Report

Sho Nakamura

2023-07-17

Introduction

For this project, we will be creating a movie recommendation system using the MovieLens dataset. The version of movielens included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. We can find the entire latest MovieLens dataset here. We will be creating your own recommendation system using all the tools we have shown us throughout the courses in this series. We will use the 10M version of the MovieLens dataset to make the computation a little easier.

We will download the MovieLens data and run code we will provide to generate our datasets.

Important: The final__holdout__test data should NOT be used for training, developing, or selecting our algorithm and it should ONLY be used for evaluating the RMSE of our final algorithm. The final__holdout__test set should only be used at the end of our project with our final model. It may not be used to test the RMSE of multiple models during model development. Weshould split the edx data into separate training and test sets and/or use cross-validation to design and test our algorithm.

Goal of project

We will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the final hold-out test set.

Method

Describe the process of data explosion, data visualization and modeling.

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")
```

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

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
```

```
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

```
## Loading required package: knitr
```

```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:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

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

```
## Loading required package: recosystem
```

```r
library(tidyverse)
library(caret)
library(ggplot2)
library(data.table)
```

```r
library(knitr)
library(recosystem)

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

options(timeout = 1000)

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
```

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

```r
# 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")
```

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

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```r
edx <- rbind(edx, removed)

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

Data exploration and visualization

```r
# Check the first 5 rows of the data set
head(edx)
```

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

```r
# Check the data set
summary(edx)
```

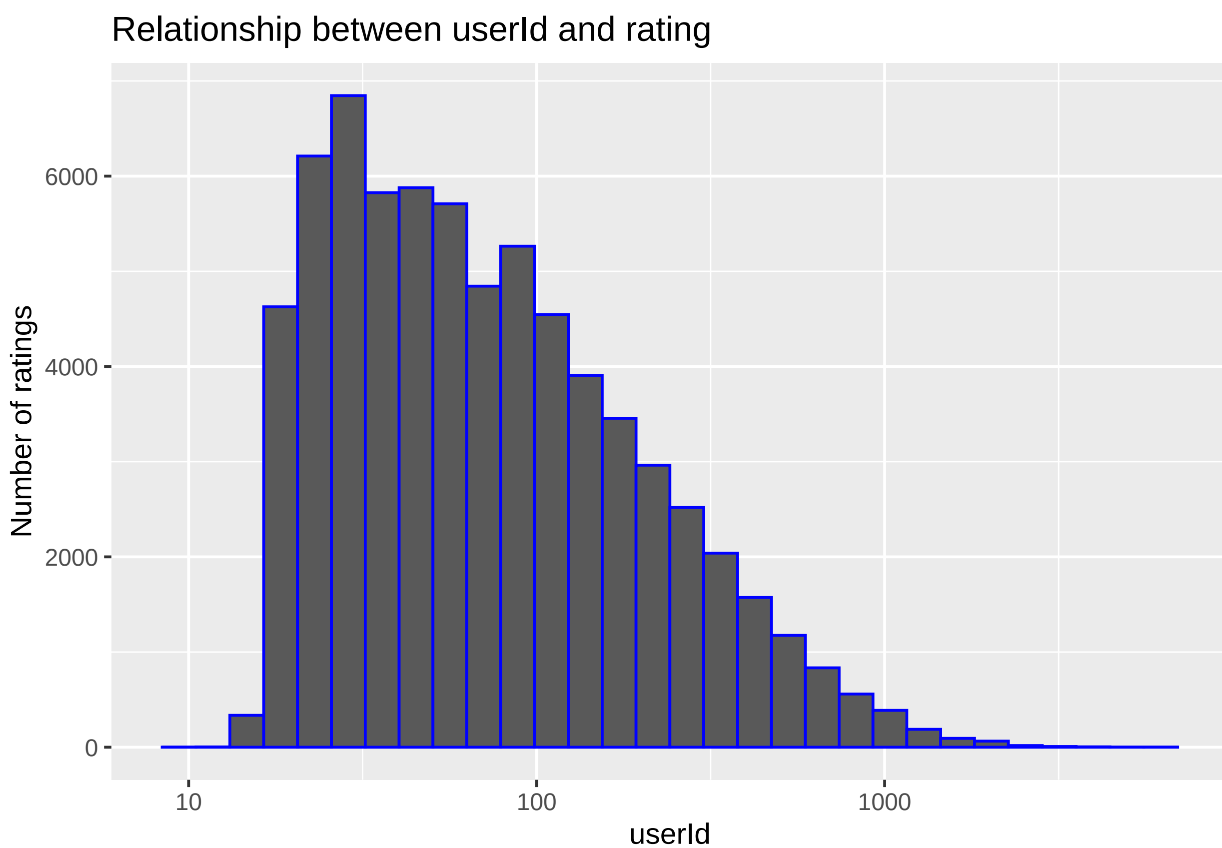
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 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
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

It turns out that there is no one with a rating of 0.

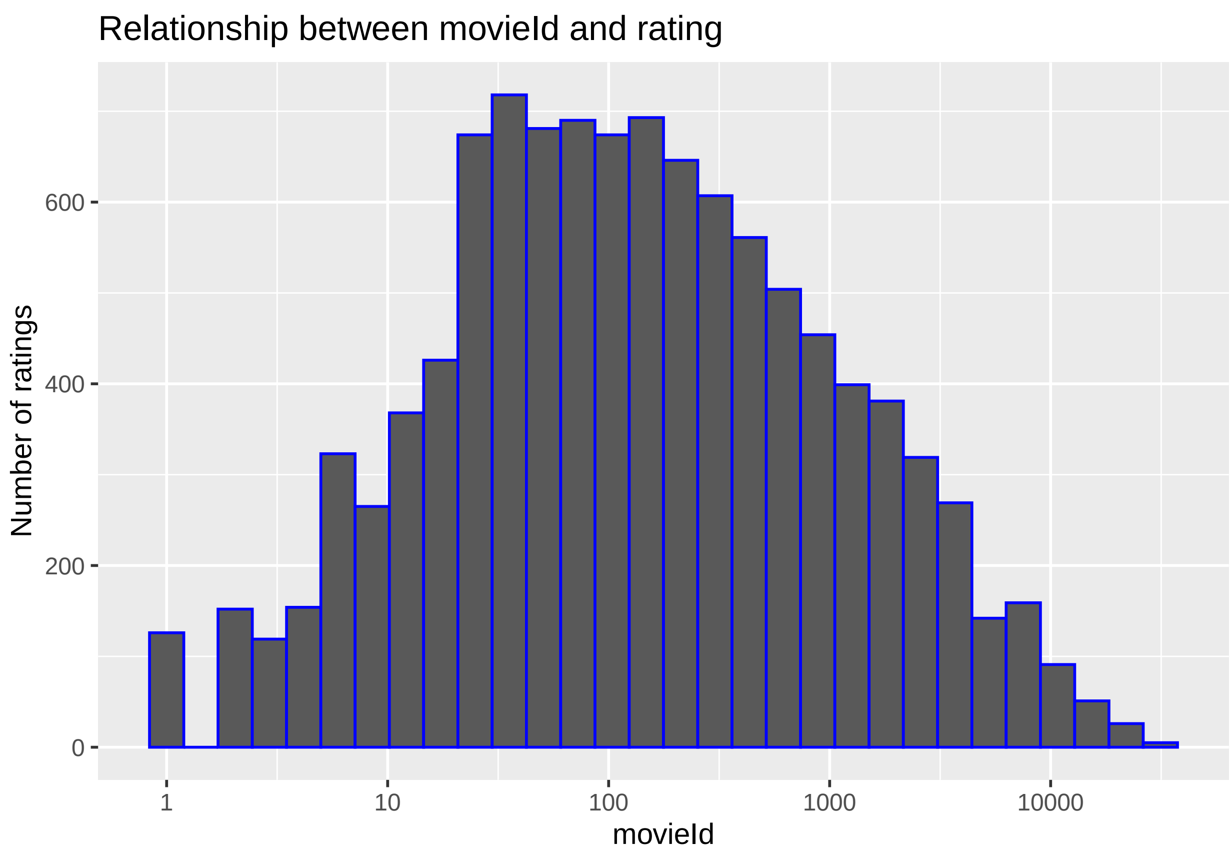
Check the number of unique users and movies
```r
# Check the number of unique users and movies
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

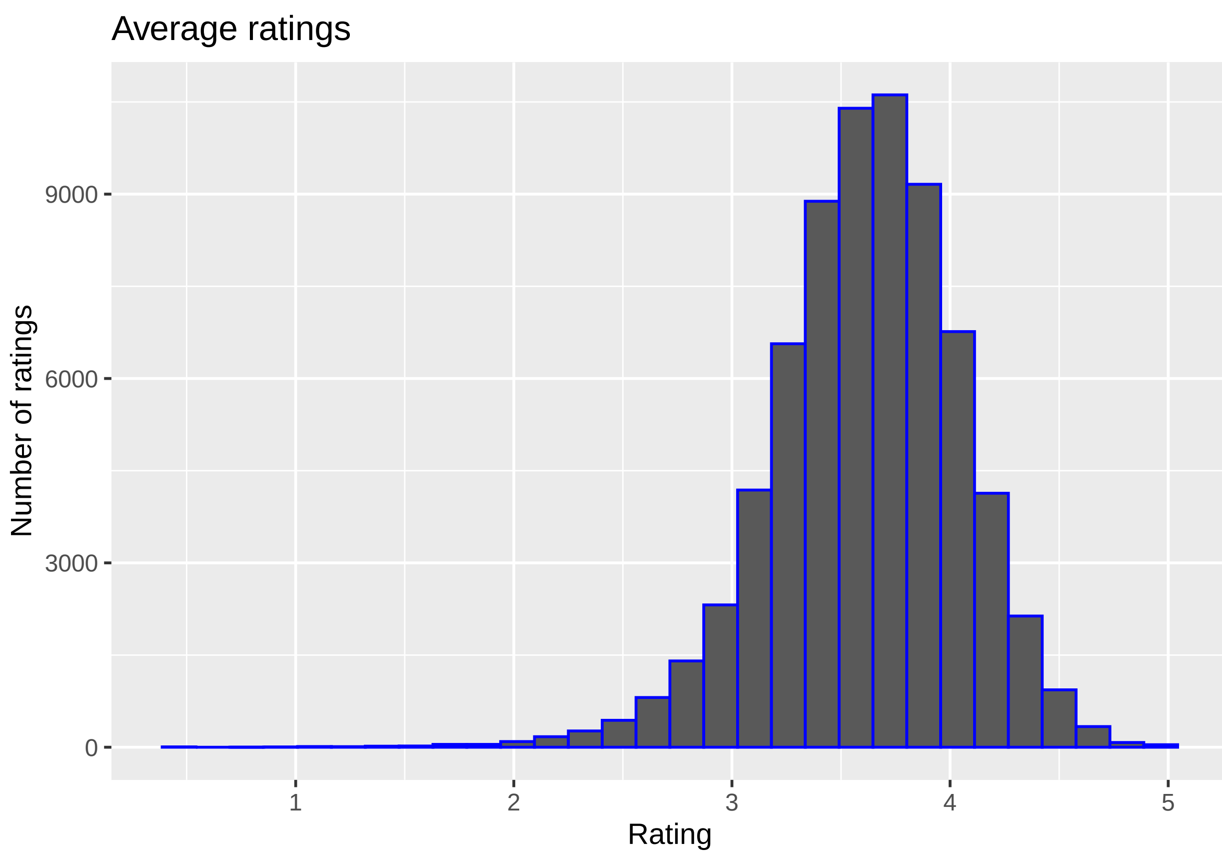
```r
# Check the relationship between userId and rating
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = 'blue') +
  scale_x_log10() +
  ggtitle("Relationship between userId and rating") +
  labs(x = "userId", y = "Number of ratings")
```



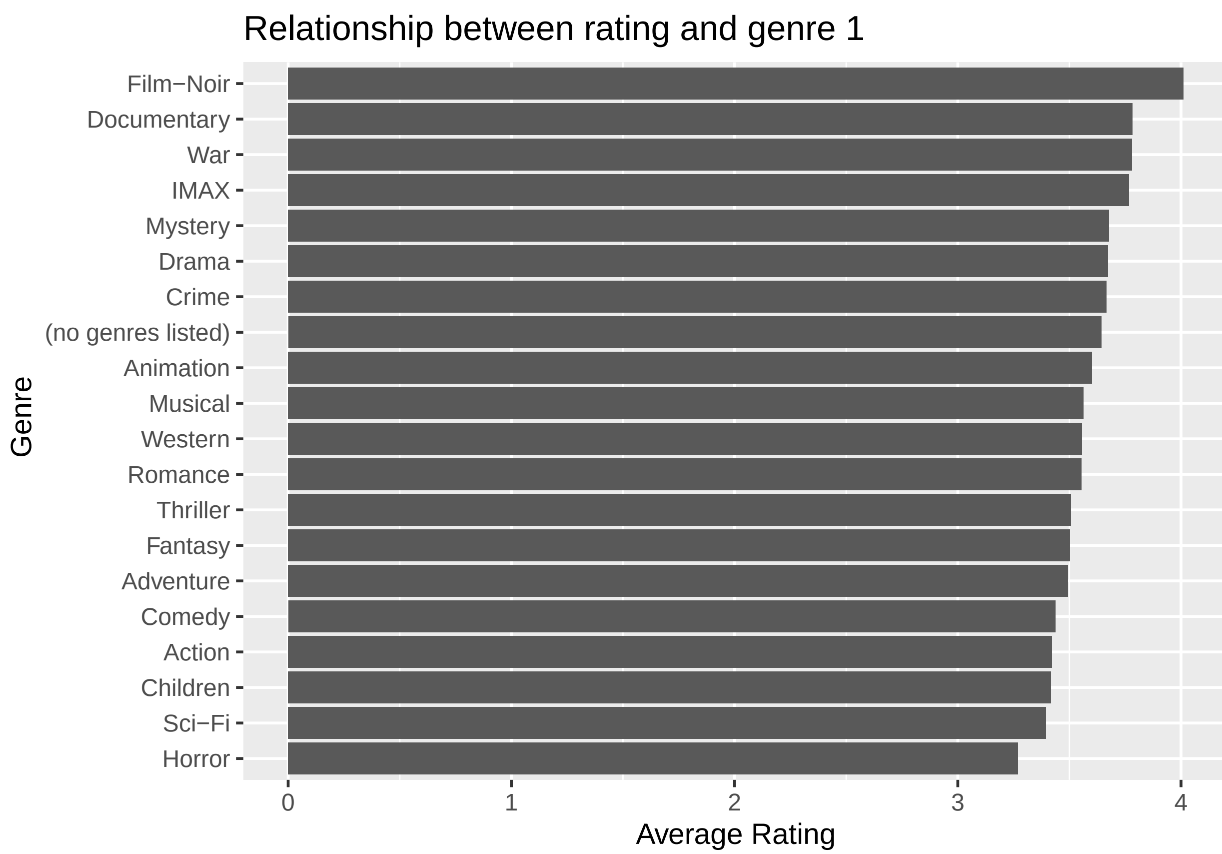

```r
# Check the relationship between movieId and rating
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = 'blue') +
  scale_x_log10() +
  ggtitle("Relationship between movieId and rating") +
  labs(x = "movieId", y = "Number of ratings")
```



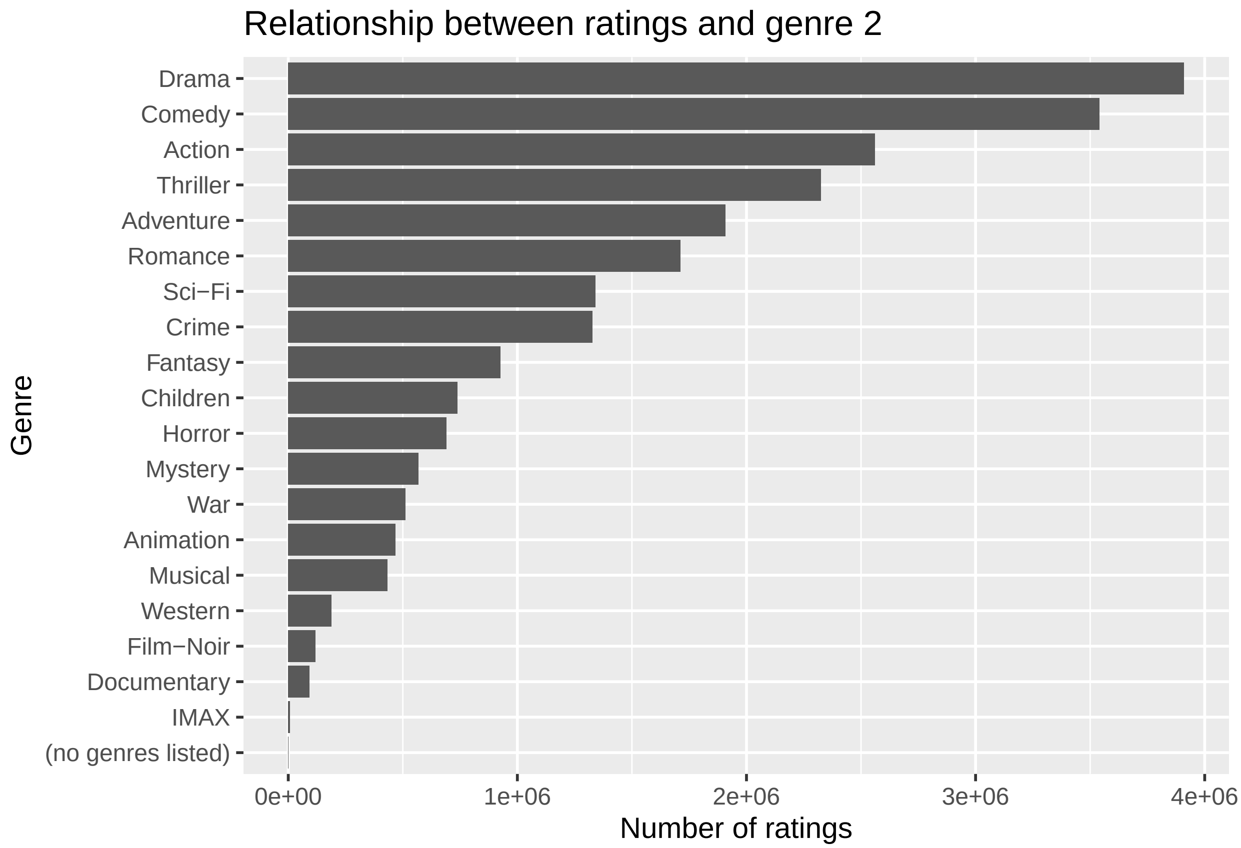

```r
# Check the rating histogram
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = 'blue') +
  ggtitle("Average ratings") +
  labs(x = "Rating", y = "Number of ratings")
```



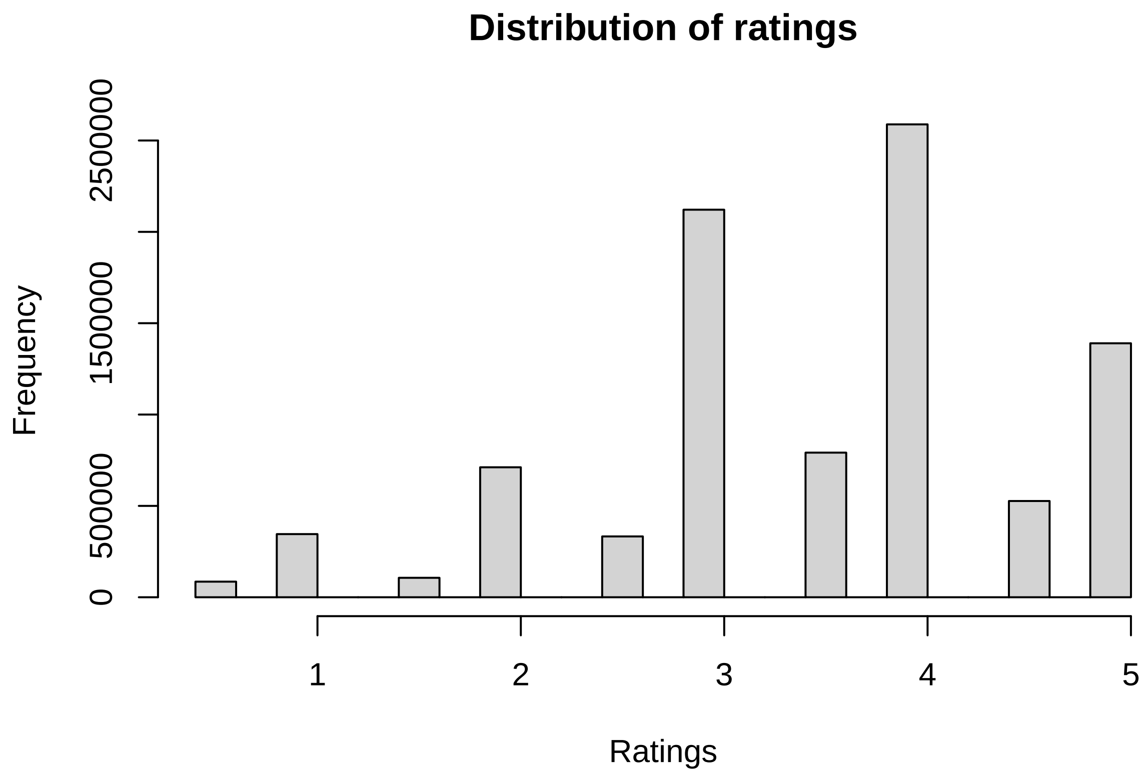

```
# Check the relationship between genre and rating
genres <- edx$genres %>% str_replace("\\|.*","") %>% unique()

nb_genres <- sapply(genres, function(x){
  index <- str_which(edx$genres, x)
  length(edx$rating[index])
})

genres_ratings <- sapply(genres, function(x){
  index <- str_which(edx$genres, x)
  mean(edx$rating[index], na.rm = T)
})

genres_table <- data.frame(genres = genres, n_genres = nb_genres, avg_rating = genres_ratings)

genres_table %>% ggplot(aes(x= reorder(genres,avg_rating), y = avg_rating)) +
  geom_col()+ labs(x=" Genre", y="Average Rating") +
  ggtitle("Relationship between rating and genre 1") +
  coord_flip()
```




```
genres_table %>% ggplot(aes(x= reorder(genres,n_genres), y = n_genres)) +
  geom_col()+ labs(x="Genre", y="Number of ratings") +
  ggtitle("Relationship between ratings and genre 2") +
  coord_flip()
```




```
# Check the distribution of ratings
hist(edx$rating, main="Distribution of ratings", xlab="Ratings")
```



As far as the rating is concerned, it is not biased data with too many 1s and 5s, so it seems to be a good data set.

Build model

Create a train set and test set for use in modeling. Also define RMSE.

```
# Create train set and test set
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

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

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

```r
## 1. Naive baseline model ##
# Compute the mean rating of the dataset
mu <- mean(train_set$rating)
mu
```

1. Naive baseline model

```
## [1] 3.512574
```

```r
# Test the results
nb_rmse <- RMSE(test_set$rating, mu)

# Save prediction to data frame
rmse_results <- tibble(Model = "Naive Baseline Model", RMSE = nb_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   Model                 RMSE
##   <chr>                <dbl>
## 1 Naive Baseline Model  1.06
```

```r
## 2. Movie Effect Model ##
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)

# Test the results
me_rmse <- RMSE(predicted_ratings, test_set$rating)

# Save prediction to data frame
rmse_results <- bind_rows(rmse_results, tibble(Model = "Movie Effect Model", RMSE = me_rmse))
rmse_results
```

2. Movie Effect Model

```
## # A tibble: 2 x 2
##   Model                 RMSE
##   <chr>                <dbl>
## 1 Naive Baseline Model 1.06
## 2 Movie Effect Model   0.944
```

```r
## 3. Movie and User Effect Model ##
user_avgs <- train_set %>%
```

```r
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Test the results
m_u_rmse <- RMSE(predicted_ratings, test_set$rating)

# Save prediction to data frame
rmse_results <- bind_rows(rmse_results, tibble(Model = "Movie and User Effect Model", RMSE = m_u_rmse))
rmse_results
```

3. Movie and User Effect Model

```
## # A tibble: 3 x 2
##   Model                        RMSE
##   <chr>                       <dbl>
## 1 Naive Baseline Model        1.06
## 2 Movie Effect Model          0.944
## 3 Movie and User Effect Model 0.866
```

```r
## 4. Regularization Model ##
lambdas <- seq(0, 10, 0.25)
best_lambda <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + l))
  b_u <- train_set %>%
    left_join(b_i, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + l))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

# Find the optimal lambda
lambda <- lambdas[which.min(best_lambda)]
lambda
```

4. Regularization Model

```
## [1] 4.75
```

```r
# Save prediction to data frame
rmse_results <- bind_rows(rmse_results, tibble(Model = "Regularization Model", RMSE = min(best_lambda))
rmse_results
```

```
## # A tibble: 4 x 2
##   Model                        RMSE
##   <chr>                       <dbl>
## 1 Naive Baseline Model        1.06
## 2 Movie Effect Model          0.944
## 3 Movie and User Effect Model 0.866
## 4 Regularization Model        0.866
```

```r
## 5. Matrix Factorization Model ##
set.seed(1)
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating,
                                           date = date))

test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating,
                                           date = date))

# Build recommender
r <-  Reco()

r$train(train_data)
```

5. Matrix Factorization Model

```
## iter      tr_rmse          obj
##    0       0.9716   1.1949e+07
##    1       0.8827   1.0697e+07
##    2       0.8619   1.0528e+07
##    3       0.8472   1.0377e+07
##    4       0.8402   1.0319e+07
##    5       0.8348   1.0274e+07
##    6       0.8307   1.0247e+07
##    7       0.8277   1.0227e+07
##    8       0.8256   1.0207e+07
##    9       0.8241   1.0197e+07
##   10       0.8230   1.0188e+07
##   11       0.8221   1.0180e+07
##   12       0.8213   1.0176e+07
##   13       0.8207   1.0170e+07
##   14       0.8203   1.0169e+07
##   15       0.8198   1.0163e+07
```

```
##   16       0.8195   1.0163e+07
##   17       0.8191   1.0157e+07
##   18       0.8189   1.0155e+07
##   19       0.8186   1.0152e+07
```

```
predicted_ratings <-  r$predict(test_data, out_memory())
matrix_rmse <- RMSE(test_set$rating, predicted_ratings)

# Save prediction to data frame
rmse_results <- bind_rows(rmse_results, tibble(Model = "Matrix Factorization Model", RMSE = matrix_rmse)
rmse_results
```

```
## # A tibble: 5 x 2
##   Model                        RMSE
##   <chr>                       <dbl>
## 1 Naive Baseline Model        1.06
## 2 Movie Effect Model          0.944
## 3 Movie and User Effect Model 0.866
## 4 Regularization Model        0.866
## 5 Matrix Factorization Model  0.834
```

```
final_data  <-  with(final_holdout_test,  data_memory(user_index = userId,
                                                    item_index = movieId,
                                                    rating = rating,
                                                    date = date))

# predict final data set
predicted_ratings <- r$predict(final_data, out_memory())
final_rmse <- RMSE(final_holdout_test$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results, tibble(Model = "Matrix Factorization (Final Holdout Test)", RMSE
rmse_results
```

6. Matrix Factorization Model (Final Holdout Test)

```
## # A tibble: 6 x 2
##   Model                                      RMSE
##   <chr>                                     <dbl>
## 1 Naive Baseline Model                      1.06
## 2 Movie Effect Model                        0.944
## 3 Movie and User Effect Model               0.866
## 4 Regularization Model                      0.866
## 5 Matrix Factorization Model                0.834
## 6 Matrix Factorization (Final Holdout Test) 0.834
```

Results

RMSE < 0.86490 using Matrix FactorizationModel.

```r
rmse_results
```

```
## # A tibble: 6 x 2
##   Model                                   RMSE
##   <chr>                                  <dbl>
## 1 Naive Baseline Model                   1.06 
## 2 Movie Effect Model                     0.944
## 3 Movie and User Effect Model            0.866
## 4 Regularization Model                   0.866
## 5 Matrix Factorization Model             0.834
## 6 Matrix Factorization (Final Holdout Test) 0.834
```

Conclusion

This Capstone Project used a dataset called MovieLens to predict movie ratings. During that time, I created several models.

The goal of this project was to have RMSE < 0.86490. Amazingly, RMSE was 0.8340, and we were able to achieve our goal.

In the future, I think that it will be further improved by adjusting the parameters of the Matrix Factorization Model.