01 Capstone Movielens Dataset pdf

Mirna Rossi

2022-09-12

Introduction/overview

The MovieLens dataset has been published for the first time on the MovieLens website in 1998. Since then, it has been downloaded several times by users because of its utility in various sectors, especially for educational purposes. This dataset can be used to explore movie rating systems or it can be adapted for assessing on the many popular ratings systems other companies use these days. The analysis of this dataset opens further questions about the factors that influence user's decisions and to future approaches (Harper & Konstan, 2015). Once ratings and movies are merged, the dataset shows the following variables: userId, movieId, rating, timestamp, title, genres; year is included in the title. My analysis has been performed using both Linux and Windows OS. The goal of this project is to create an original analysis providing movie predictions after using the dataset. As per "MovieLens Grading Rubric", accuracy will be measured after providing an RMSE < 0.86490.

Methods/analysis including our modeling approach (we must provide at least 2 models).

Exploratory data analysis (EDA) is the "human" intervention to the dataset and it is fundamental to get users familiar with data. It sometimes require removal of missing or incorrect data, or changes to make observations more workable (Theobald, 2017, p. 36). In my analysis I start with the initial code provided by the course instructions in the "Create train and validation sets" section (attached in Appendix). Here we install the packages required, we download a temporary file containing 2 datasets, we merge them, we create train and test dataset paying attention that the validation dataset must be 10% of the original file. As we have seen, the Movielens dataset contains a lot of observations and it takes time to load or process coding. For this reason and to attempt the many tests this analysis requires, I have printed the temporary file in csv (named edx.csv); the dataset I have used is available for download on GitHub.

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(visreg)) install.packages("visreg", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: visreg
```

```r
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
if(!require(readxl)) install.packages("readxl", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: readxl
```

```r
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(plotly)) install.packages("plotly", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: plotly
##
## Attaching package: 'plotly'
##
## The following object is masked from 'package:ggplot2':
##
##     last_plot
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lubridate
##
## 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 objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
if(!require(tidytext)) install.packages("tidytext", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidytext
```

```r
if(!require(textrecipes)) install.packages("textrecipes", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: textrecipes
## Loading required package: recipes
##
## Attaching package: 'recipes'
##
## The following object is masked from 'package:stringr':
##
##     fixed
##
## The following object is masked from 'package:stats':
##
##     step
```

```r
if(!require(textfeatures)) install.packages("textfeatures", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: textfeatures
```

```r
if(!require(LiblineaR)) install.packages("LiblineaR", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: LiblineaR
```

```r
if(!require(doParallel)) install.packages("doParallel", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: doParallel
## Loading required package: foreach
##
## Attaching package: 'foreach'
##
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: iterators
## Loading required package: parallel
```

```
if(!require(vip)) install.packages("vip", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: vip
##
## Attaching package: 'vip'
##
## The following object is masked from 'package:utils':
##
##     vi
```

```
if(!require(skimr)) install.packages("skimr", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: skimr
```

```
theme_set(theme_classic())

#Load the packages required (in Linux #use readxl instead of xlsx).
library(visreg)
library(readr)
library(tidyverse)
library(caret)
library(data.table)
library(readxl)
library(dplyr)
library(ggplot2)
library(plotly)
library(lubridate)
library(tidytext)
library(textrecipes)
library(textfeatures)
library(LiblineaR)
library(doParallel)
library(vip)
library(skimr)
```

Preparation of the datasets

```
ratings <- read.csv("ratings.csv")
head(ratings)
```

```
##   X userId movieId rating timestamp
## 1 1      1     122      5 838985046
## 2 2      1     185      5 838983525
## 3 3      1     231      5 838983392
```

```
## 4 4      1     292      5 838983421
## 5 5      1     316      5 838983392
## 6 6      1     329      5 838983392
```

```
str(ratings)
```

```
## 'data.frame':    10000054 obs. of  5 variables:
##  $ X        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
```

```
movies <- read_csv("movies.csv")
```

```
## New names:
## Rows: 10681 Columns: 4
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (2): title, genres dbl (2): ...1, movieId
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

```
head(movies)
```

```
## # A tibble: 6 x 4
##    ...1 movieId title                              genres
##   <dbl>   <dbl> <chr>                              <chr>
## 1     1       1 Toy Story (1995)                   Adventure|Animation|Children~
## 2     2       2 Jumanji (1995)                     Adventure|Children|Fantasy
## 3     3       3 Grumpier Old Men (1995)            Comedy|Romance
## 4     4       4 Waiting to Exhale (1995)           Comedy|Drama|Romance
## 5     5       5 Father of the Bride Part II (1995) Comedy
## 6     6       6 Heat (1995)                        Action|Crime|Thriller
```

```
class(movies)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
#check if the object is a data frame and coerce as a dataframe if necessary, then use mutate
#to change existing variables
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

#join the datasets then create edx.csv file
edx <- left_join(ratings, movies, by = "movieId")
head(edx)
```

```
##   X userId movieId rating timestamp ...1                         title
## 1 1      1     122      5 838985046  121              Boomerang (1992)
## 2 2      1     185      5 838983525  184               Net, The (1995)
## 3 3      1     231      5 838983392  229          Dumb & Dumber (1994)
## 4 4      1     292      5 838983421  290               Outbreak (1995)
## 5 5      1     316      5 838983392  314               Stargate (1994)
## 6 6      1     329      5 838983392  326 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

```
write.csv(edx, file = "edx.csv")
```

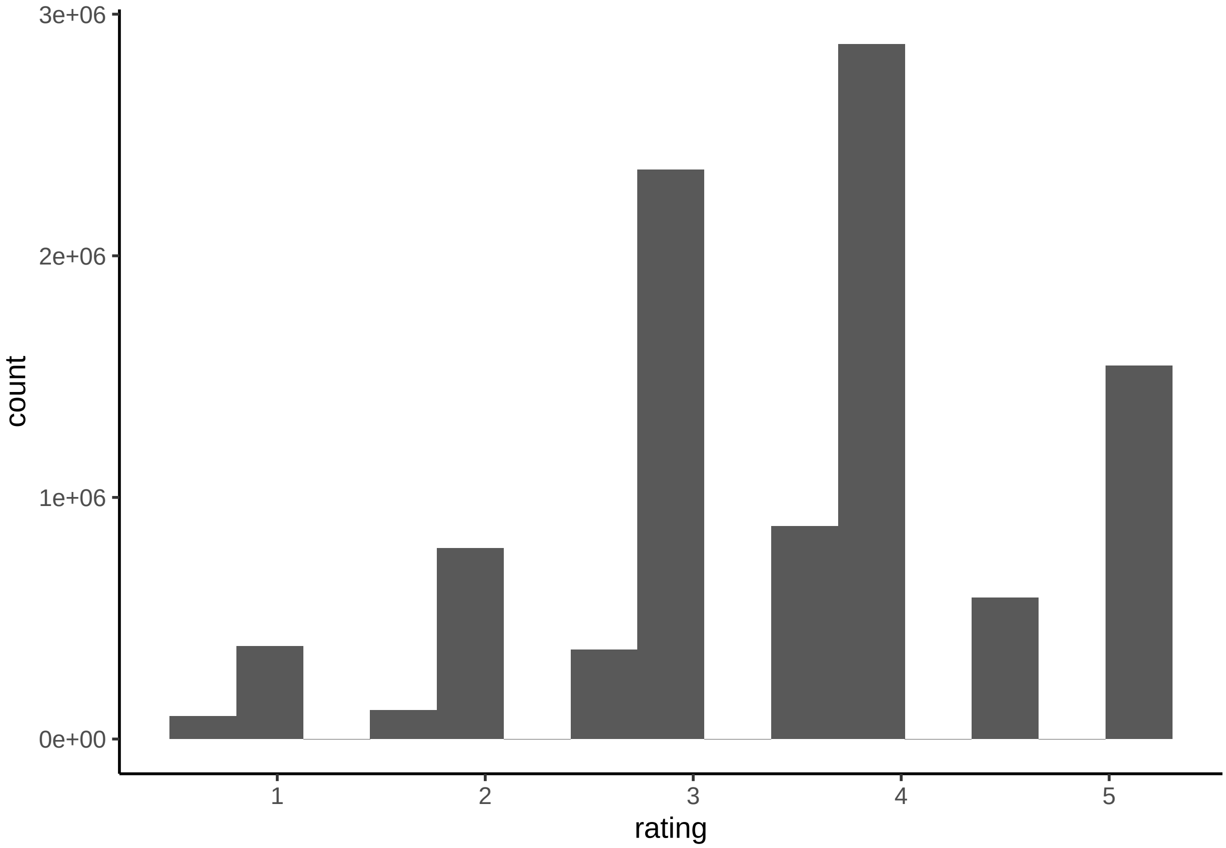
Load the file named edx.csv, give instructions to omit Na's, and read the first lines.

```
edx <- read.csv("edx.csv")
edx <- subset(edx, select = -c(X.1, X,...1 ))
edx <- na.omit(edx)
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

The scope of this project is to predict rating, thus I immediately plot rating to see how it looks like. There are more high ratings than low ones, and "half ratings" are used less.

```
edx %>%
  ggplot(aes(rating))+
  geom_histogram(bins = 15)
```



The following code comes from the capstone exercise and tells how many unique users the dataset contains (69878); this is relevant to understand if some users have voted many movies. In this case, will this aspect be relevant for the model?

```
#unique users
length(edx$userId)
```

```
## [1] 10000054
```

```
n_distinct(edx$userId)
```

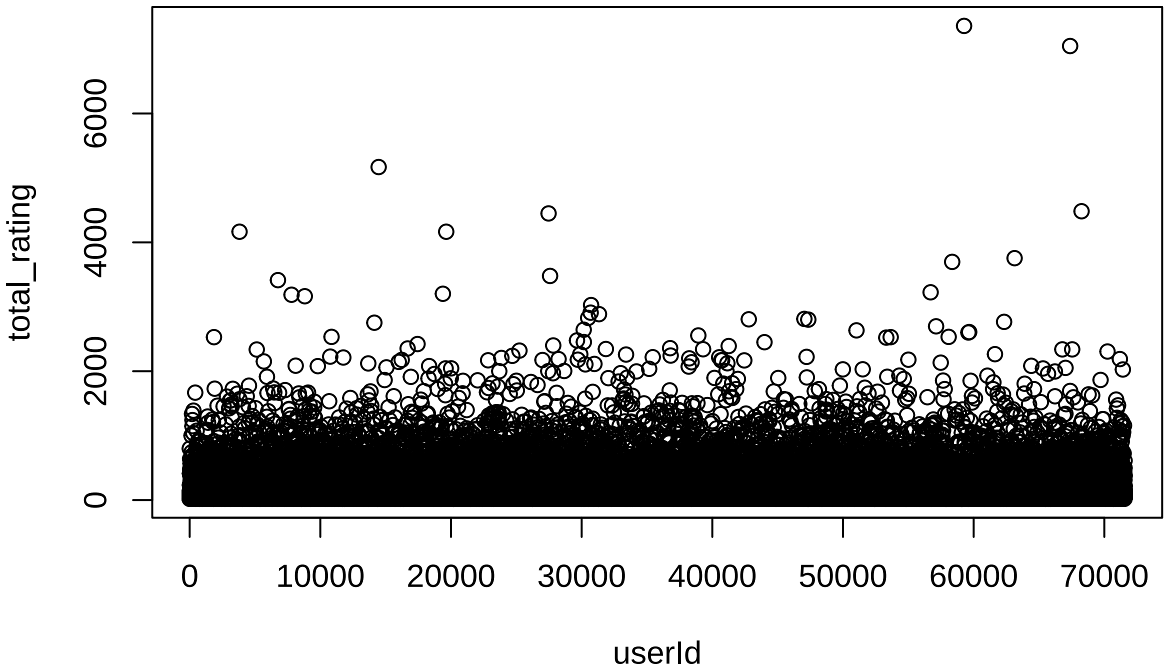
```
## [1] 69878
```

```
#some users made a lot of reviews, the "reviewer_weigth" variable illustrates them
reviewer_weigth <- edx %>%
  select(userId, rating, title) %>%
  group_by(userId) %>%
  summarise(total_rating=n())%>%
  arrange(desc(total_rating))
head(reviewer_weigth)
```

```
## # A tibble: 6 x 2
##   userId total_rating
##    <int>        <int>
## 1  59269         7359
```

```
## 2  67385          7047
## 3  14463          5169
## 4  68259          4483
## 5  27468          4449
## 6   3817          4165
```

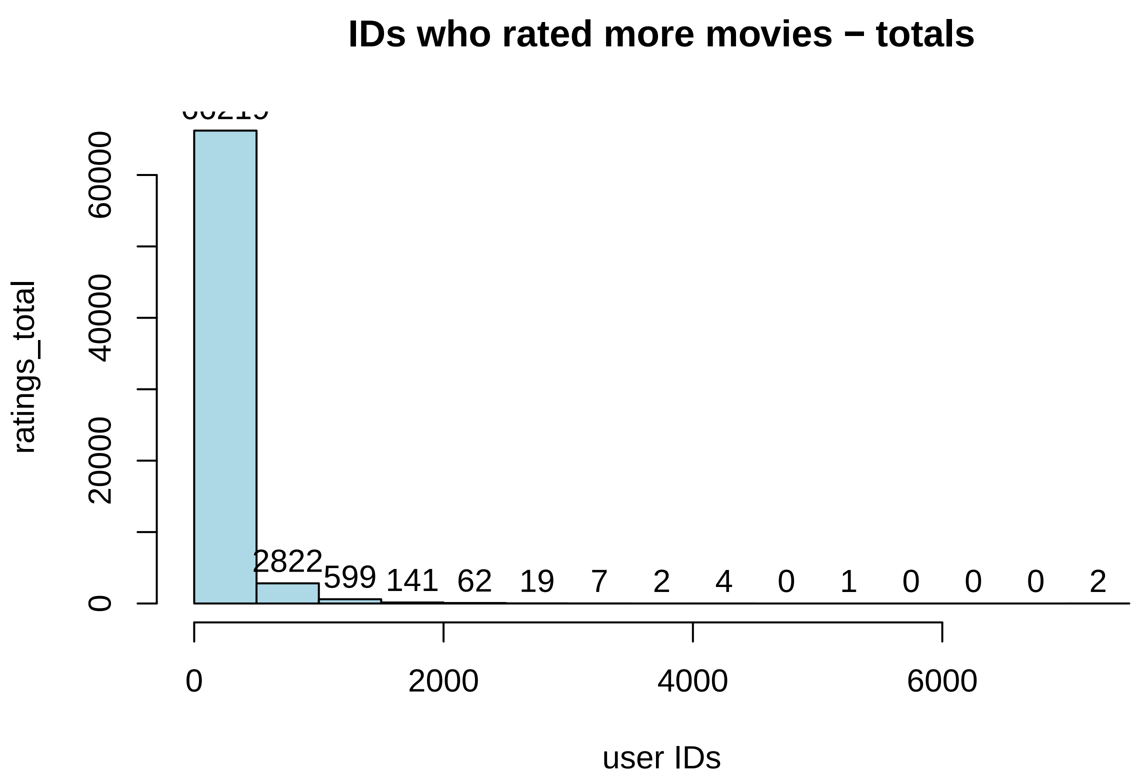
```
plot(reviewer_weigth)
```



These boxplot and histogram represent the users who rated more movies.These results show that just a few users rated a really high number of movies (such as those who rated 6616 movies), and most users rated less than 250 movies.

```
#Boxplotof total ratings for each userID
boxplot(reviewer_weigth$total_rating,
        col="red",
        main ="Boxplot of IDs who rated more movies - totals ")
```


```
#Histogram
hist(reviewer_weigth$total_rating,
     col= "lightblue",
     main= "IDs who rated more movies - totals",
     xlab= "user IDs",
     ylab= "ratings_total",
     labels = TRUE)
```



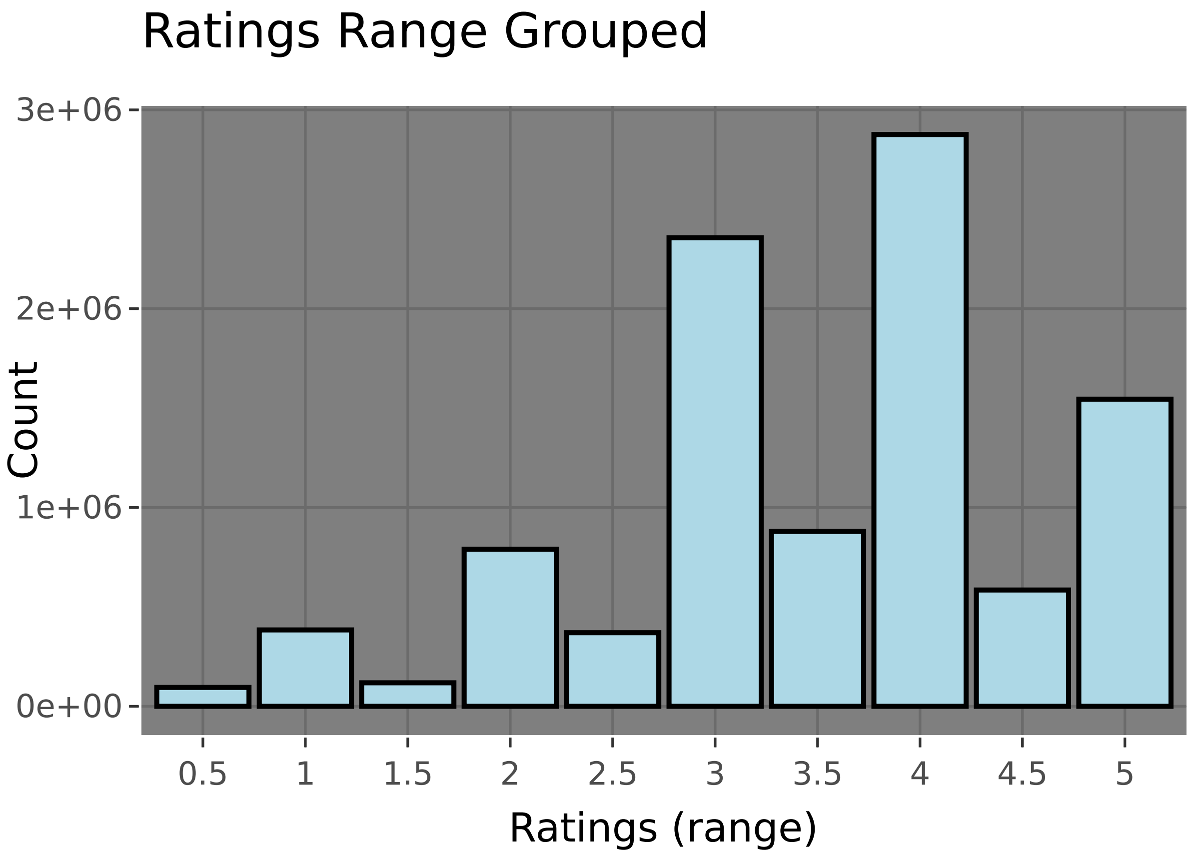
After exploring users, I check ratings, and the top three most used ratings were 4 stars (2588430), then 3 stars (2121240), 5 stars (1390114).

```
edx_totratings <- edx %>% group_by(rating) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
head(edx_totratings)
```

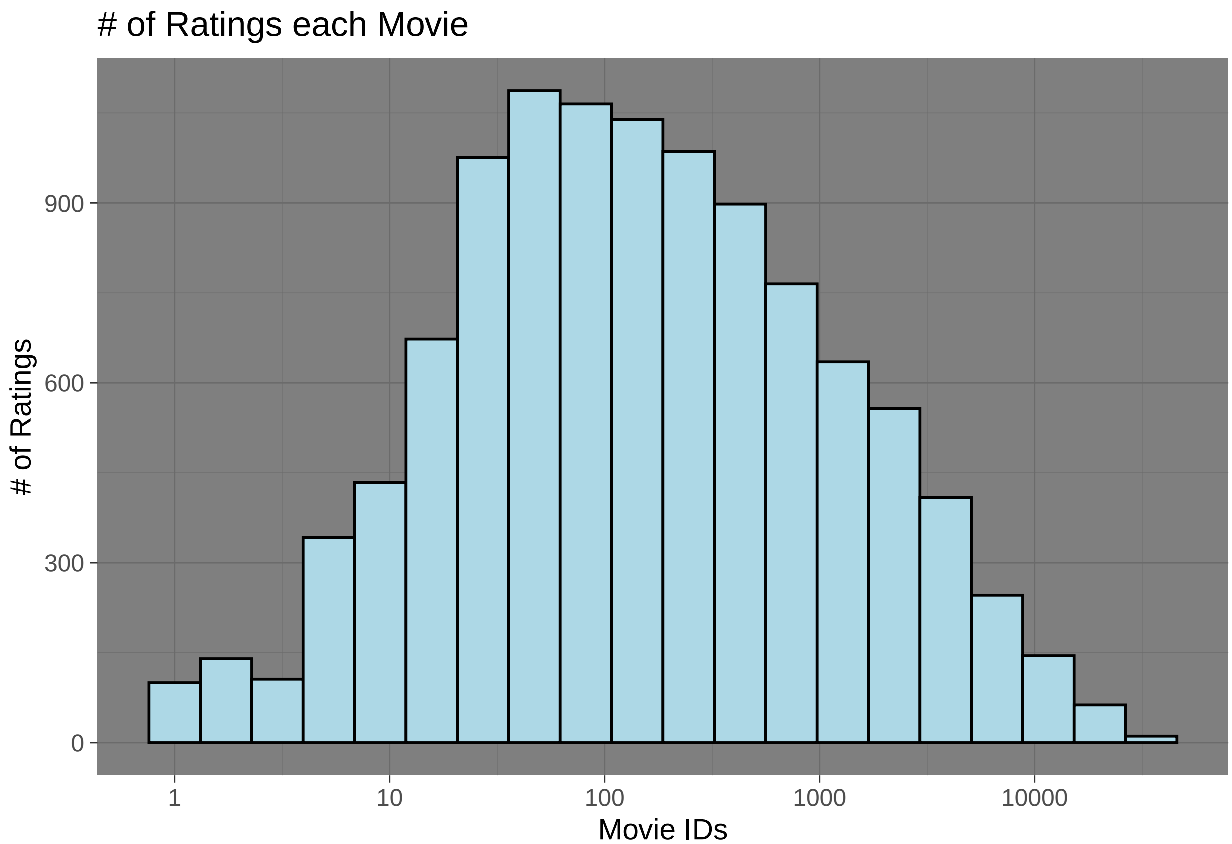
```
## # A tibble: 6 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2875850
## 2    3   2356676
## 3    5   1544812
## 4    3.5  879764
## 5    2    790306
## 6    4.5  585022
```

The bar chart represents in details such results. My takeaway is that some users were benevolent so I wonder if these ratings were given to the same movies, or maybe the same genres? I create an histogram representing movies that received more rating assessments, and I identify the titles of such movies.

```
gedxratings <- edx_totratings %>% mutate(rating = factor(rating)) %>%
  ggplot(aes(rating, count)) +
  geom_col(fill = "lightblue", color = "black") +
  theme_dark() +
  labs(x = "Ratings (range)", y = "Count",
       title = "Ratings Range Grouped",
       caption = "Figure 1 - edx dataset ratings")
ggplotly(gedxratings)
```




```
movie_ratings <- edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=20, color = "black", fill = "lightblue") +
  scale_x_log10() +
  ggtitle("# of Ratings each Movie") +
  xlab("Movie IDs") +
  ylab("# of Ratings")+
  theme_dark()
movie_ratings
```




```
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  top_n(10) %>%
  arrange(desc(count))
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
## Selecting by count
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        count
##      <int> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          34864
##  2     356 Forrest Gump (1994)                                          34457
##  3     593 Silence of the Lambs, The (1991)                             33668
##  4     480 Jurassic Park (1993)                                         32631
##  5     318 Shawshank Redemption, The (1994)                             31126
##  6     110 Braveheart (1995)                                            29154
##  7     457 Fugitive, The (1993)                                         28951
##  8     589 Terminator 2: Judgment Day (1991)                            28948
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 28566
## 10     150 Apollo 13 (1995)                                             27035
## # ... with 10,667 more rows
```

Some movies received many more ratings than others, now I investigate genres. Drama (3910127) and comedy (3540930) are the most rated, but we need to further clean the genres variable since some of them are grouped.

```
genres = c("Drama", "Comedy", "Thriller", "Romance")
sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
})
```

```
##    Drama   Comedy Thriller  Romance
##  4344198  3934068  2584435  1901883
```

Since the year released is attached to the title variable, I extract the year and I check if the most rated movies were released in some specific years, or maybe some genres were most popular in certain years.The new dataset edx2 contains the new variable named "year_released".

```
edx2 <- edx %>% mutate(year_released = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"), reg
head(edx2)
```

```
##   userId movieId rating timestamp                   title
## 1      1     122      5 838985046               Boomerang
## 2      1     185      5 838983525                Net, The
## 3      1     231      5 838983392           Dumb & Dumber
## 4      1     292      5 838983421                Outbreak
## 5      1     316      5 838983392                Stargate
## 6      1     329      5 838983392 Star Trek: Generations
##                          genres year_released
## 1                Comedy|Romance          1992
## 2         Action|Crime|Thriller          1995
## 3                        Comedy          1994
## 4  Action|Drama|Sci-Fi|Thriller          1995
## 5       Action|Adventure|Sci-Fi          1994
## 6 Action|Adventure|Drama|Sci-Fi          1994
```

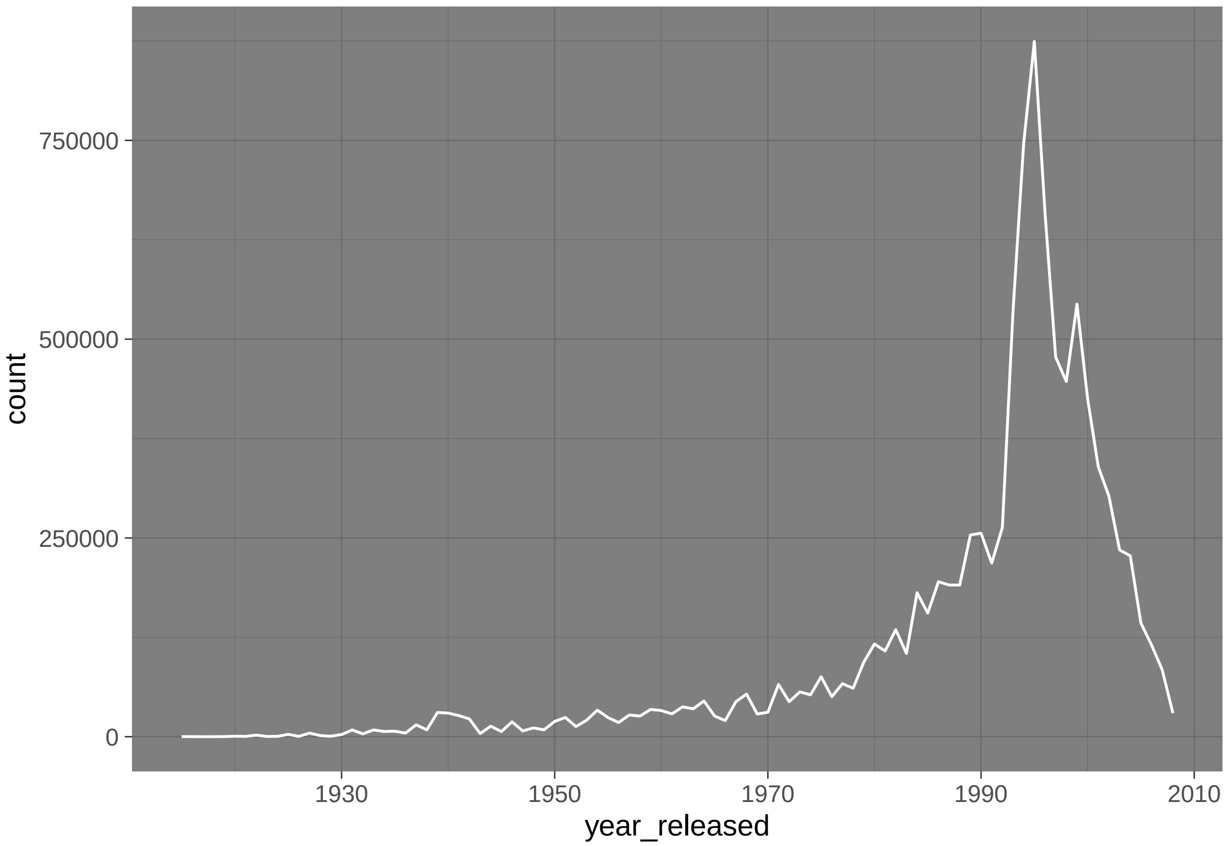
After extracting the year released, I split the number of movies for each year. Looks like most movies have been released during the 90s, so I plot this information to illustrate yearly movie trends.The graphic confirms there was a peak of movies released around the 90s. This is an element I will consider when approaching the choice of the models I will use.

```
movies_per_year <- edx2 %>%
  select(movieId, year_released) %>% # I need movieId and year_released variables
  group_by(year_released) %>% # group_by to collect them by year
  summarise(count = n())  %>% # summarise/count to sum movies per year
  arrange(desc(count))# to see them in order from top released year
movies_per_year
```

```
## # A tibble: 94 x 2
##    year_released  count
##            <dbl>  <int>
##  1          1995 874436
##  2          1994 746042
##  3          1996 659425
##  4          1999 543990
```

```
## 5         1993 534899
## 6         1997 477463
## 7         1998 446739
## 8         2000 425218
## 9         2001 339508
## 10        2002 302452
## # ... with 84 more rows
```

```
ggmovies_per_year <- movies_per_year %>%
  ggplot(aes(x = year_released, y = count)) +
  geom_line(color="white")+
  theme_dark()
ggmovies_per_year
```



Here I understand that dates are important but I must correct some of them.

```
edx2 <- mutate(edx2, year_rated = year(as_datetime(timestamp)))

release <- stringi::stri_extract(edx$title, regex = "(\\d{4})", comments = TRUE) %>% as.numeric()
edx_age <- edx2 %>% mutate(release_date = year_released) %>% select(-timestamp) #change the name of the
head(edx_age)
```

```
##   userId movieId rating             title                genres
## 1      1     122      5         Boomerang       Comedy|Romance
## 2      1     185      5          Net, The Action|Crime|Thriller
## 3      1     231      5      Dumb & Dumber               Comedy
```

```
## 4      1     292      5               Outbreak   Action|Drama|Sci-Fi|Thriller
## 5      1     316      5               Stargate        Action|Adventure|Sci-Fi
## 6      1     329      5 Star Trek: Generations  Action|Adventure|Drama|Sci-Fi
##   year_released year_rated release_date
## 1          1992       1996         1992
## 2          1995       1996         1995
## 3          1994       1996         1994
## 4          1995       1996         1995
## 5          1994       1996         1994
## 6          1994       1996         1994
```

```
edx_age %>%
  filter(release_date < 1900) %>% #filter release dates
  group_by(movieId, title, release_date) %>% #group the variables
  summarize(n = n())
```

```
## `summarise()` has grouped output by 'movieId', 'title'. You can override using
## the `.groups` argument.

## # A tibble: 0 x 4
## # Groups:   movieId, title [0]
## # ... with 4 variables: movieId <int>, title <chr>, release_date <dbl>, n <int>
```

```
edx_age[edx_age$movieId == "4311", "release_date"] <- 1998 #remove wrong dates after 2000
edx_age[edx_age$movieId == "5472", "release_date"] <- 1972
edx_age[edx_age$movieId == "6290", "release_date"] <- 2003
edx_age[edx_age$movieId == "6645", "release_date"] <- 1971
edx_age[edx_age$movieId == "8198", "release_date"] <- 1960
edx_age[edx_age$movieId == "8905", "release_date"] <- 1992
edx_age[edx_age$movieId == "53953", "release_date"] <- 2007

# fix out of range dates
edx_age %>% filter(release_date > 2020) %>% group_by(movieId, title, release_date) %>% summarize(n = n(
```

```
## `summarise()` has grouped output by 'movieId', 'title'. You can override using
## the `.groups` argument.

## # A tibble: 0 x 4
## # Groups:   movieId, title [0]
## # ... with 4 variables: movieId <int>, title <chr>, release_date <dbl>, n <int>
```

```
edx_age[edx_age$movieId == "27266", "release_date"] <- 2004 #remove remaining wrong dates
edx_age[edx_age$movieId == "671", "release_date"] <- 1996
edx_age[edx_age$movieId == "2308", "release_date"] <- 1973
edx_age[edx_age$movieId == "4159", "release_date"] <- 2001
edx_age[edx_age$movieId == "5310", "release_date"] <- 1985
edx_age[edx_age$movieId == "8864", "release_date"] <- 2004
edx_age[edx_age$movieId == "1422", "release_date"] <- 1997


edx_age <- edx_age %>%
  mutate(age_movie = 2022 - release_date, rating_age = year_rated - release_date) #update new age
```

Now that release dates are clean, I wonder: "do movies with the highest number/count of rating have higher ratings? I check the previous table I have created with the top 10 movies and I use movieId to calculate their average ratings.I get 4.154789 for"Pulp Fiction" and 4.012822 for "Forrest Gump" . T

```
pulp_fiction <- edx2 %>%
  select (movieId, rating, title, genres) %>%
  filter (movieId == 296) %>% #here I filer Pulp Fiction
  summarise(avg = mean(rating)) %>%
  arrange(avg)
pulp_fiction
```

```
##        avg
## 1 4.157426
```

```
forrest_gump <- edx2 %>%
  select (movieId, rating, title, genres) %>%
  filter (movieId == 356) %>% #here I filer Forrest Gump
  summarise(avg = mean(rating)) %>%
  arrange(avg)
forrest_gump
```

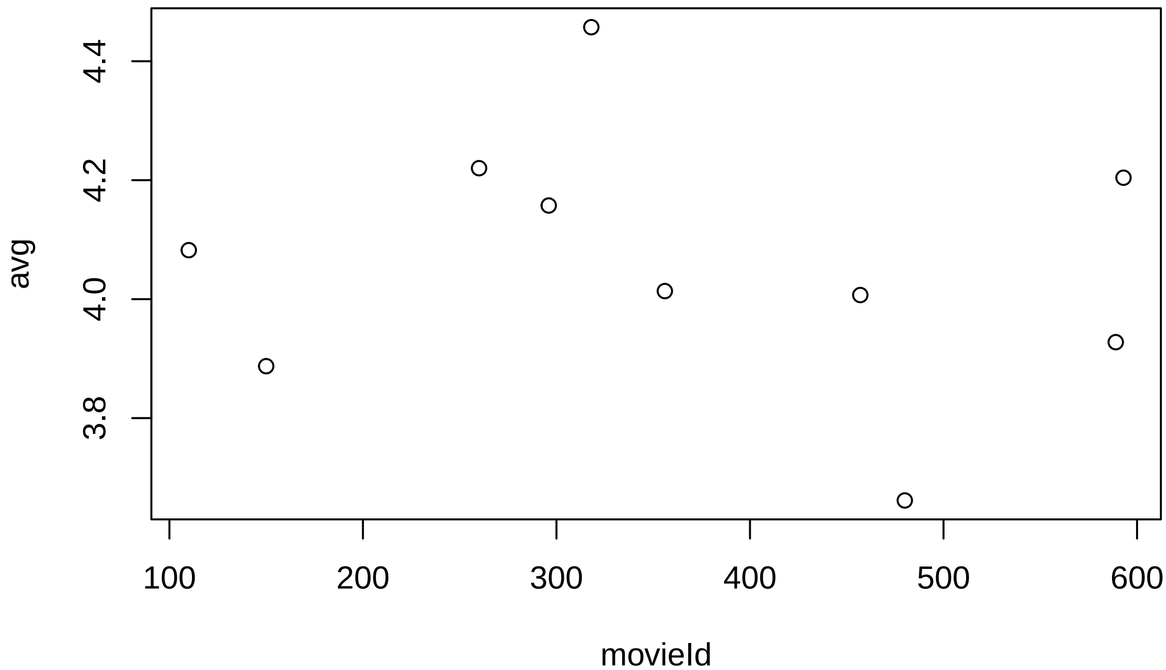
```
##        avg
## 1 4.013582
```

Thus, I use movieId to pull out average ratings of top 10 movies.

```
eamean_movie <- edx2 %>%
  select (movieId, rating, title, genres) %>%
  filter (movieId %in% c(296, 356, 593, 480, 318, 110, 457, 589, 260, 150)) %>% #these are the IDs of t
  group_by(movieId) %>%
  summarise(avg = mean(rating)) %>%
  arrange(avg)
eamean_movie
```

```
## # A tibble: 10 x 2
##    movieId   avg
##      <int> <dbl>
##  1     480  3.66
##  2     150  3.89
##  3     589  3.93
##  4     457  4.01
##  5     356  4.01
##  6     110  4.08
##  7     296  4.16
##  8     593  4.20
##  9     260  4.22
## 10     318  4.46
```

```
plot(eamean_movie)
```



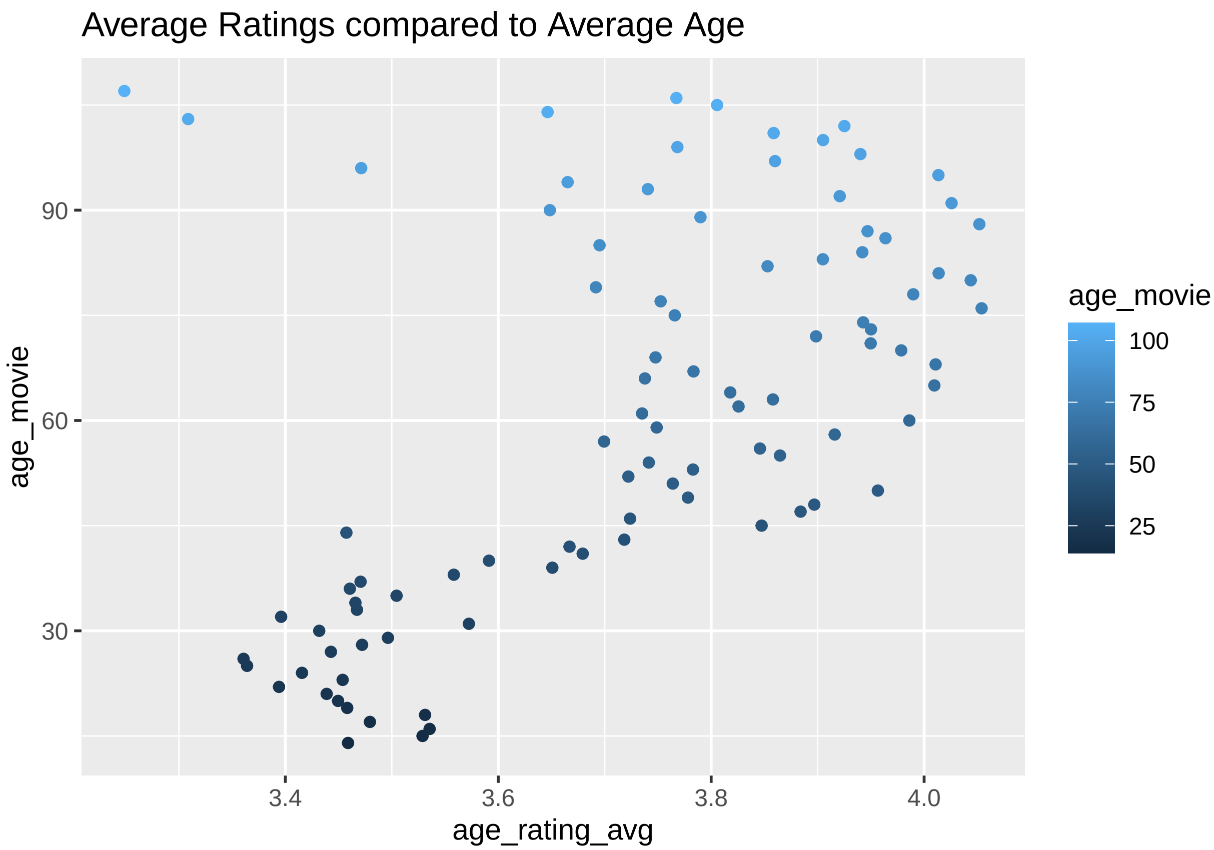
I want to compare the mean of all the movies in the dataset. Since the top 10 movies together have an average rating of 4.063742; the average ratings of all the movies in the dataset is 3.512465. This is the most interesting aspect to me, and I will use these results to choose my algorithm model.

```
mean(edx$rating) #calculate the mean of all the movies in the dataset
```

```
## [1] 3.512422
```

In the next chart I use the mean rating of all movies and the average age. This way I understand whether the movie's age increases or decreases ratings.The scatterplot shows that the oldest movies have higher ratings compared to most recent ones.

```
edx_avg_age <- edx_age %>% group_by(age_movie) %>% summarize(age_rating_avg = mean(rating)) #create ave

edx_avg_age %>%
  ggplot(aes(age_rating_avg, age_movie)) +
  geom_point(aes(color=age_movie)) +
  ggtitle("Average Ratings compared to Average Age")+
  theme_gray()
```



At this point I recall that a lot of movies have been released in the 90's, and for the purpose of this project, I want to check if some genres have been released in particular years, for example whether customers' preferences had seasonal trends. So, first I clean the genres variable since some of these are grouped together, then I pull out the correct release year.

```
genres_df <- edx_age  %>%
  separate_rows(genres, sep = "\\|") %>% #tell R to use sep to separate genres
  group_by(genres) %>% #group the genres I have split
  summarise(number = n()) %>% #sum the number of genres
  arrange(desc(number))
head(genres_df, 10)
```

```
## # A tibble: 10 x 2
##    genres     number
##    <chr>       <int>
##  1 Drama     4344198
##  2 Comedy    3934068
##  3 Action    2845349
##  4 Thriller  2584435
##  5 Adventure 2121074
##  6 Romance   1901883
##  7 Sci-Fi    1490489
##  8 Crime     1474957
##  9 Fantasy   1028482
## 10 Children   820149
```

```r
genres_per_year <- edx_age %>%
  select(genres, year_released) %>% # genres and year_released are selected from edx_age dataset
  group_by(genres, year_released) %>% # group by year released
  summarise(count = n())  %>% # count how many movies were released every year
  arrange(desc(count))
```

```
## `summarise()` has grouped output by 'genres'. You can override using the
## `.groups` argument.
```

```r
head(genres_per_year, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   genres [4]
##    genres                 year_released count
##    <chr>                          <dbl> <int>
##  1 Comedy                          1994 88206
##  2 Drama                           1994 87118
##  3 Drama                           1999 78834
##  4 Comedy                          1996 77866
##  5 Drama                           1993 71849
##  6 Comedy|Romance                  1995 64508
##  7 Comedy                          1995 62023
##  8 Comedy                          1999 56148
##  9 Drama                           2000 53182
## 10 Action|Crime|Thriller           1995 50811
```

Modeling approach 1 (we must provide at least 2 models).

According to Theobald, choosing the most relevant variables to use for a model is fundamental for obtaining the best results; following the same logic, wrong variables can decrease the model's accuracy (2017, p. 36). According to (Serrano, 2021, p. 2) discovering patterns and correlations is the recommended approach for machine learning predictions. To choose where to start from, Theobald suggests to begin with "simple supervised algorithms such as linear regression, logistic regression, decision trees, or K-means clustering" (2017, p. 52). Since the goal of this project is to predict an unknown variable (future ratings) I start with regression analysis using Caret package. Furthermore, the EDA has disclosed the possibility that the some variables might have similarities.For example, the chart illustrating the high number of positive ratings, average ratings among top movies that received more ratings, the average of the rest of the dataset, the concentration of movies during some years, and genres. After performing my regression analysis using Caret and after the good RMSE result I attempt a cluster analysis. For the second model I use the tidymodels package to identify those variables that likely have elements in common.

RMSE The goal of this project is to asses our model using RMSE, and the result must be lower than 0.86490. RMSE is widely used in regression analysis statistics to measure the relationship between predictor and response variables; it tells how good our model is (Bobbit, 2020).

The formula is:

$$RMSE = \sqrt{\left[\Sigma (P_i - O_i)^2 / n \right]}$$

where:

- Σ is a fancy symbol that means "sum"
- P_i is the predicted value for the i^{th} observation in the dataset
- O_i is the observed value for the i^{th} observation in the dataset
- n is the sample size

To prepare data for my model I use the whole dataset named edx and I select some numeric variables. Since I was impressed by the number of ratings those first 10 movies had, I decide to start with them and I select three variables: movieId, rating, and userId; I assign them the name "edx_reduced".

```
edx_reduced <- edx %>% select(movieId, rating, userId) %>% #select the numeric variables I want to inve
  group_by(movieId, userId) %>%
  summarise(rating = mean(rating)) %>% #assign rating variable to the mean rating of the movies selecte
  top_n(10) #pick the top 10 movies
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
## Selecting by rating
```

```
head(edx_reduced)
```

```
## # A tibble: 6 x 3
## # Groups:   movieId [1]
##   movieId userId rating
##     <int>  <int>  <dbl>
## 1       1     23      5
## 2       1     24      5
## 3       1     30      5
## 4       1     90      5
## 5       1    101      5
## 6       1    115      5
```

```
print(edx_reduced, n=10)
```

```
## # A tibble: 1,615,844 x 3
## # Groups:   movieId [10,677]
##    movieId userId rating
##      <int>  <int>  <dbl>
##  1       1     23      5
```

```
## 2      1     24     5
## 3      1     30     5
## 4      1     90     5
## 5      1    101     5
## 6      1    115     5
## 7      1    125     5
## 8      1    126     5
## 9      1    134     5
## 10     1    137     5
## # ... with 1,615,834 more rows
```

The formula for linear regression is mod <- lm(y ~ x, my_data)

The formula to make predictions is pred <- predict(mod, my_data)

The formula to calculate RMSE is sqrt(mean(error ˆ 2))

I will fit my linear model, make my prediction, and calculate errors using the formula errors = predicted - actual(Mayer & Kuhn, n.d.)

```
model <- lm(rating ~ ., edx_reduced) #this is the formula
model
```

```
##
## Call:
## lm(formula = rating ~ ., data = edx_reduced)
##
## Coefficients:
## (Intercept)      movieId       userId
##   4.982e+00   -9.605e-06   -2.520e-08
```

Now I use the same dataset to compute "Out-of-sample" RMSE for linear regression. This is important because it tells me how my model performs on new data. I randomly order my data and then split the dataset using train/test functions; this process is often compared to "shuffling decks of playing cards" before playing. The train and split functions are also very important to avoid over optimistic predictions (overfitting) after using the same dataset (Mayer & Kuhn, n.d.). "The model is accurate when the error rate for the training and test dataset is low" (Theobald, 2017 p. 48).

```
set.seed(42) #set a random seed

rows <- sample(nrow(edx_reduced)) #the sample function shuffles row indices in the edx_reduced dataset


edx_age_reduced <- edx_reduced[rows, ] # randomly reorder data
```

There are many ways to train/test and split data (also called "split validation"). Theobald (2017, p. 46) recommends 70/30 or 80/2, but we should also consider the size of the dataset, so there is not a fixed rule. The instructions of the Capstone exercise required that "Validation set to be 10% of the whole data", so I will use 90/10.

```
split <- round(nrow(edx_reduced) * 0.90) # use the split function to tell the percentage to split

train <- edx_reduced[1:split, ] # Create train

test <- edx_reduced[(split + 1):nrow(edx_reduced), ] # Create test
```

To predict on test set I have split edx_reduced using the split function to train and test, then I use the lm() function for model fitting only on the training dataset (instead of the whole dataset).

In R, the predict() function predicts the model on new data - the test dataset - because this has not been used for training the model. This way I obtain the error for the out-of-sample model; then, I use the error for RMSE's formula (sqrt(mean(error^2))).

```
model <- lm(rating ~ ., train) # regression formula to train model

p <- predict(model, test) # assign prediction to "p" and predict using test

error <- p - test[["rating"]] #apply formula errors = predicted - actual, thus errors between the predi


sqrt(mean(error^2)) #RMSE formula, the next result is our RMSE
```

```
## [1] 0.7256236
```

According to this RMSE the model is accurate.

Model Number 2 Kmeans k-Means is an unsupervised clustering model that groups similar data points. The method splits data in k groups and it is helpful to discover new patterns or similarities, or disclose information about the number of clusters identified (Theobald, 2017, p. 72). I am using it following the lofica of the previous analysis related to the top 10 movies, those receiving most ratings. The next model runs using the tidymodels package (Silge & Kuhn, 2022). This code takes some time to run; in order to see how it works I recommend running just a part of the dataset.

```
#Do not run if you have the other dataset loaded from EDA, but if you want to run this faster, then sel
edx <- read.csv("edx.csv", nrows = 10000)


#reload edx_reduced with fewer observations
edx_reduced <- edx %>% select(movieId, rating, userId) %>% #select the variables we have previously ide
  group_by(movieId, userId) %>%
  summarise(rating = mean(rating)) %>%
  top_n(10)
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
## Selecting by rating
```

In Tidymodels, recipes are used to prepare data we will use (feature engineering). The extraction method named Principal Component Analysis (PCA) is an unsupervised method and it combines new features with the predictors we originally used. PCA's new features are not correlated each other and they berform better when variables are normalized. The next code implies that we have already performed EDA as we did at the beginning of this analysis (Silge & Kuhn, 2022, chapter 8).

```
edx_rec <- recipe(~ ., data = edx_reduced) %>%
  step_normalize(all_predictors()) %>% #normalize variables
  step_pca(all_predictors(), num_comp = 2, id = "pca")

# Print out recipe
edx_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##  predictor          3
##
## Operations:
##
## Centering and scaling for all_predictors()
## PCA extraction with all_predictors()
```

I call prep() to estimate the statistics required by PCA and I apply them to a new variable named "features_2d_edx" I call bake(new_data = NULL) to get fitted PC transformation of "features_2d_edx"

```
edx_estimates <- prep(edx_rec) #the function prep() estimates the necessary statistics and applies them


features_2d_edx <- edx_estimates %>% # the function bake(new_data = NULL) returns preprocessed data
  bake(new_data = NULL)

features_2d_edx %>% # Print baked data
  slice_head(n = 5)
```

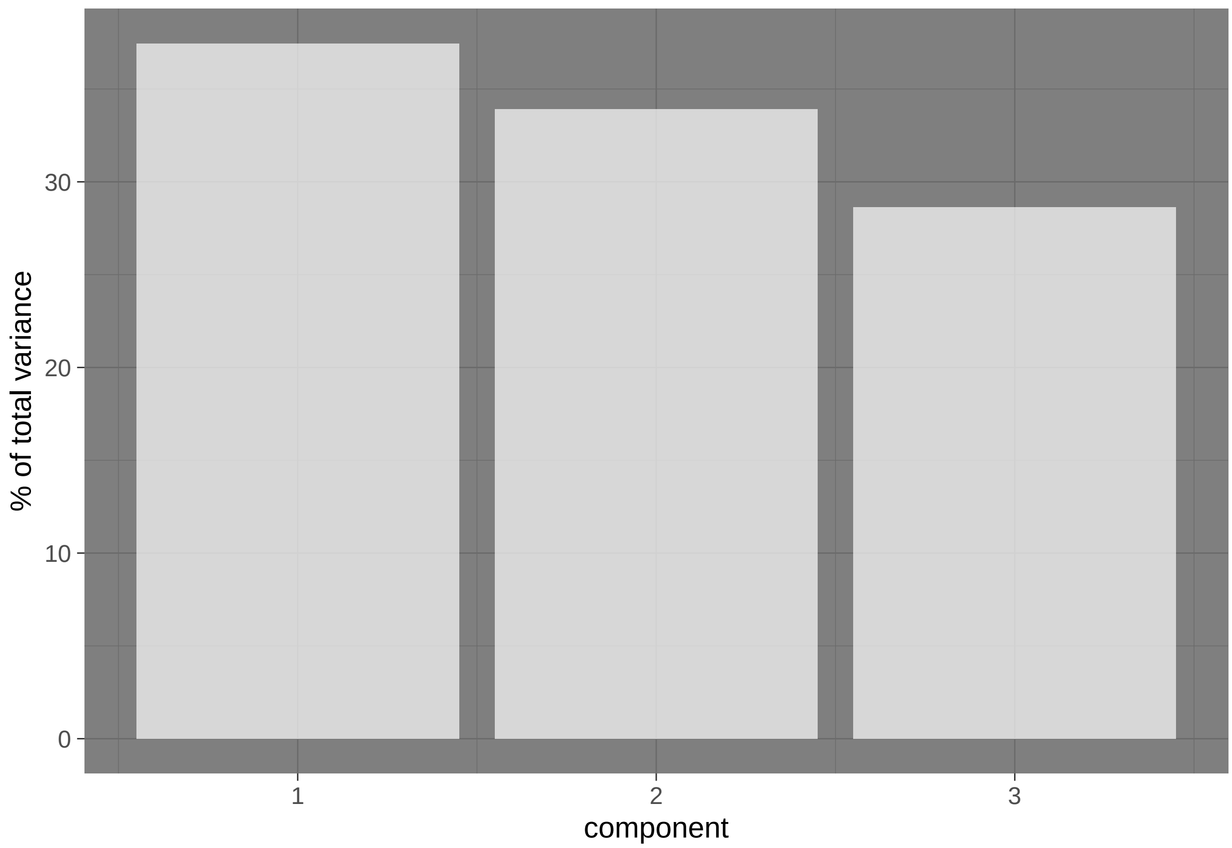
```
## # A tibble: 5 x 2
##       PC1    PC2
##     <dbl>  <dbl>
## 1 -0.262  -0.966
## 2 -0.289  -0.956
## 3 -0.450  -0.899
## 4  0.172  -0.653
## 5  0.0914 -0.625
```

Components containing more information (i.e. variance); "pca_estimates" returns each component's variance (RPubs, 2021).

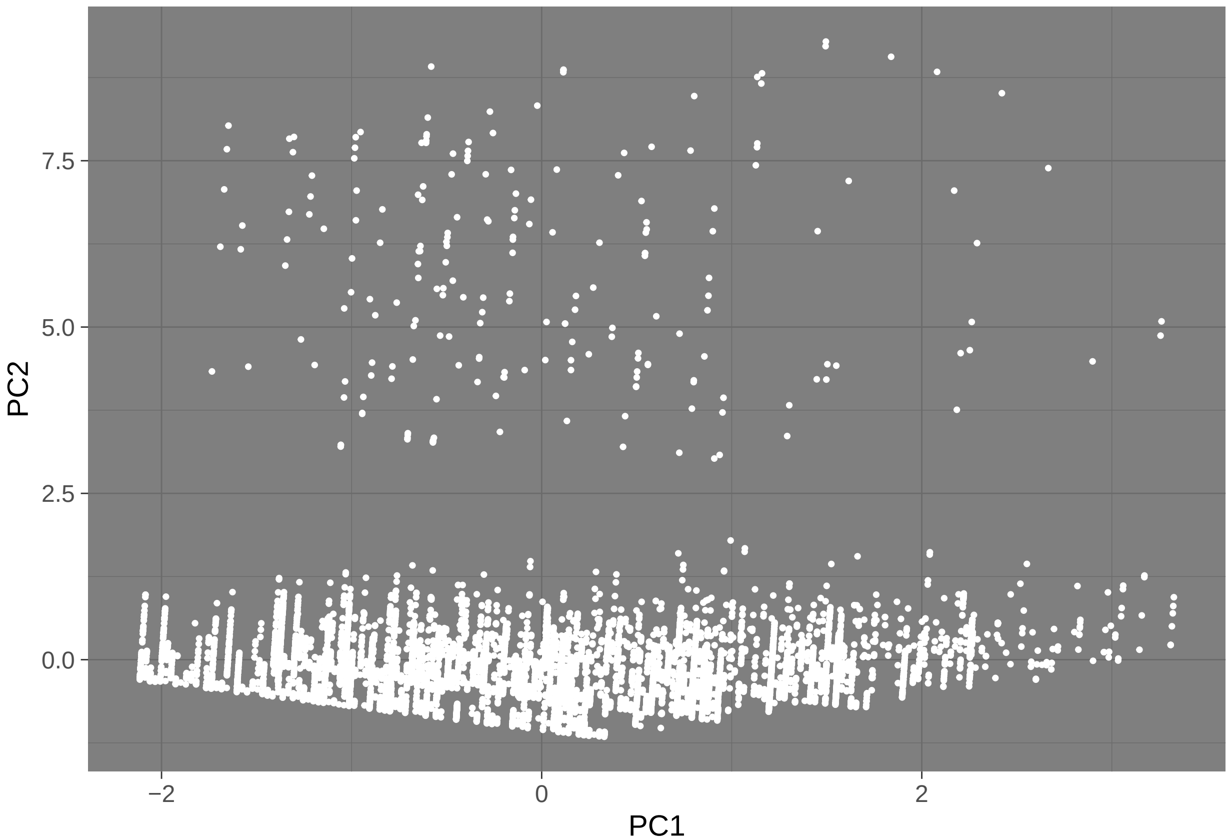
```
edx_estimates %>%
  tidy(id = "pca", type = "variance") %>% #variance for each component in original variables
  filter(str_detect(terms, "percent"))
```

```
## # A tibble: 6 x 4
##   terms                       value component id
##   <chr>                       <dbl>     <int> <chr>
## 1 percent variance             37.5         1 pca
## 2 percent variance             33.9         2 pca
## 3 percent variance             28.6         3 pca
## 4 cumulative percent variance  37.5         1 pca
## 5 cumulative percent variance  71.4         2 pca
## 6 cumulative percent variance 100           3 pca
```

```
theme_set(theme_dark())
# Plot PC variance
edx_estimates %>%
  tidy(id = "pca", type = "variance") %>%
  filter(terms == "percent variance") %>%
  ggplot(mapping = aes(x = component, y = value)) +
  geom_col(fill = "white", alpha = 0.7) +
  ylab("% of total variance")
```




```
#Plot of PC scores
features_2d_edx %>%
  ggplot(mapping = aes(x = PC1, y = PC2)) +
  geom_point(size = 0.5, color = "white")
```



kmeans() built-in function runs after using numeric values having the same scale

```
edx_features<- recipe(~ ., data = edx_reduced) %>%
  step_normalize(all_predictors()) %>% #normalize data
  prep() %>%
  bake(new_data = NULL)

# Print to see data
edx_features %>%
  slice_head(n = 5)
```

```
## # A tibble: 5 x 3
##   movieId userId rating
##     <dbl>  <dbl>  <dbl>
## 1  -0.494 -0.902  1.25
## 2  -0.494 -0.864  1.25
## 3  -0.494 -0.635  1.25
## 4  -0.494 -0.521  0.258
## 5  -0.494 -0.406  0.258
```

Create model, at this point I stil do not know the ideal number of clusters

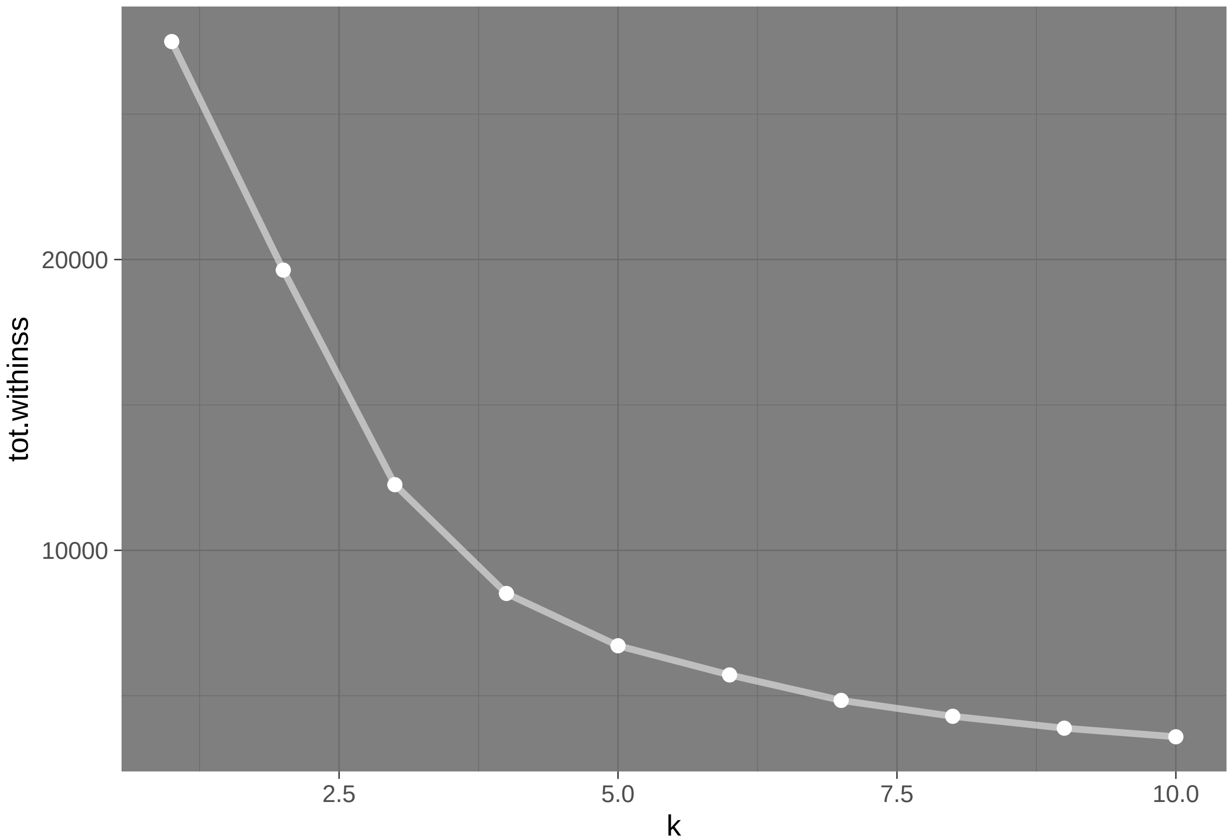
```
#set.seed(2056)
# Create 10 models with 1 to 10 clusters
kclusts <- tibble(k = 1:10) %>%
  mutate(
```

```r
    model = map(k, ~ kmeans(x = edx_features, centers = .x, nstart = 20)), #use map to replace for loop
    glanced = map(model, glance)) %>%
  unnest(cols = c(glanced))

# See kclusts
kclusts
```

```
## # A tibble: 10 x 6
##        k model    totss tot.withinss betweenss  iter
##    <int> <list>   <dbl>        <dbl>     <dbl> <int>
##  1     1 <kmeans> 27498       27498.  7.20e-10     1
##  2     2 <kmeans> 27498       19637.  7.86e+ 3     1
##  3     3 <kmeans> 27498       12258.  1.52e+ 4     2
##  4     4 <kmeans> 27498        8517.  1.90e+ 4     4
##  5     5 <kmeans> 27498        6719.  2.08e+ 4     3
##  6     6 <kmeans> 27498        5714.  2.18e+ 4     3
##  7     7 <kmeans> 27498        4839.  2.27e+ 4     3
##  8     8 <kmeans> 27498        4292.  2.32e+ 4     4
##  9     9 <kmeans> 27498        3883.  2.36e+ 4     4
## 10    10 <kmeans> 27498        3587.  2.39e+ 4     4
```

Plot to obtain an elbow curve showing the ideal number of clusters using the Total within-cluster sum of squares (WCSS) method (tot.withinss)'. The result shows a change at the 4th point, meaning that the optimal clusters are 4 (I tried and this result is similar if you run the whole edx_reduced dataset or 100000 observations).

```r
kclusts %>%
  ggplot(mapping = aes(x = k, y = tot.withinss)) +
  geom_line(size = 1.2, alpha = 0.5, color = "white") +
  geom_point(size = 2, color = "white")
```

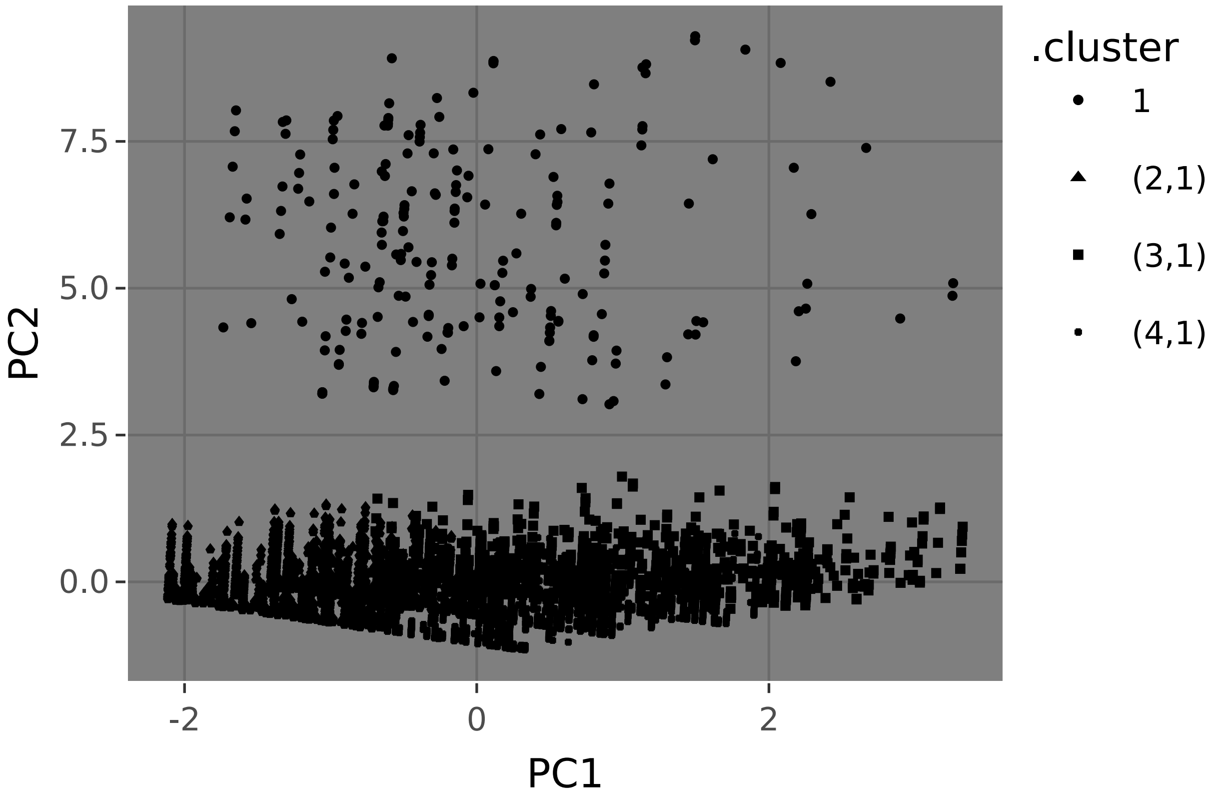


Now I use K-Means with k = 4 clusters as per previous elbow plot

```
set.seed(2056)
final_kmeans <- kmeans(edx_features, centers = 4, nstart = 100, iter.max = 1000)

results <- augment(final_kmeans, edx_features) %>% #prediction is added
  bind_cols(features_2d_edx)  # bind columns pca_data - features_2d_edx

results %>%
  slice_head(n = 5) #see results
```

```
## # A tibble: 5 x 6
##   movieId userId rating .cluster     PC1    PC2
##     <dbl>  <dbl>  <dbl> <fct>      <dbl>  <dbl>
## 1  -0.494 -0.902  1.25  4        -0.262  -0.966
## 2  -0.494 -0.864  1.25  4        -0.289  -0.956
## 3  -0.494 -0.635  1.25  4        -0.450  -0.899
## 4  -0.494 -0.521  0.258 4         0.172  -0.653
## 5  -0.494 -0.406  0.258 4         0.0914 -0.625
```

```
#Visualize clusters using plotly package to so hover and see data together with clusters
cluster_plot <- results %>%
  ggplot(mapping = aes(x = PC1, y = PC2)) +
  geom_point(aes(shape = .cluster), size = 0.5) +
  scale_color_manual(values = c("darkorange","purple","cyan4"))

ggplotly(cluster_plot)
```




```
#If you zoom you can see the 4 distinct clusters represented by the different shapes. #These are the gr
#similar ones at the bottom.
```

Results and comments about the model

The regression analysis performed using the caret package, a selection of the variables movieId, rating, and userId returns an RMSE in line with our objective. This result is also consistent with EDA and shows that some movies received more ratings and higher ratings than others. Just for curiosity, I have performed the same linear regression using tidymodels package and I have obtained similar results. One interesting aspect is that tidymodels has the vip() function, that illustrates which variables are the most important ones and movieId turned out to be the most relevant (the chart is attached in the appendix 2 section). One interesting thing to note is that this exercise required a 90/10 split and results were quite different using other splits (e.g. 80/20). Additionally, after using different seeds and shuffles RMSE's result sensibly changes, but always reaches the goal when I use the top_n(10) movies approach.

K-means is an unsupervised method that can be used to identify clusters, thus used to identify patterns, groups, clusters, and similar characteristics; however, it does not provide an outcome such as a dependent variable as we might expect from other models. In fact, the output of a prediction made using regression analysis is numeric, while classification models return qualitative values that can be both ordered or not (Silge & Kuhn, 2022).

K-means provided interesting information about existing clusters. Another aspect of this approach is that I have used the Tidymodels package that offers an easy way to execute machine learning predictions using workflows. That means, once the model has been created, we can easily switch and try another one without re-writing the code from scratch.

Conclusion, limitations, and future work

This project has confirmed the importance of EDA and data cleaning to understand the way variables work, and the relationships between them. The results obtained using the Movielens datasets could be used to explore further correlations, for example, the most rated movies were released in the 90s, and this aspect can be related to other elements such as social, economic, and technological. For example, what is the psychology behind ratings? And since the most rated movies have higher ratings, do people mostly rate only what they like? And does the order or the time when the first rating is made affect other users' behavior? Another example is that according to YouTube experts, users are "all or nothing", that means they like it or they do not. For this reason the company together with Netflix and other firms have substituted ratings with "thumbs up and thumbs down" (Khanna, 2017).

It would also be interesting to investigate the overall budget invested for movie production and advertising, and how users' interests have been influenced in recent years by social media. The Numbers website (2019) states that movie budgets are not easy to find, but they have published the ranking of the most expensive movie budgets, and these were all released between 2007 and 2019. However, in terms of budget the comparison between recent movies and the ones we used in the dataset should consider other elements, for example technology advancement. For instance, Lewis (1987) explained that movie popularity increased at the end of the 80s thanks to video cassettes. This statement is consistent with the results of our project and the peaks in released movies around the 90s., but technologies changed a lot since then. New services such a Netflix have disrupted the movie landscape and these companies have a lot of data about their customers, so they can create customized proposals for their users after analyzing their behaviors. In future work I expect to find more useful variables, for example the channels were the movies were released (cinema or video streaming).

References

Bobbit, Z. (2020, February 10). How to Calculate Root Mean Square Error (RMSE) in Excel. Statology. https://www.statology.org/root-mean-square-error-excel/Capstone instructions. (n.d.). Data Science: Capstone. EdX. Retrieved September 4, 2022, from https://www.edx.org/course/data-science-capstoneDeane-Mayer, Z., & Kuhn, M. (n.d.). Machine Learning with caret in R. Datacamp.HARPER, F. M., & KONSTAN, J. (2015). The MovieLens Datasets: History and Context. ACM Trans. Interact. Intell. Syst, 20. University of Minnesota. http://files.grouplens.org/papers/harper-tiis2015.pdfKhanna, H. (2017). The Psychology of Rating Systems. Hackernoon.com. https://hackernoon.com/the-psychology-of-rating-systems-3103e26fddd8Lewis, P. H. (1987, February 11). BUSINESS TECHNOLOGY: ADVANCES IN FILM; Low-Budget Movies Get A High Gloss. The New York Times. https://www.nytimes.com/1987/02/11/business/business-technology-advances-in-film-low-budget-movies-get-a-high-gloss.htmlRPubs. (2021). RPubs - Train and Evaluate Clustering Models using Tidymodels and friends. Rpubs.com. https://rpubs.com/eR_ic/clusteringSerrano, L. G. (2021). Grokking machine learning (pp. 1–512). Manning Publications. https://www.manning.com/books/grokking-machine-learning#tocSilge, J., & Kuhn, M. (2022). Tidy Modeling with R. In www.tmwr.org. https://www.tmwr.org/The Numbers. (2019). The Numbers - Movie Budgets. The-Numbers.com. https://www.the-numbers.com/movie/budgets/allTheobald, O. (2017). Machine learning for absolute beginners : a plain English introduction (pp. 1–162). The Author.

Appendix 1

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

Note: this process could take a couple of minutes

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

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

MovieLens 10M dataset:

https://grouplens.org/datasets/movielens/10m/

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

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

movies <- str_split_fixed(readLines(unzip(dl, “ml-10M100K/movies.dat”)), “\::”, 3) colnames(movies) <- c(“movieId”, “title”, “genres”)

if using R 3.6 or earlier:

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId], title = as.character(title), genres = as.character(genres)) # if using R 4.0 or later: movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId), title = as.character(title), genres = as.character(genres))

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

Validation set will be 10% of MovieLens data

set.seed(1, sample.kind=“Rounding”) # if using R 3.5 or earlier, use `set.seed(1)` test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE) edx <- movielens[-test_index,] temp <- movielens[test_index,]

Make sure userId and movieId in validation set are also in edx set

validation <- temp %>% semi_join(edx, by = “movieId”) %>% semi_join(edx, by = “userId”)

Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation) edx <- rbind(edx, removed)

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

Appendix 2 *2* Graphic showing variable importance (obtained after running the same analysis and regression model using Tydimodels)

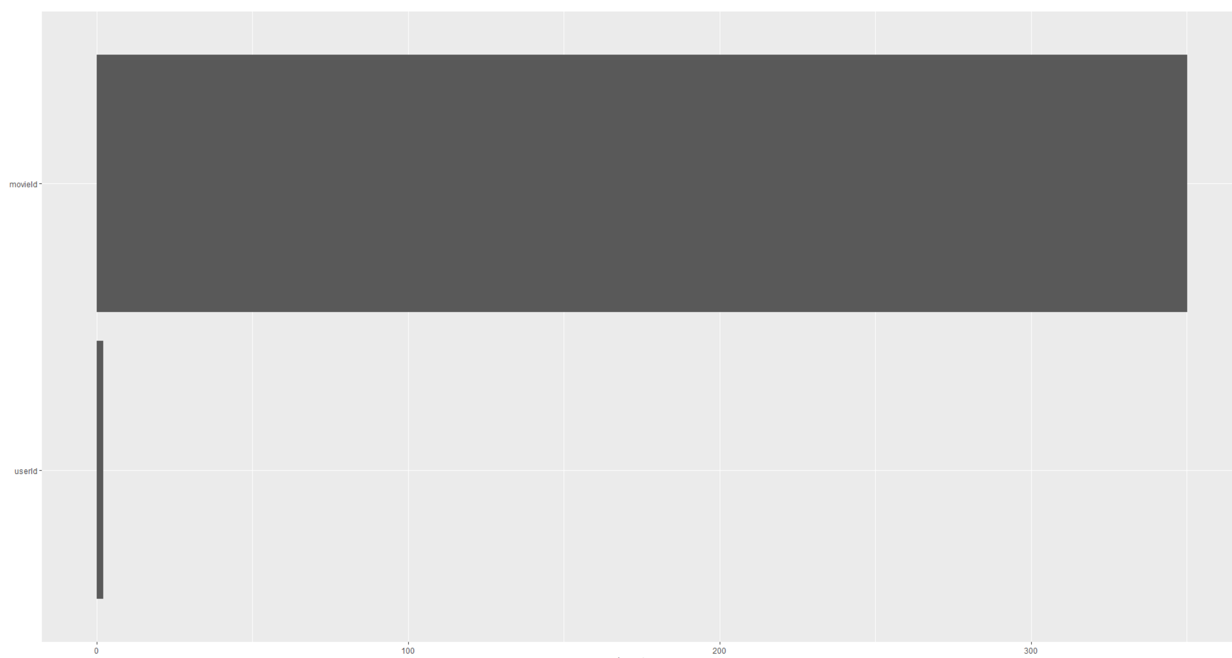


Figure 1: Variable importance