Code

Building a movie recommendation system

22/01/2023

Introduction

In this project, I aim to create a movie recommendation system using the MovieLens dataset, which was instructed on the website of https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+3T2022/block-v1:HarvardX+PH125.9x+3T2022+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+3T2022+type@vertical+block@5dc89f24ec02450b91ac16eac0cec1f6 (https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+3T2022/block-v1:HarvardX+PH125.9x+3T2022+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+3T2022+type@vertical+block@5dc89f24ec02450b91ac16eac0cec1f6). The dataset includes userId, movieId, rating, timestamp, title and genres. This can be then used for modelling and prediction.

Methods

Data

The creation of the dataset was provided here https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+3T2022/block-v1:HarvardX+PH125.9x+3T2022+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+3T2022+type@vertical+block@e9abcdd945b1416098a15fc95807b5db (https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+3T2022/block-v1:HarvardX+PH125.9x+3T2022+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+3T2022+type@vertical+block@e9abcdd945b1416098a15fc95807b5db)

Data loading

Hide

```
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# 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")

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Data saving and reloading

This procedure is for the convenience of use only.

Hide

```
library(data.table)
fwrite(edx, "edx.csv")
```

Data exploration

This step was to understand some more details of the data (e.g.distribution).

Hide

```
#Variables
edx <- fread("edx.csv")
str(edx)
```

```
Classes ‘data.table’ and 'data.frame':  9000055 obs. of  6 variables:
 $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
 $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
 $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 8389
84885 838983707 838984596 ...
 $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)"
...
 $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "A
ction|Adventure|Sci-Fi" ...
 - attr(*, ".internal.selfref")=<externalptr>
```

Hide

```
# Summary
summary(edx)
```

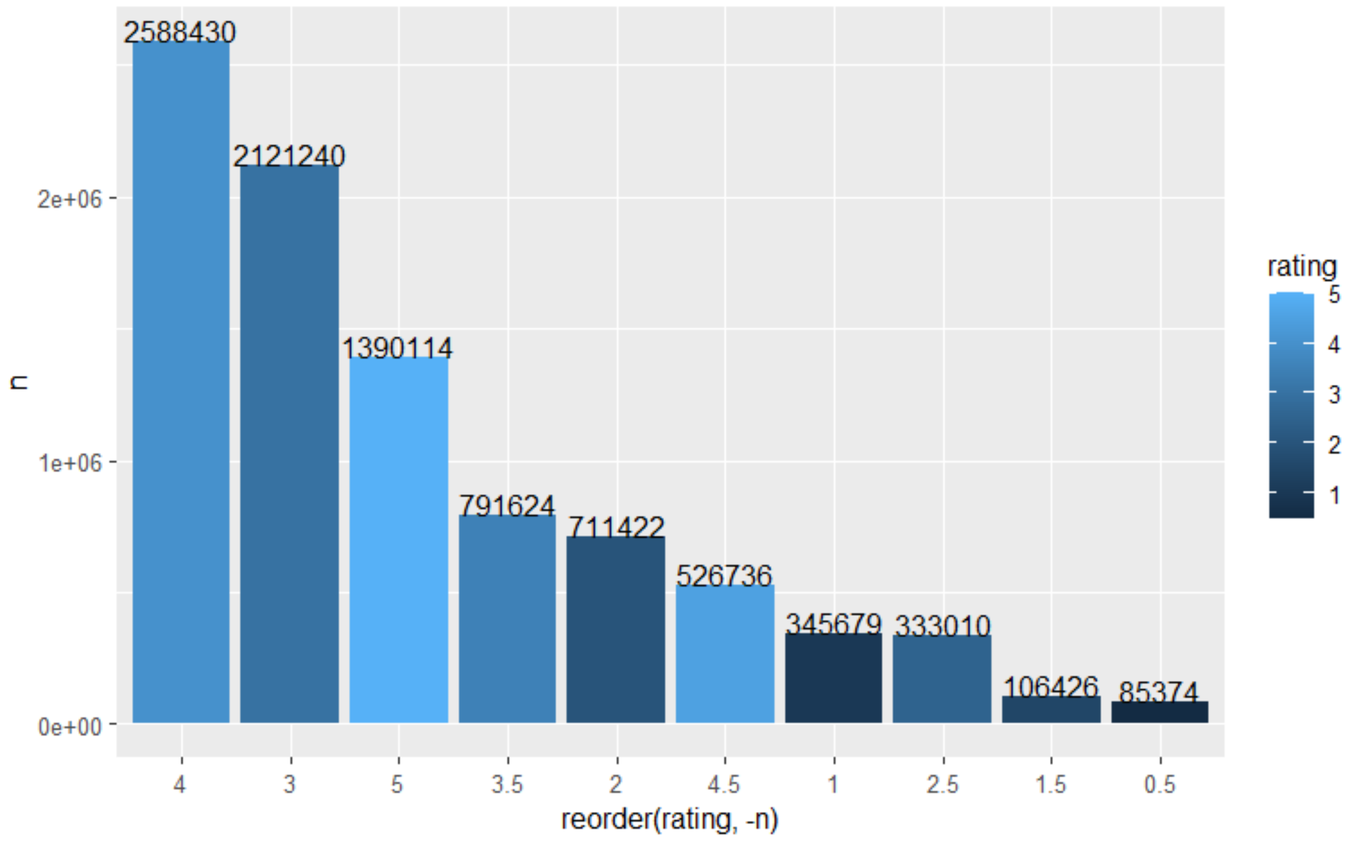
```
     userId         movieId          rating        timestamp           title                ge
nres
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08   Length:9000055     Lengt
h:9000055
 1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08   Class :character   Class
:character
 Median :35738   Median : 1834   Median :4.000   Median :1.035e+09   Mode  :character   Mode
:character
 Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
```

Hide

```
#Convert all character variables to factors
edx <- edx %>% mutate_if(is.character,as.factor)
```

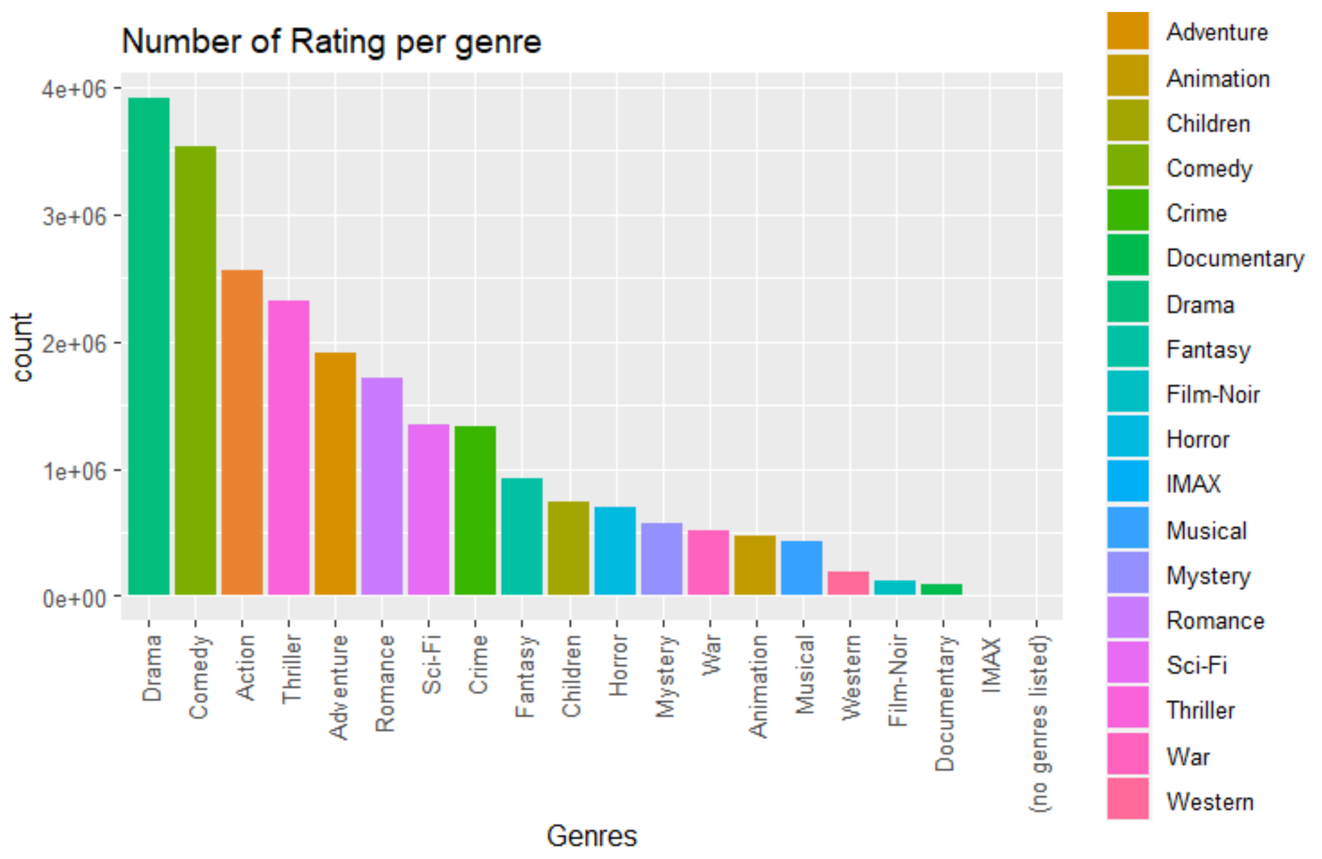
Hide

```
#Frequencies of different ratings
edx %>% group_by(rating) %>%
  tally() %>%
  ggplot() +
  aes(x = reorder(rating, -n), y = n, fill = rating) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = n), vjust = 0)
```



Hide

```
#Counts of different genres (slow but comprehensive)
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = reorder(genres, -count), y = count)) +
  geom_bar(aes(fill = genres), stat = "identity") +
  ggtitle("Number of Rating per genre") +
  xlab("Genres") +
  theme(axis.text.x  = element_text(
    angle = 90,
    vjust = 0.5,
    hjust = 1
  ))
```



Hide

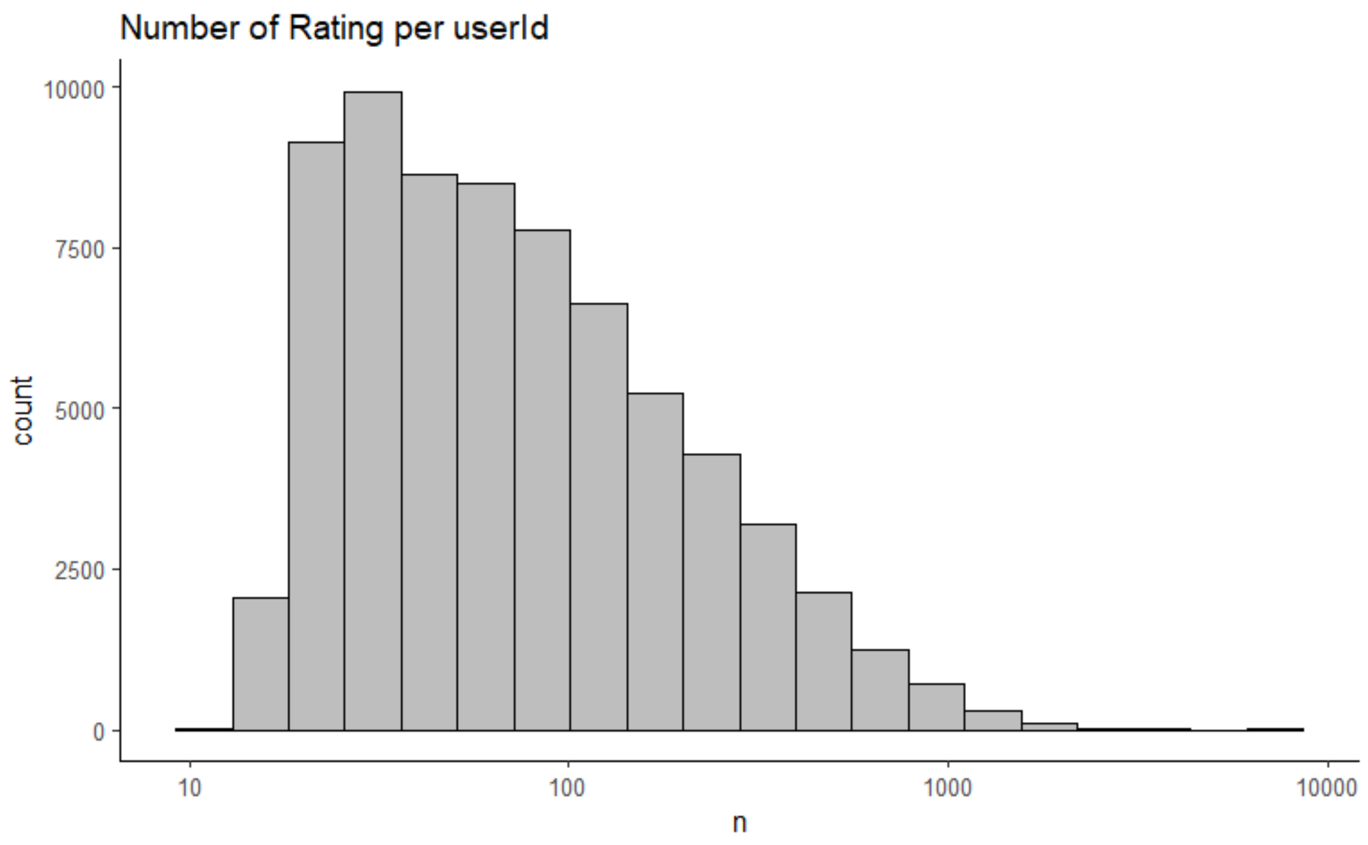
```
# Number of users and movies according to their IDs
edx %>%
  summarise(Users = n_distinct(userId),
            Movies = n_distinct(movieId))
```

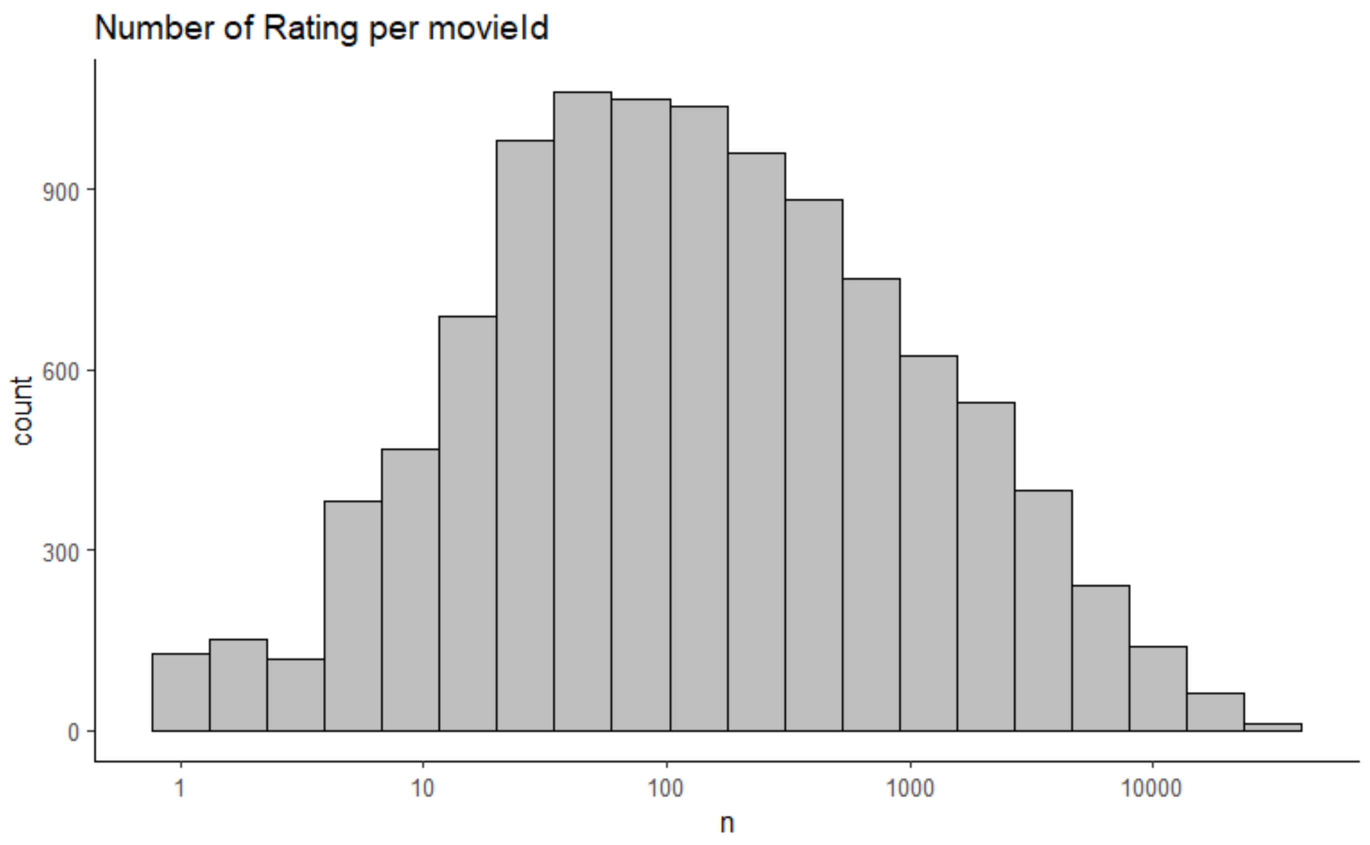
Users <int>	Movies <int>
69878	10677

1 row

Hide

```
#Number of ratings per movie and per user, respectively
for (i in c("userId", "movieId")){
  p <- edx %>%
    count(.data[[i]]) %>% ggplot(aes(n))+
  geom_histogram(bins = 20 , color = "black", fill = "gray")+
  scale_x_log10()+
  ggtitle(paste0("Number of Rating per ", i)) +
    theme_classic()
  print(p)
}
```

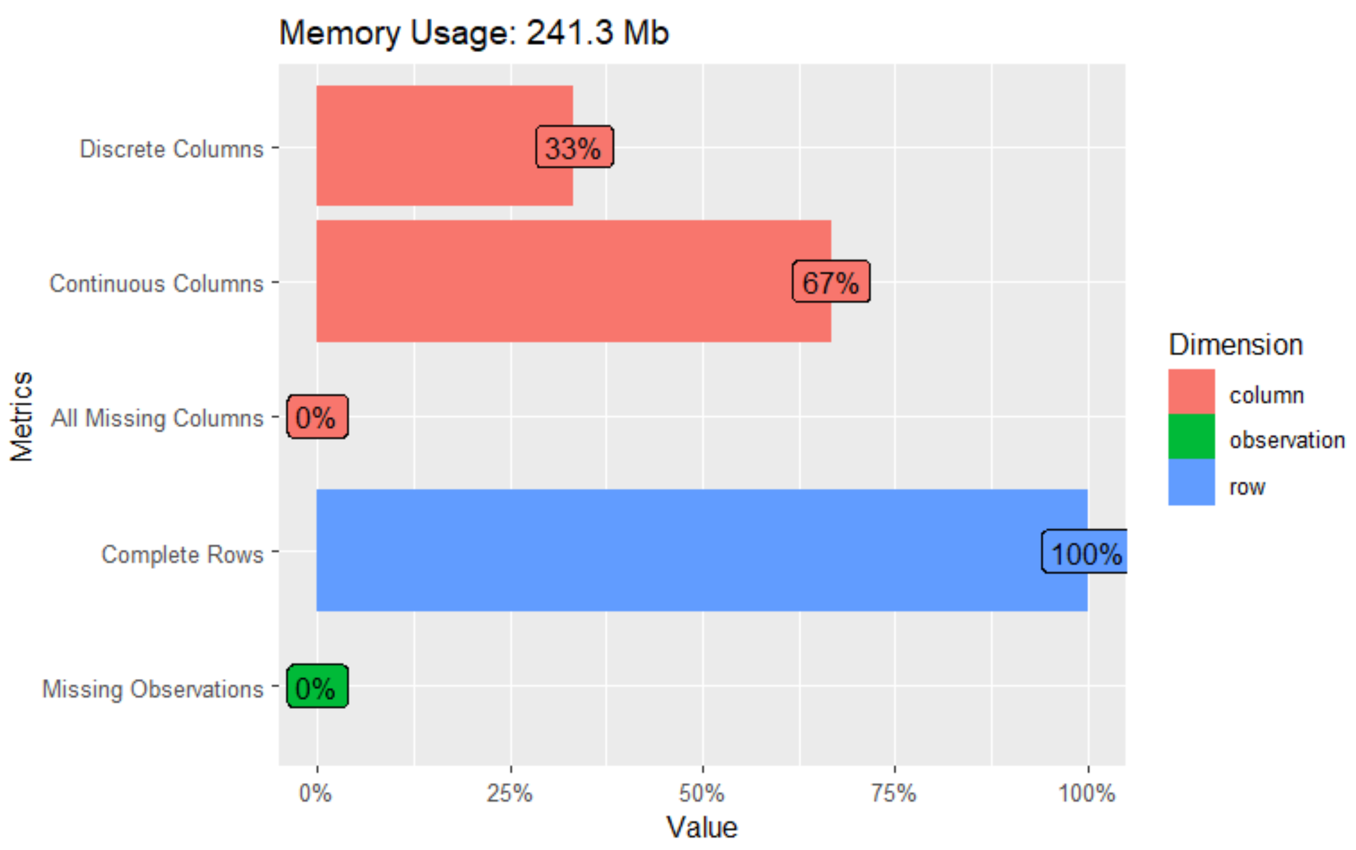




```
#Automated Exploratory Data Analysis (EDA)
if(!require(DataExplorer)) install.packages("DataExplorer", repos = "http://cran.us.r-projec
t.org")
```

```
Loading required package: DataExplorer
Registered S3 method overwritten by 'htmlwidgets':
  method           from
  print.htmlwidget tools:rstudio
```

```
library(DataExplorer)
plot_intro(edx)
```



Modelling and prediction

Data participation before modeling

This step will split the dataset to two parts: train_set and test_set. And the split percentage is 80%, which gives a ratio of train_set:test_set to 4:1.

Hide

```
# A method from caret
set.seed(123)
inTrain = createDataPartition(y = edx$rating, p = .80, list = FALSE)
train_set = edx[inTrain,]
test_set = edx[-inTrain,]
```

RMSE calculation function

Root Mean Square Error (RMSE) is used to measure the error of a model in predicting quantitative data. RMSE indicates the accuracy. The lower the RMSE, the better the accuracy of a model and its prediction. The equation is RMSE = √[Σ(Pi – Oi)^2 / n], where:

```
Σ symbol indicates “sum”
Pi is the predicted value for the ith observation in the dataset
Oi is the observed value for the ith observation in the dataset
n is the sample size
```

Hide

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

Model 1: Average rating

This model assumes the same rating for all movies and users with all the differences explained by random variation. The regression equation is: $Y\ u,i = \mu + \varepsilon\ u,i$

$\varepsilon\ i,u$ is independent errors sampled from the same distribution centered at 0; μ is the "true" rating for all movies

Hide

```
# Calculate mu.
mu <- mean(train_set$rating) %>% print()
```

```
[1] 3.51266
```

Hide

```
# If all unknown ratings are predicted with mu, then
rmse_model_1 <- RMSE(test_set$rating, mu)%>% print()
```

```
[1] 1.061238
```

A result table is created to collect the RMSE results from different modelling

Hide

```
rmse_results <- data.frame(method = "Average rating", RMSE = rmse_model_1) %>%  print()
```

method <chr>	**RMSE** <dbl>
Average rating	1.061238

1 row

Model 2: Movie effect

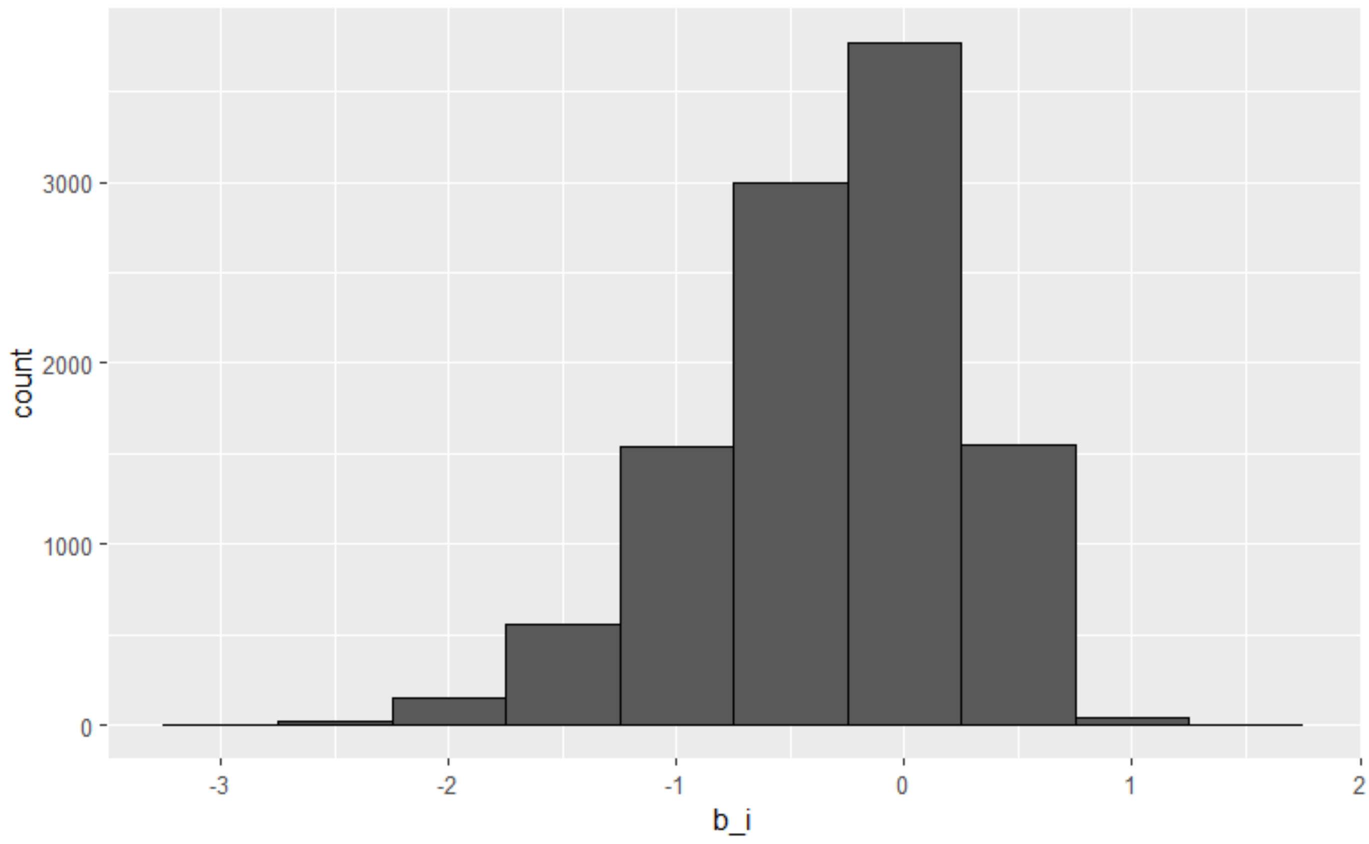
During data exploration, we find some movies have higher ratings than others. Therefore the previous simple model can be modified by adding a term $b\ i$ which represents the average ranking for movie i. Statistically, they are itemized as an effect, while described as "Bias" in the Netflix challenge paper so noted as "b". The regression model equation will be modified to:

$$Y\ u,i = \mu + b\ i + \varepsilon\ u,i$$

There are thousands of $b\ i$ as each movie gets one, so the averages are calculated in order to increase the speed of the regression using lm().

Hide

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

# Visualization of the distribution of b_i
movie_avg %>% ggplot()+
  aes(x=b_i)+
  geom_histogram(bins = 10, color = I("black"))
```



Hide

```
# The prediction is then calculated using the $Y~u,i~ = μ~ + b~i$
fit_movies <- data.frame(movieId = movie_avg [,1],
                         mu = mu, b_i = movie_avg$b_i)

rmse_model_2 <- left_join(test_set, fit_movies, by = "movieId")  %>%
  mutate(pred = mu + b_i)  %>%
  summarize(rmse = RMSE(rating, pred)) %>%
  print()
```

```
rmse
<dbl>
```

rmse <dbl>
0.9442583

1 row

Hide

```
rmse_results <- bind_rows(rmse_results, tibble(method="Movie effect model",  RMSE = rmse_mode
l_2 [1,])) %>%
  arrange(RMSE) %>%
  print()
```

method <chr>	RMSE <dbl>
Movie effect model	0.9442583
Average rating	1.0612380

2 rows

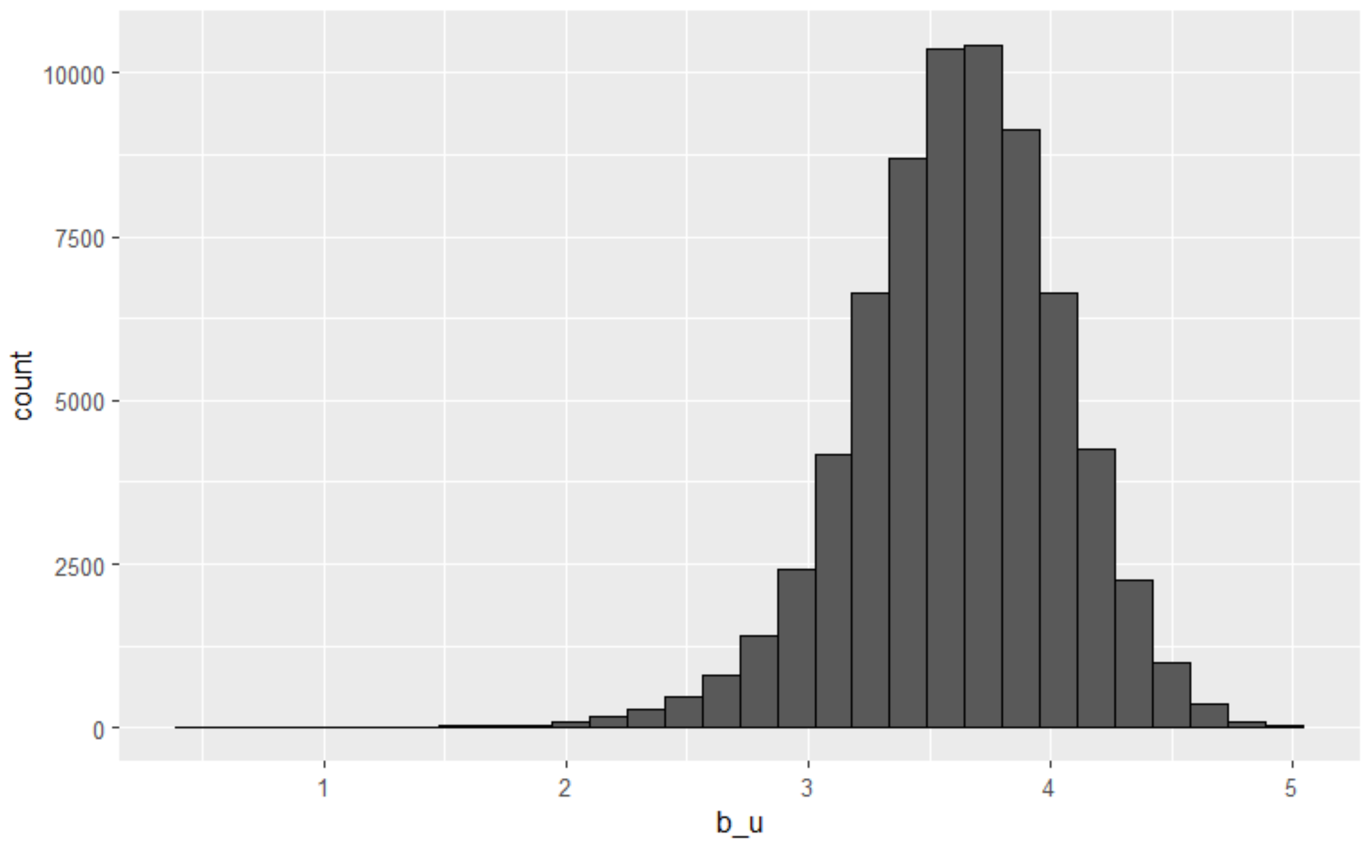
Model 3: User effect

This model will further consider the effect of the users.

Let's compute the average rating for user u for those that have rated 100 or more movies.

Hide

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = I("black"))
```



As there is considerable variability of the across users ratings. The model may be further improved by adding this factor:

$$Y\,u,i = \mu + b\,i + b\,u + \varepsilon\,u,i$$

To fit this model, lm could be like this:

lm(rating ~ as.factor(movieId) + as.factor(userId))

Again, to improve the speed instead, we will compute an approximation by computing $^{\mu}$ and $^{b}_{i}$ and estimating ^{b}u as the average of $y\,u,i\,-^{\mu}\,-^{b}i$.

Hide

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

rmse_model_3 <- left_join(test_set, fit_movies, by = "movieId")  %>%
  left_join(fit_users, by = "userId")  %>%
  mutate(pred = mu + b_i + b_u) %>%
  summarize(rmse = RMSE(rating, pred)) %>%
  print()
```

```
     rmse
    <dbl>
0.8666644
```

1 row

Hide

```
rmse_results <- bind_rows(rmse_results, tibble(method="Movie plus user effects model",  RMSE
= rmse_model_3 [1,])) %>%
  arrange(RMSE) %>%
  print()
```

method <chr>	**RMSE** <dbl>
Movie plus user effects model	0.8666644
Movie effect model	0.9442583
Average rating	1.0612380

3 rows

Hide

```
NA
```

Model 4: Matrix factorization

A more advanced model could be matrix factorization. This model will process the data as a large and parse matrix, and then decompose it into two smaller dimensional matrices with latent features and less sparsity. The Package “recosystem” can be used to run this model. The details of this package can be found at https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html)

Hide

```
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.or
g")
library(recosystem)

set.seed(1, sample.kind="Rounding")
train_reco <- with(train_set, data_memory(user_index = userId, item_index = movieId, rating =
rating))
test_reco <- with(test_set, data_memory(user_index = userId, item_index = movieId, rating = r
ating))
r <- Reco()

para_reco <- r$tune(train_reco, opts = list(dim = c(20, 30),
                                            costp_l2 = c(0.01, 0.1),
                                            costq_l2 = c(0.01, 0.1),
                                            lrate = c(0.01, 0.1),
                                            nthread = 8,
                                            niter = 10))

r$train(train_reco, opts = c(para_reco$min, nthread = 8, niter = 30))
results_reco <- r$predict(test_reco, out_memory())

rmse_model_4 <- RMSE(results_reco, test_set$rating)

rmse_results <- bind_rows(rmse_results, tibble(method = "Matrix factorization model", RMSE =
rmse_model_4))
```

Model 5: Noise-regularized modelling

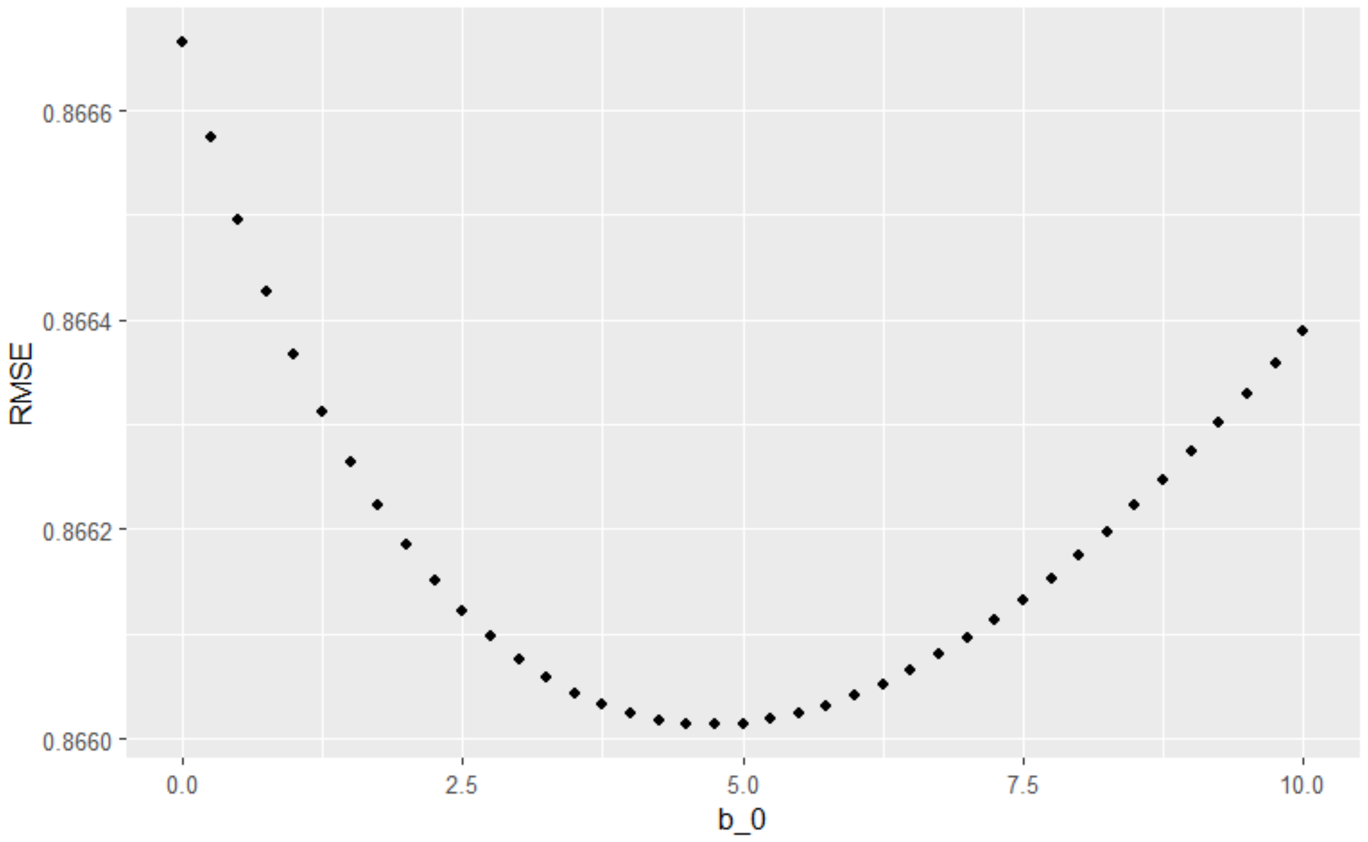
In the dataset, there might be some noises because some movies were just voted by a few users. These noises may not represent the true properties of your data, but rather random chance. A model may run too hard to capture some noises in the training dataset, which is called overfitting. Overfitting can decrease the accuracy of the prediction. One of the methods to minimize overfitting is noise-regularized modelling, which regularizes or shrinks the variances of coefficient estimates close to 0. The regression equation will be like this:

$Y\,u, i = \mu + (b\,i + b\,0) + (b\,u + b\,0) + \varepsilon\,u, i$

If training data have noise, the adjustment of the model using the $b\,0$ will make the model less flexible, and minimize potential overfitting. However, $b\,0$ is like a tuning parameter. As its value increases, it reduces the variance (overfitting) until reaching an optimal point without losing important properties of the dataset. But beyond that point, the model starts losing true properties and raises bias (underfitting). So the optimal value of $b\,0$ should be carefully selected.

Hide

```
# Set up the range of the b_0
b_0 <- seq(0, 10, 0.25)

# Calculate RMSE results with the change of b_0.
rmses <- data.frame()

for (j in b_0) {
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + j))

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

  pred_rating <- list(test_set, b_i, b_u) %>% reduce(left_join) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  res <- RMSE(pred_rating, test_set$rating)

  rmses <- rmses %>% rbind(c(j, res))
}

# Plot the results
rmses %>% ggplot(aes_string(x = names(.)[1], y = names(.)[2])) +
geom_point() +
  xlab ("b_0")+
  ylab ("RMSE")
```



Hide

```
# Add the best rmse result to the table
rmse_results <-
  bind_rows(
    rmse_results,
    tibble(method = "User and movie effects plus regularized model", RMSE = min(rmses[, 2]))
  )
```

Results

The RMSE results after the predictions using the different models are shown below. The smaller RMSE, the higher accuracy.

Hide

```
rmse_results %>%
  arrange(RMSE) %>%
  print()
```

method <chr>	RMSE <dbl>
Matrix factorization model	0.7902752
User and movie effects plus regularized model	0.8660133

method <chr>	RMSE <dbl>
Movie plus user effects model	0.8666644
Movie effect model	0.9442583
Average rating	1.0612380

5 rows

Conclusion

As indicated by the results, the more possible actual factors are considered, the smaller RMSE (the higher accuracy) could be achieved. Among the models evaluated, the accuracy orders are Matrix factorization > Noise regularization > Movie plus user effects model > Movie effect model > Average rating. The lowest RMSE value achieved here is ~ 0.79 from the matrix factorization model. Interestingly, The addition of noise regularization to the “Movie plus user effects model” shows marginal benefit in this case.

Some advanced tools may allow users to evaluate multiple algorithms/models and identify the best automatically (e.g. H2O’s AutoML), However, extremely low speed could be an issue when a dataset is too large (e.g. 9000055 observations in this edx data). The movie recommendation system might be further evaluated for potential improvement if some more information on the actual ratings could be available, for example, the years of ratings which may influence rating outcome.