56.00 2639.00
```

```
print(p1)
```

```
## Warning: Removed 1 rows containing missing values (geom_bar).
```



Take a look from above, at the average number of ratings given per each of the movie. The left skew distribution shown above indicate that there are a handful of movies with very high reviews, probably

reflecting those films in the dataset with mass commercial appeal and the majority of films in the dataset are scarcely reviewed.

With a median number of reviews of 51 per user and 10474 different movies available to rate, we know that the data is sparse caused by not every user watch every movie.

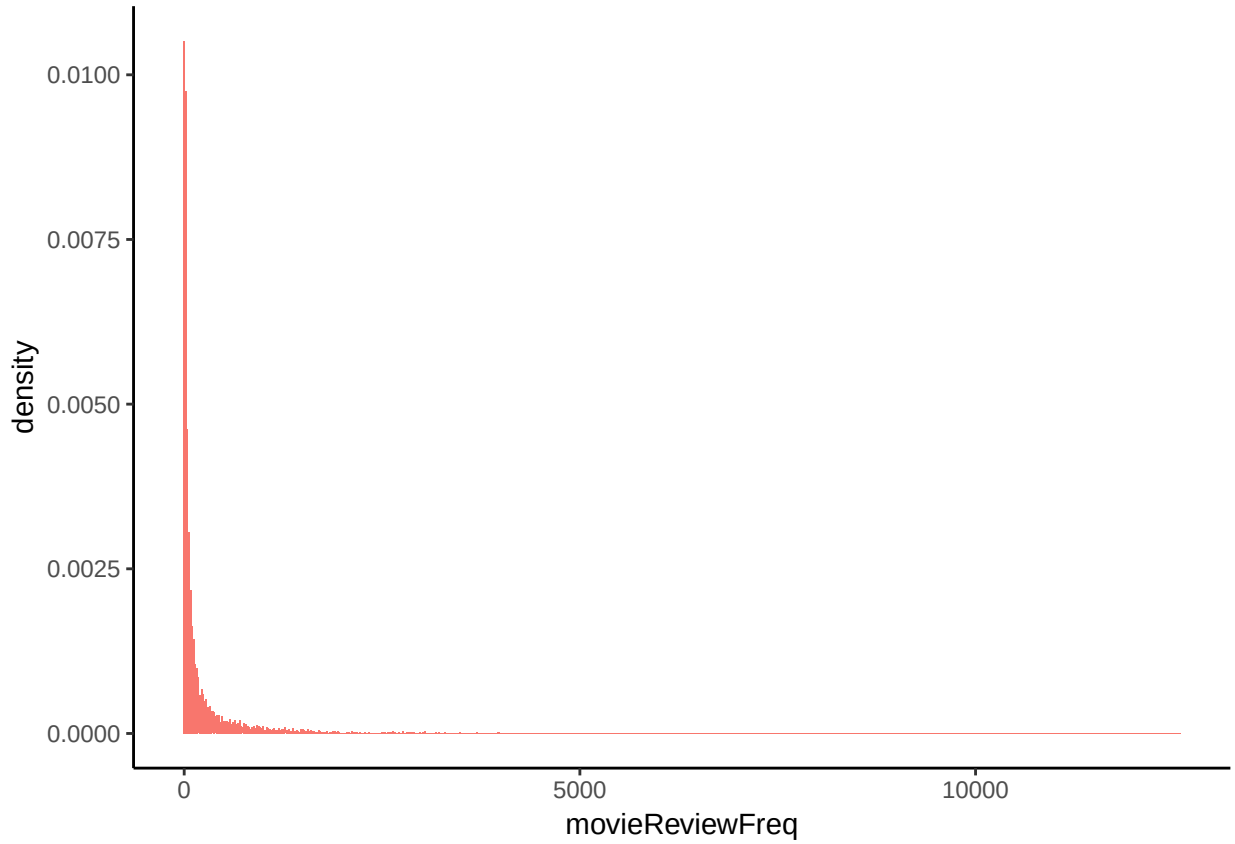
```
mrf <- colCounts(matxrat) %>%
  data.frame(movieReviewFreq = .) %>%
  ggplot(aes(x = movieReviewFreq)) +
    geom_histogram(aes(y = ..density.., fill = 'blue'),show.legend = FALSE, binwidth = 20) +theme_class
    ggtitle('Number of Reviews Per MovieLense listed Movie')
```

```
## $title
## [1] "Number of Reviews Per MovieLense listed Movie"
##
## attr(,"class")
## [1] "labels"
```

```
print(summary(colCounts(matxrat)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    13.0    51.0   343.7   236.8 12578.0
```

```
print(mrf)
```



Data Transformation

Now, let's transform our data into a rating matrix which we will used as an input in recommender labs package

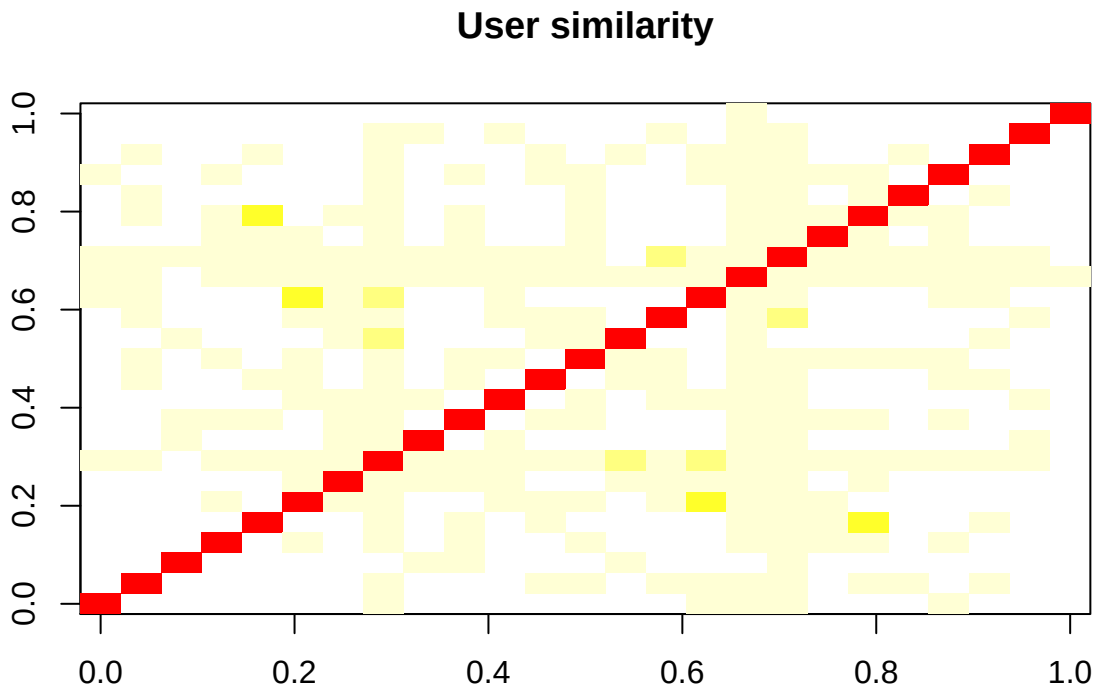
```
#Convert rating matrix into a recommenderlab sparse matrix
matxrat <- new("realRatingMatrix", data = sparseRate)
matxrat
```

```
## 69878 x 10474 rating matrix of class 'realRatingMatrix' with 3600051 ratings.
```

Notice that now we had a pretty huge matrix due to how big our dataset is. This will lead to a memory issues. By reducing the dimensionality using Matrix package the data processing will reduce the cost of time and deal more efficiently with the memory problem.

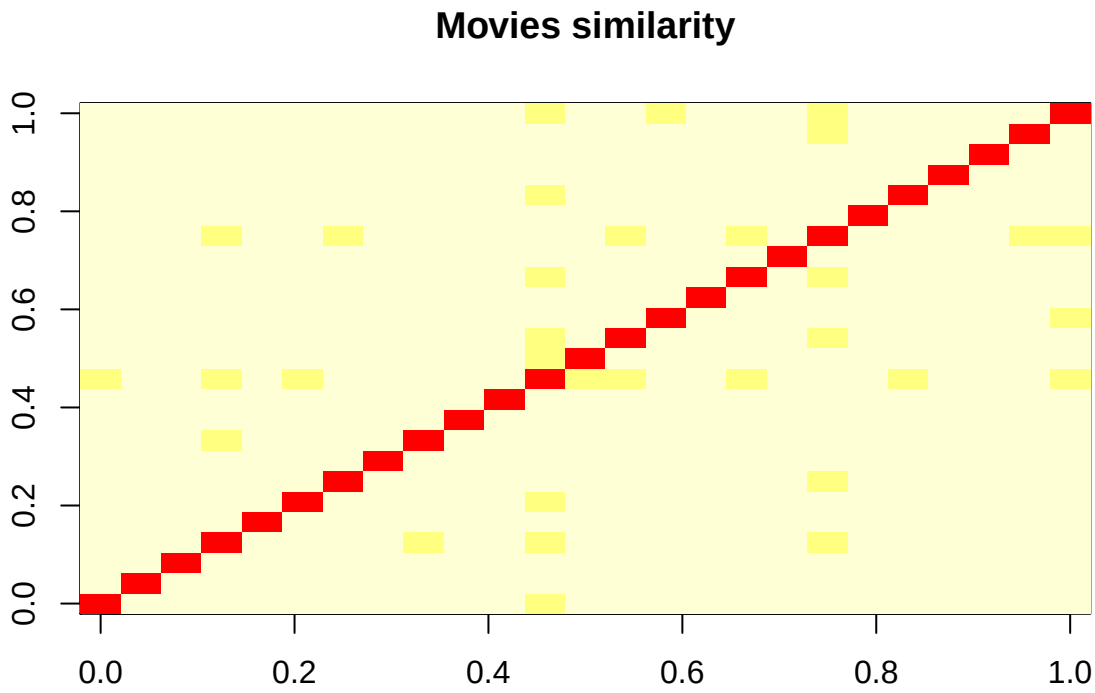
Dimensionality reduction could be done by observing is there any similar user or movies that have the same rating characteristics towards each other. Let's see the similarity for each user and movies, for readibilily we would only take and calculate the similarity of the first 25 users and movies.

```
#using cosine to calculate user-user similarity
simuser <- similarity(matxrat[1:25,],
                                method = "cosine",
                                which = "users")

image(as.matrix(simuser), main = "User similarity")
```




```
#using cosine to calculate movies-movies similarity

simmov <- similarity(matxrat[,1:25],
```

```
                                  method = "cosine",
                                  which = "items")

image(as.matrix(simmov), main = "Movies similarity")
```



Notice two things: 1. Each rows and columns represent user-user as well for the movies. 2. The red diagonal indicate the similarity between the user and itself, it's comparing to him/herself.

There are evidences the existence of a group of user and movies.Since there are more similar ratings between certain users than others, and more similar ratings between certain movies than others.

Addresssing the RAM memory problem, the Irlba package is used, which it is a fast and memory-efficient way to compute a partial SVD. The augmented implicitly restarted Lanczos bidiagonalization algorithm (IRLBA) finds a few approximate largest (or, optionally, smallest) singular values and corresponding singular vectors of a sparse or dense matrix using a method of Baglama and Reichel. We're going to use Matrix factorization[https://rafalab.github.io/dsbook/matrix-factorization.html] which decompose the I x J matrix M with J < I as follow:

$$M = UVD^T$$

into three matrices,

U : orthogonal matrix of dimensions N x m D : diagonal matrix containing the singular values of the original matrix, m x m V : orthogonal matrix of dimensions m x P

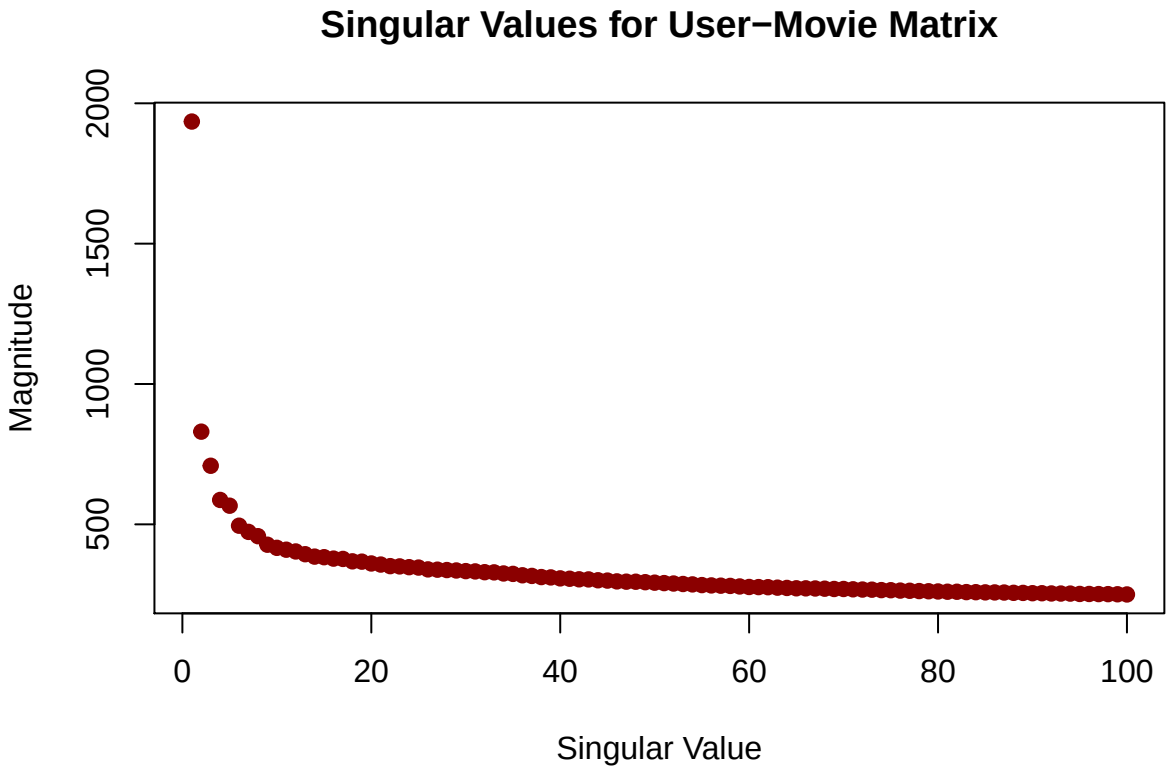
```
set.seed(1)
X <- irlba(sparseRate,tol=1e-4,verbose=TRUE,nv = 100, maxit = 1000)
```

```
## Working dimension size 107
```

```
## Initializing starting vector v with samples from standard normal distribution.
## Use `set.seed` first for reproducibility.
```

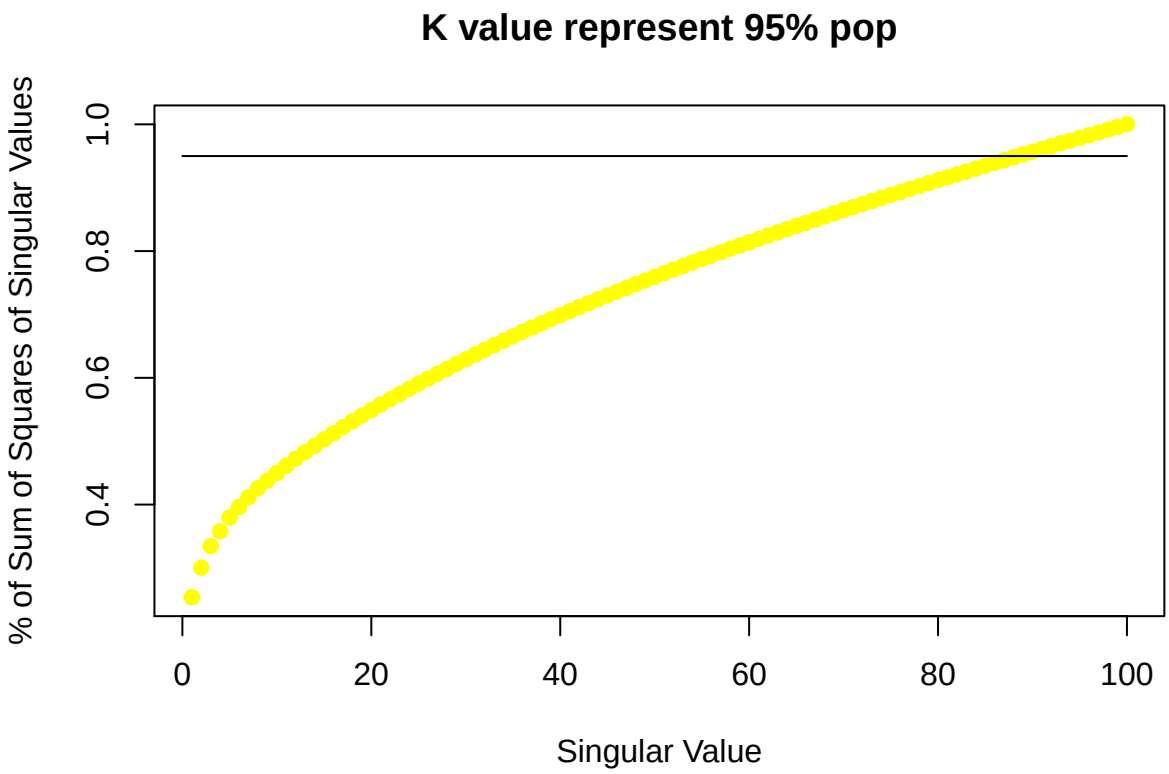
```
## irlba: using fast C implementation
```

```
# plot singular values

plot(X$d, pch=20, col = "darkred", cex = 1.5, xlab='Singular Value', ylab='Magnitude',
     main = "Singular Values for User-Movie Matrix")
```

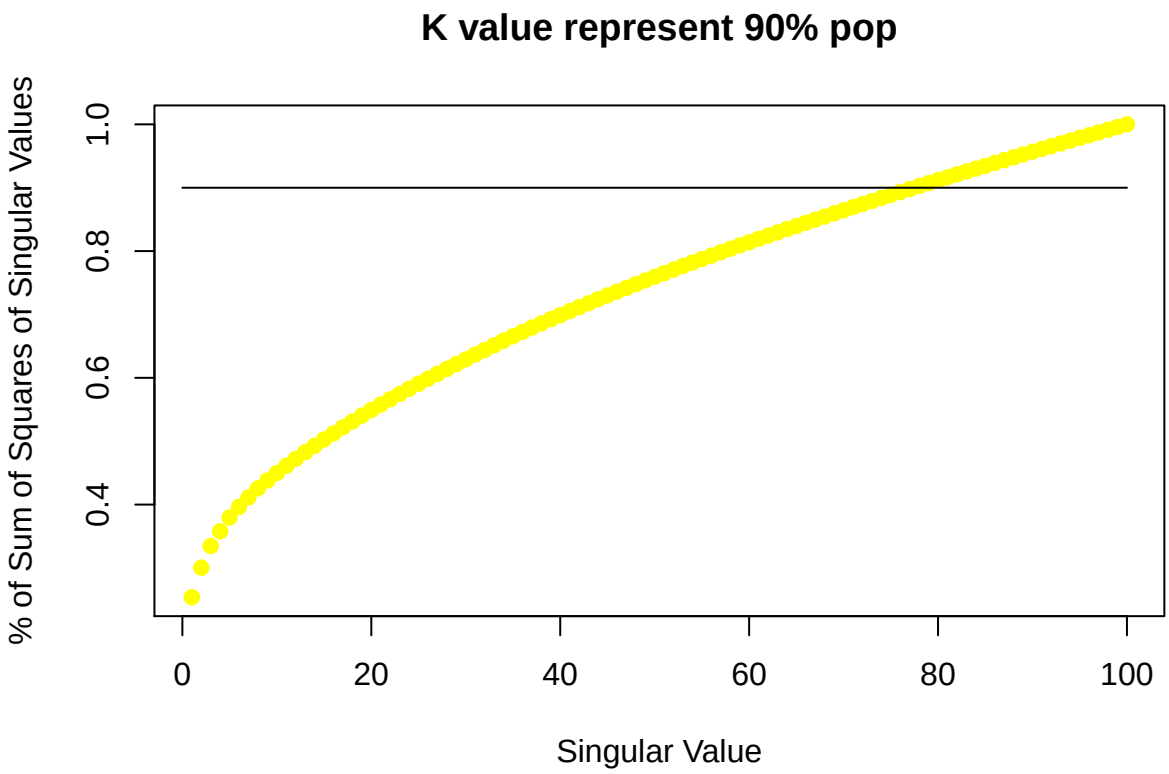


We try to find the optimal range of k suing trial and error method. We want the a k value which will represent 90-95% of population of our ratings set.

```
set.seed(2)
# calculate sum of squares of all singular values
singularSQ <- sum(X$d^2)
v <- NULL
for (i in 1:length(X$d)) {
  v[i] <- sum(X$d[1:i]^2) / singularSQ
}

plot(v, pch=20, col = "yellow", cex = 1.5, xlab='Singular Value', ylab='% of Sum of Squares of Singular
lines(x = c(0,100), y = c(.95, .95))
```




```
plot(v, pch=20, col = "yellow", cex = 1.5, xlab='Singular Value', ylab='% of Sum of Squares of Singular
lines(x = c(0,100), y = c(.90, .90))
```



The goal is to identify the optimal value of K which represent our rating matrix in other words whose squares sum to who lies between 90%-95% of the total of the sums of the squares of all of the singular values. From above plot we can see k ranging from 76-79 for 90% and 80-89 for 95%. To get the exact value of k for each of them we done the following

```
#calculate  the length of the vector that remains from our sum of squares, including any items within t
k90 = length(v[v<= 0.90])
k95 = length(v[v<= 0.95])
print(k90)
```

```
## [1] 77
```

```
print(k95)
```

```
## [1] 88
```

```
library(diagonals)
```

```
## Warning: package 'diagonals' was built under R version 3.5.3
```

```
##
## D I
## A G
##     O N
##     A L
##         S
```

```
u90 <- dim(X$u[, 1:k90])
d90 <- dim(Diagonal(x = X$d[1:k90]))
v90 <- dim(t(X$v)[1:k90, ])
```

```
print(dim(X$u[, 1:k90]))
```

```
## [1] 69878    77
```

```
print(dim(Diagonal(x = X$d[1:k90])))
```

```
## [1] 77 77
```

```
print(dim(t(X$v)[1:k90, ]))
```

```
## [1]    77 10474
```

```
u95 <- dim(X$u[, 1:k95])
d95 <- dim(Diagonal(x = X$d[1:k95]))
v95 <- dim(t(X$v)[1:k95, ])

print(dim(X$u[, 1:k95]))
```

```
## [1] 69878    88
```

```
print(dim(Diagonal(x = X$d[1:k95])))
```

```
## [1] 88 88
```

```
print(dim(t(X$v)[1:k95, ]))
```

```
## [1]    88 10474
```

```
ttl90 <- u90[1]*u90[2]+d90[1]*d90[2]+v90[1]*v90[2]
ttl95 <- u95[1]*u95[2]+d95[1]*d95[2]+v95[1]*v95[2]

print(ttl90)
```

```
## [1] 6193033
```

```
print(ttl95)
```

```
## [1] 7078720
```

We notice that k=6193033 will retain 90% of variability which has total required size of matrix . While for k=7078720 for 95% variability, end up with size of matrix equal to . The size of the matrix originally is `r(69878*10516)` For k=77, we decrease into 0.8427764, and 0.9633047 for k=88.

The difference between size is 12.0528354, while the difference for k value is 11. It means the difference between k-value, i.e. 11 is equal to the addition of 1 Million rows. The cost is high, hence we select k equal to k=6193033.

Intuitively we know some users have been more ACTIVE than others also some movies that has been rated by many users. We'll use this to filter our user-movies ratings matrix.

```
#minimum num of movie per user
movperuser <- quantile(rowCounts(matxrat), 0.9)
#minimum num of user per movie
userpermov <- quantile(colCounts(matxrat), 0.9)
#filter using above condition
matxrat <- matxrat[rowCounts(matxrat) > movperuser,colCounts(matxrat) > userpermov]

print(' minimum number of movies per user')
```

```
## [1] " minimum number of movies per user"
```

```
print(movperuser)
```

```
## 90%
## 121
```

```
print('minimum number of users per movie')
```

```
## [1] "minimum number of users per movie"
```

```
print(userpermov)
```

```
## 90%
## 882
```

```
print('matrix contains users and movies matching these criteria')
```

```
## [1] "matrix contains users and movies matching these criteria"
```

```
print(matxrat)
```

```
## 6949 x 1047 rating matrix of class 'realRatingMatrix' with 916873 ratings.
```

Now, we ended up with a rating matrix of 6949 distinct users (rows) x 1047 distinct movies(columns) , with 7275603 ratings.

Training, Testing Method and Analysis

We use the following approach to build our model, we know that: 1. Some of the movies had a higher rating compare to the others. 2. Some users are more active than others at rating movies. 3. From earlier, there are some rating fluctuations caused by time effect.

Regression Models

```
#calculate mean of the rating dataset
mu <- mean(train$rating)
movieav <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

userav <- train %>%
  left_join(movieav, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))


# calculate time affect from the training set
tmp <- train %>%
  left_join(movieav, by='movieId') %>%
  left_join(userav, by='userId') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))


#tmp for date feature creation from timestamp at validation dataset
valid <- validation
```

```
valid <- valid %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))

#use model from training and applying to validation set
pred <- valid %>%
  left_join(movieav, by='movieId') %>%
  left_join(userav, by='userId') %>%
  left_join(tmp, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  .$pred

#calculate the rmse of validation after applying the model from train set
#note that I using sampling at the begining, therefore there must be some rows which doesn't exist in t
df <- cbind(pred, validation$rating) %>% as_tibble() %>% drop_na()
```

```
## Warning: `as_tibble.matrix()` requires a matrix with column names or a `.name_repair` argument. Using
## This warning is displayed once per session.
```

```
colnames(df) <- c('pred', 'act')
rmseLin <- RMSE(df$pred,df$act)
print(rmseLin)
```

```
## [1] 0.8729142
```

Even though we use the combination of movie, user, and time effects, we still end up with a rmse about 0.8729142.

Now we've stored the rmse, let's take a look for the first 40 prediction.

```
df %>% head(40)
```

```
## # A tibble: 40 x 2
##     pred   act
##    <dbl> <dbl>
##  1  3.43   3.5
##  2  4.16   4
##  3  3.47   4
##  4  4.40   4
##  5  4.25   4
##  6  4.01   4
##  7  4.21   4
##  8  2.52   3
##  9  2.75   2.5
## 10  3.54   4
## # ... with 30 more rows
```

Matrix Factorization with SGD

The purpose of SGD is to get the right gradient on average for much less computation, by using only uses only a single example (a batch size of 1) per iteration. Given enough iterations, SGD works but is very noisy. The term "stochastic" indicates that the one example comprising each batch is chosen at random [https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent]

While the use of matrix factorization work as follow: Matrix P represents latent factors of users. So, each k-elements column of matrix P represents each user. Each k-elements column of matrix Q represents each item . So, to find rating for item i by user u we simply need to compute two vectors: P[,u] x Q[,i].[http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf.]

```
#clear unused memory
invisible(gc())

edxcopy <-  train %>%
            select(-c("genres","title","timestamp"))
names(edxcopy) <- c("user", "item", "rating")
edxcopy <- as.matrix(edxcopy)

validcopy <-  validation %>%
  select(-c("genres","title","timestamp"))
names(validcopy) <- c("user", "item", "rating")
validcopy <- as.matrix(validcopy)

#backup the copy and write it on the disk
write.table(edxcopy , file = "edxset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
write.table(validcopy, file = "validset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
```

```
edx_set <- data_file(file.path("edxset.txt"))
valid_set <- data_file(file.path("validset.txt"))

r <-  Reco()
```

```
set.seed(564627) # for reproducible
# costp_l1 Tuning parameter, the L1 regularization cost for user factors. Can be specified as a numeric
#costq_l1 Tuning parameter, the L1 regularization cost for item factors. Can be specified as a numeric
opt <- r$tune(edx_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                   costp_l1 = 0, costq_l1 = 0,
                                   nthread = 1, niter = 10))
```

```
r$train(edx_set, opts = c(opt$min, nthread = 1, niter = 80))
```

```
## iter      tr_rmse          obj
##    0       1.0908  5.9576e+006
##    1       0.8934  4.3000e+006
##    2       0.8728  4.0650e+006
##    3       0.8474  3.8437e+006
##    4       0.8279  3.6842e+006
##    5       0.8104  3.5578e+006
##    6       0.7949  3.4550e+006
##    7       0.7805  3.3655e+006
##    8       0.7677  3.2897e+006
##    9       0.7557  3.2240e+006
##   10       0.7446  3.1636e+006
##   11       0.7349  3.1132e+006
##   12       0.7259  3.0683e+006
##   13       0.7179  3.0282e+006
##   14       0.7106  2.9923e+006
##   15       0.7040  2.9606e+006
##   16       0.6981  2.9314e+006
##   17       0.6928  2.9065e+006
##   18       0.6879  2.8840e+006
##   19       0.6836  2.8629e+006
##   20       0.6794  2.8432e+006
##   21       0.6758  2.8263e+006
##   22       0.6723  2.8115e+006
```

```
##   23       0.6692  2.7958e+006
##   24       0.6663  2.7837e+006
##   25       0.6635  2.7709e+006
##   26       0.6610  2.7598e+006
##   27       0.6585  2.7487e+006
##   28       0.6563  2.7388e+006
##   29       0.6542  2.7299e+006
##   30       0.6522  2.7206e+006
##   31       0.6504  2.7132e+006
##   32       0.6486  2.7056e+006
##   33       0.6469  2.6982e+006
##   34       0.6453  2.6913e+006
##   35       0.6439  2.6852e+006
##   36       0.6424  2.6796e+006
##   37       0.6410  2.6738e+006
##   38       0.6397  2.6682e+006
##   39       0.6384  2.6624e+006
##   40       0.6372  2.6588e+006
##   41       0.6360  2.6537e+006
##   42       0.6349  2.6492e+006
##   43       0.6338  2.6450e+006
##   44       0.6328  2.6410e+006
##   45       0.6318  2.6374e+006
##   46       0.6309  2.6342e+006
##   47       0.6299  2.6297e+006
##   48       0.6290  2.6268e+006
##   49       0.6281  2.6234e+006
##   50       0.6273  2.6200e+006
##   51       0.6265  2.6174e+006
##   52       0.6257  2.6143e+006
##   53       0.6249  2.6119e+006
##   54       0.6241  2.6089e+006
##   55       0.6235  2.6068e+006
##   56       0.6227  2.6043e+006
##   57       0.6220  2.6013e+006
##   58       0.6214  2.5991e+006
##   59       0.6207  2.5967e+006
##   60       0.6201  2.5951e+006
##   61       0.6195  2.5926e+006
##   62       0.6189  2.5909e+006
##   63       0.6182  2.5886e+006
##   64       0.6177  2.5868e+006
##   65       0.6171  2.5846e+006
##   66       0.6166  2.5828e+006
##   67       0.6160  2.5816e+006
##   68       0.6155  2.5794e+006
##   69       0.6149  2.5778e+006
##   70       0.6144  2.5763e+006
##   71       0.6139  2.5745e+006
##   72       0.6134  2.5731e+006
##   73       0.6130  2.5714e+006
##   74       0.6125  2.5704e+006
##   75       0.6120  2.5687e+006
##   76       0.6116  2.5674e+006
```

```
##   77       0.6111  2.5663e+006
##   78       0.6106  2.5643e+006
##   79       0.6102  2.5630e+006
```

apply the model to the validation set to make prediction

```
pred_file = tempfile()
r$predict(valid_set, out_file(pred_file))
```

```
## prediction output generated at C:\Users\LB3\AppData\Local\Temp\RtmpwzFyvw\file16f037fe7e3a
```

How about it's RMSE?

```
scores_real <- read.table("validset.txt", header = FALSE, sep = " ")$V3
scores_pred <- scan(pred_file)

#free some memory
rm(edxcopy, validcopy)

rmseVal <- RMSE(scores_real,scores_pred)
rmseVal
```

```
## [1] 0.8416904
```

Performing the Matrix factorization with SGD recommender method, we achieved a RMSE .

Now we've stored the rmse, let's take a look for the first 40 prediction.

```
dfpred <- cbind(scores_pred, scores_real)
colnames(dfpred) <- c('pred', 'actual')
dfpred %>% head(40)
```

```
##          pred actual
##  [1,] 3.14271    3.5
##  [2,] 4.21501    4.0
##  [3,] 3.27954    4.0
##  [4,] 4.44979    4.0
##  [5,] 4.83377    4.0
##  [6,] 4.05699    4.0
##  [7,] 3.69810    4.0
##  [8,] 2.40023    3.0
##  [9,] 1.85722    2.5
## [10,] 3.48505    4.0
## [11,] 4.40681    5.0
## [12,] 4.42376    3.0
## [13,] 3.95381    4.0
## [14,] 4.69908    5.0
## [15,] 4.09803    3.5
## [16,] 4.28997    4.5
## [17,] 3.18262    3.0
## [18,] 2.94914    3.0
## [19,] 2.20032    0.5
## [20,] 3.38960    3.0
## [21,] 3.49887    3.0
## [22,] 2.36863    2.0
## [23,] 3.61162    3.0
```

```
## [24,] 4.14210    2.5
## [25,] 2.86103    3.5
## [26,] 2.70381    3.0
## [27,] 4.07971    5.0
## [28,] 3.67812    4.5
## [29,] 1.09874    1.0
## [30,] 2.24005    2.0
## [31,] 2.71280    3.0
## [32,] 4.39295    3.5
## [33,] 2.51729    2.0
## [34,] 4.01898    4.0
## [35,] 3.44499    2.5
## [36,] 2.86042    4.0
## [37,] 3.87096    4.0
## [38,] 2.08225    1.0
## [39,] 4.60854    4.0
## [40,] 3.47104    4.0
```

Conclusion

The objectives of the project is to predict movie ratings for the 10M version of the Movielens data. The training set is provided (90% from edx dataset) while the validation set is taken about 10% from edx dataset, we had try using different models, such as regression and Matrix factorization with SGD.

The model evaluation performance through the RMSE (root mean squared error) showed that the between Linear regression model with effects on users and movies and time, and the Matrix Factorization with Stochastic gradient descent (SGD), the SGD is perform bettter than the regression models, i.e. $0.8416904 < 0.8729142$.