Capstone Project: Movielens

Stiina Kilk

November 13, 2020

Overview

The purpose of this report is to introduce the reader to the movie recommendation system developed by the author. This assignment is a part of HarvardX Professional Certificate in Data Science. Some base coding is provided within the courses in the program. Exploratory analysis is conducted in order to familiarize the reader with the dataset. The modeling along with the reasoning is explained in detail. As a conclusion, the author chooses the best performing model and gives final remarks.

Introduction

The aim of a recommendation system is to predict the user's preferences. This is done by fitting a model based on a dataset and making predictions using the model. The dataset for modeling and predicting is the 10M version of the MovieLens dataset, collected by GroupLens Research with user ratings from 1997 to 2008. The author defines an evaluation function, which is used to find the best-fitting model.

Data exploration and modeling

Loading necessary packages

```
library(tidyverse)
library(caret)
library(data.table)
library(dslabs)
library(tidyverse)
library(ggplot2)
library(dplyr)
library(stringr)
library(knitr)
library(data.table)
```

Dataset

The dataset is split into two parts: edx/training set and validation set. edx is used to develop models and determine the best fitting one. The validation set is to test the model in a so-called outside environment, which we have no control over. The validation set is used only after we have found the best-fitting model using edx.

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

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

ratings <- fread(text = gsub("::", "\t", 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")

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")
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)
```

Data Exploration

The data consists of 6 columns. Each row is an observation, or in this context, a rating given by one user. For example, according to the first row a rating of 5 was given to movie "Boomerang" with a movieId 122 by userId 1. The movie belongs to both comedy and romance genres and the time of the rating can be determined from the timestamp column.

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

```
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The edx set or training set has 9 000 055 rows or obervations and no missing values

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

```
## [1] FALSE
```

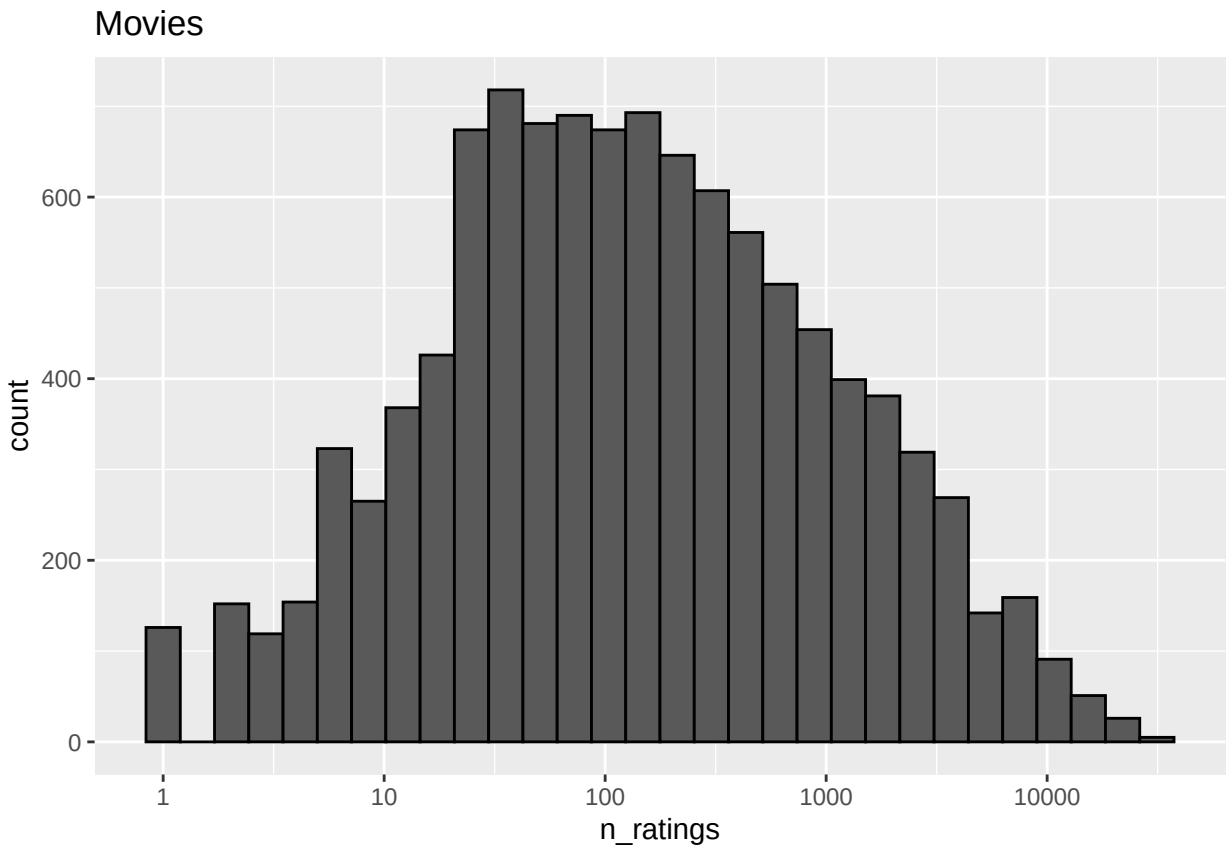
These are the ratings given to movies. No user has rated a movie with a 0.

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

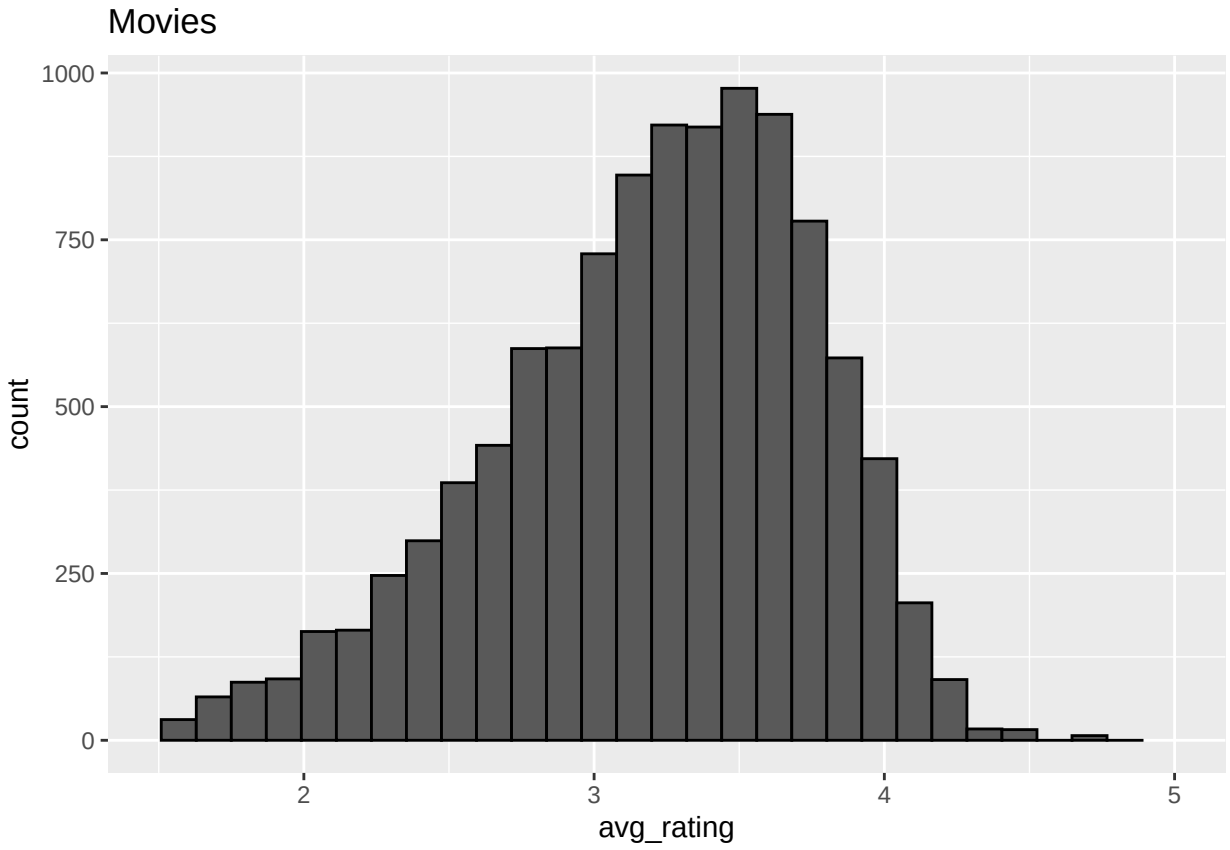
The total number of movies is 10 677 and the number of users 69 878. Therefore, many or all movies have multiple ratings and many or all users have rated multiple times.

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

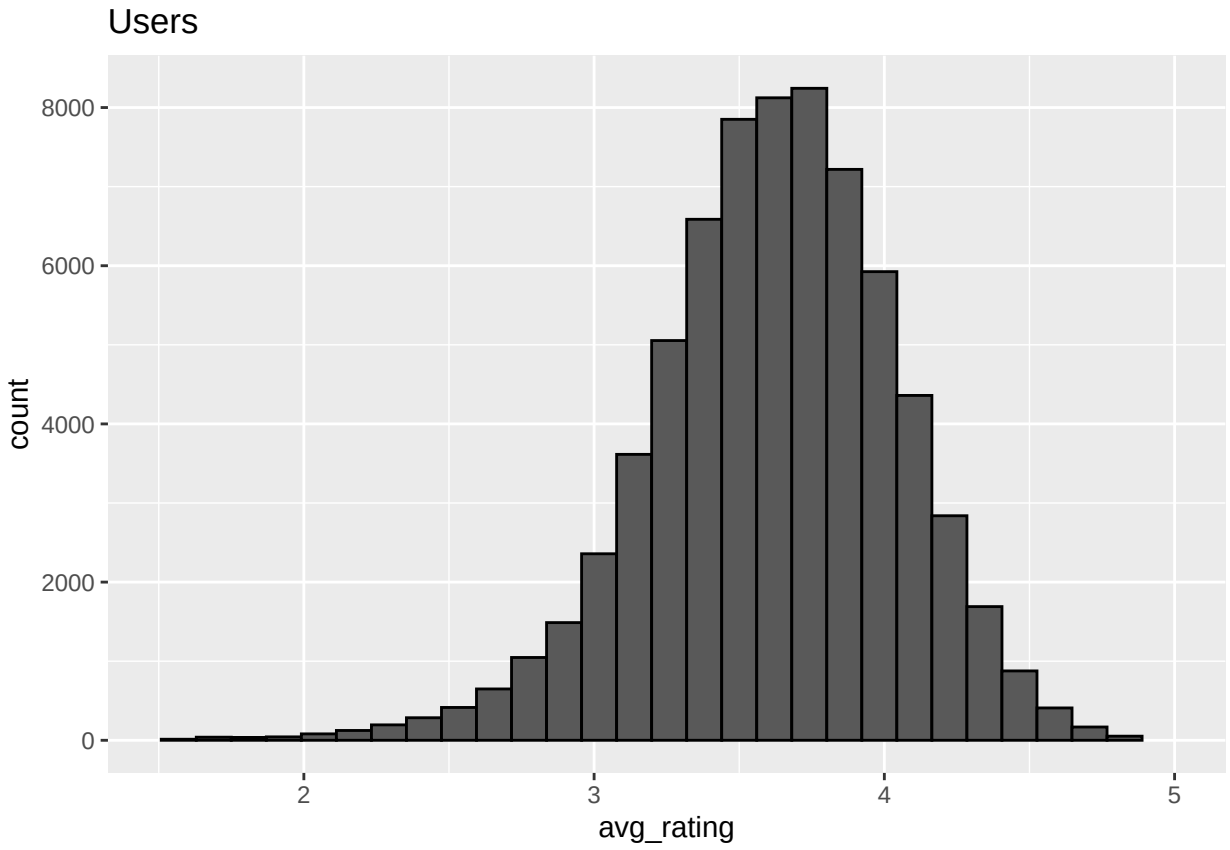
This is also visible in the distribution of ratings per movie. Some movies even have over 10 000 ratings.



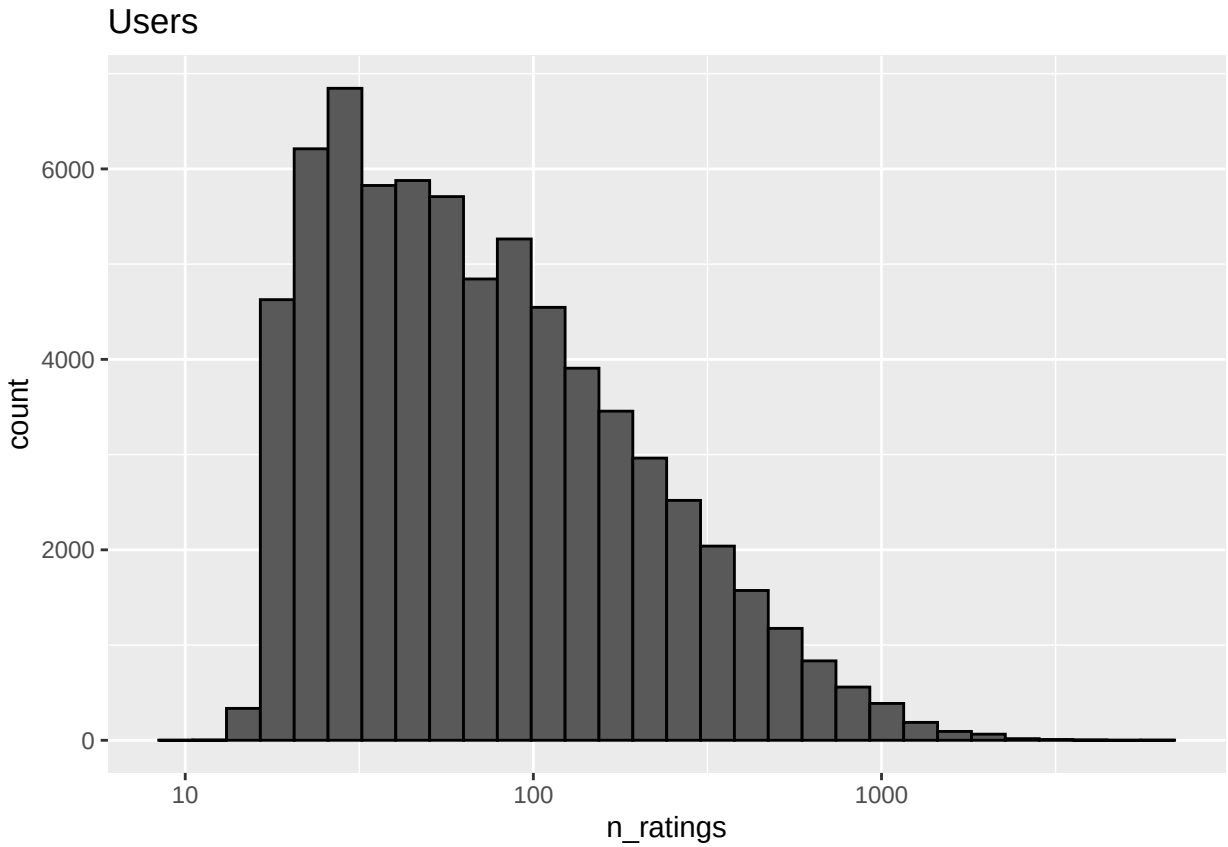
The average rating per movie is around 3.5. Some movies have exceptionally better ratings than others.



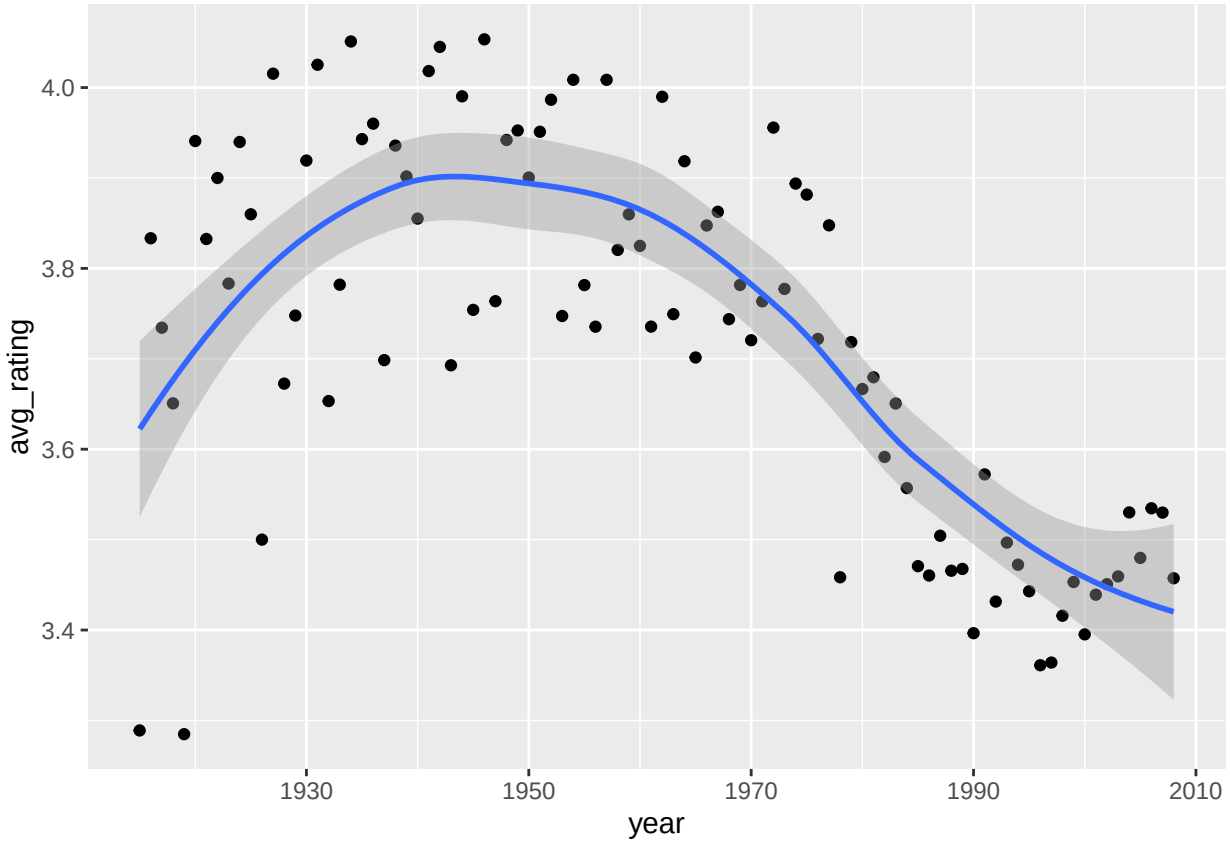
Users rate some movies higher than others.



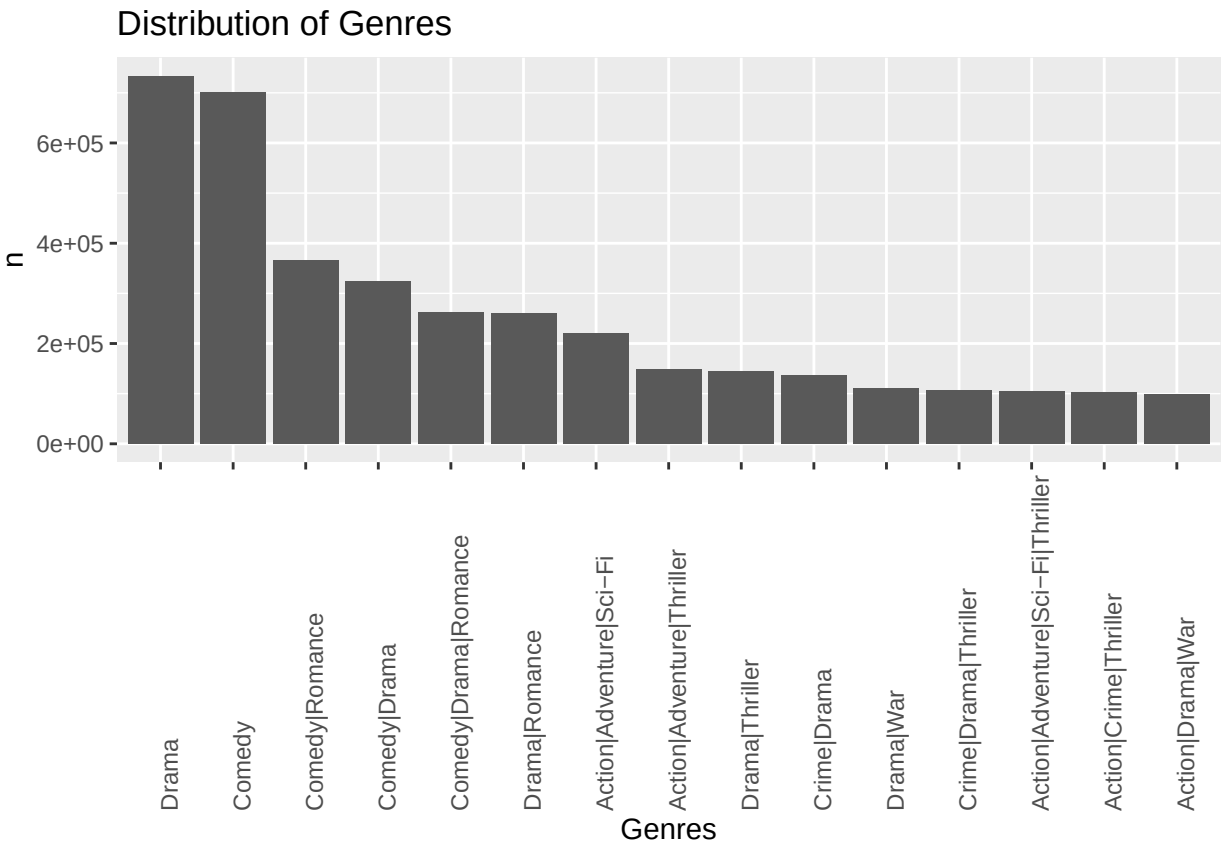
And some of the users are more diligent raters than others.



The release year of the movie has an influence over the rating as is evident in the graph below.



Most commonly rated genres are drama and comedy.



These are the ten highest rated genres. The differences between genres are notable.

```
## # A tibble: 10 x 2
##    genres                                   avg
##    <chr>                                  <dbl>
##  1 Animation|IMAX|Sci-Fi                   4.71
##  2 Drama|Film-Noir|Romance                 4.30
##  3 Action|Crime|Drama|IMAX                 4.30
##  4 Animation|Children|Comedy|Crime         4.28
##  5 Film-Noir|Mystery                       4.24
##  6 Crime|Film-Noir|Mystery                 4.22
##  7 Film-Noir|Romance|Thriller              4.22
##  8 Crime|Film-Noir|Thriller                4.21
##  9 Crime|Mystery|Thriller                  4.20
## 10 Action|Adventure|Comedy|Fantasy|Romance 4.20
```

As a contrast, the 10 lowest rated genres.

```
## # A tibble: 10 x 2
##    genres                                        avg
##    <chr>                                       <dbl>
##  1 Adventure|Animation|Children|Fantasy|Sci-Fi  1.92
##  2 Action|Adventure|Children                   1.92
##  3 Action|Children|Comedy                      1.91
##  4 Action|Adventure|Drama|Fantasy|Sci-Fi       1.90
##  5 Adventure|Drama|Horror|Sci-Fi|Thriller      1.75
```

```
##  6 Action|Drama|Horror|Sci-Fi                    1.75
##  7 Comedy|Film-Noir|Thriller                    1.64
##  8 Action|Horror|Mystery|Thriller               1.61
##  9 Action|Animation|Comedy|Horror               1.5
## 10 Documentary|Horror                           1.45
```

In conclusion, the rating of the movie depends not only on the movie itself, but also the user, the genre and the year of the release.

Evaluation function

Loss function is used throughout the analysis in order to evaluate the performance of the model. The ouptput of the function shows how much the prediction deviates from the actual result. The aim from the assignment is for the predictions to deviate less than 0.86490.The Loss function is defined as follows

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

Defining the data

The validation set can only be used after the final model has been picked, therefore only the edx set can be used for training. edx set is partitioned into two parts: training set and test set.

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)

train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test_set set are also in train_set set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test_set set back into train_set set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

First model: Proportions

A simple way of predicting the ratings would be to take the proportion of each available rating in training set and apply those proportions for predicting the new ratings.

```
#count the ratings in training_set
ratings <- train_set %>% group_by(rating) %>% summarise(n = n())
ratings <- as.data.frame(ratings)
pred <- c()
#create function for calculating predictions in test_set
predict <- function(x) {
prop <- ratings$n[x]/sum(ratings$n)
```

```
pred <- c(pred, rep(ratings$rating[x], nrow(test_set)*prop))
}
t <- 1:nrow(ratings)
predictions <- predict(t)
#calculating RMSE
prop_rmse <- RMSE(test_set$rating, predictions)
#create a table for storing the RMSE results of all models
rmse_results <- tibble(method = "Proportions", RMSE = prop_rmse)
prop_rmse
```

```
## [1] 1.498747
```

Unfortunately, this results in a deviation of 1.4987. The prediction deviates from the actual result by 1.5 stars, which means a better performing model is needed.

Second model: the Average

Another, even simpler way of predicting is taking the average of the training set and predicting that as a new rating for all movies.

```
#predict the same rating regardless of other variables
mu <- mean(train_set$rating)
#calulating RMSE
avg_rmse <- RMSE(test_set$rating, mu)
#adding to the table
rmse_results <- add_row(rmse_results, method = "Just the Average", RMSE = avg_rmse)
avg_rmse
```

```
## [1] 1.059904
```

This model is performing better by almost 0.5 stars, but and improvement can further be made.

Third Model: Adding the Movie Bias

According to the data exploration, some movies are rated higher than others while some of them lower. This introduces a movie bias, which is incorporated into the model.

```
#calculating movie bias for each movie
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
#making predictions
predictions <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

movie_rmse <- RMSE(test_set$rating, predictions)
rmse_results <- add_row(rmse_results, method = "Avg + Movie Bias", RMSE = movie_rmse)

movie_rmse
```

```
## [1] 0.9437429
```

An improvement to the previous model is notable, but not enough. It is necessary to take into account the other discoveries made in exploration.

4th Model: Adding the User Bias

As is evident in the exploration, some users tend to rate higher than other users. This can be explained by anything from personal preference to the current mood the user was in at the moment of rating. A user bias, or user effect, is added to the model.

```
#calculating user bias for each user
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#making predictions
predictions <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_rmse <- RMSE(test_set$rating, predictions)
rmse_results <- add_row(rmse_results, method = "Avg + Movie Bias + User Bias", RMSE = user_rmse)
user_rmse
```

```
## [1] 0.8659319
```

This model lowers the deviation down to 0.86593, which is not enough. Futher improvement is needed.

5th model: Adding the Year Bias

The release year of the movie has an impact on the rating and should be added to the model as the target RMSE has not been reached.

```
#extract year from title
train_set <- train_set %>%
  mutate(year = str_extract(train_set$title, "\\((\\d{4}\\))"))
train_set$year <- as.numeric(gsub("\\(|\\)", "", train_set$year))

test_set <- test_set %>%
  mutate(year = str_extract(test_set$title, "\\((\\d{4}\\))"))
test_set$year <- as.numeric(gsub("\\(|\\)", "", test_set$year))
#calculate the year bias
year_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))
#make predictions
predictions <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by = "year") %>%
```

```
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)
year_rmse <- RMSE(test_set$rating, predictions)
rmse_results <- add_row(rmse_results, method = "Avg + Movie Bias + User Bias + Year Bias", RMSE = year_
year_rmse
```

```
## [1] 0.8656117
```

6th Model: Adding the Genre Bias

Genre bias was evident in the exploration and is further explored. This is explained by some people having a preference towards comedies and other dramas etc.

```
#calculating genre bias
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "year") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_y))
#making predictions
predictions <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by = "year") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  pull(pred)
genre_rmse <- RMSE(test_set$rating, predictions)
rmse_results <- add_row(rmse_results, method = "Avg + Movie Bias + User Bias + Year Bias + Genre Bias",
genre_rmse
```

```
## [1] 0.8653661
```

This has improved the RMSE, but target has not been reached.

Regularization

Regularization is the technique used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting and underfitting. For example, if movie has a rating of 5, but only by one person, then the bias will only be single rating's deviation from the average. This will be equal to a bias, which has been rated by 100 people. This can lead to overfitting or underfitting. Regularization solves this problem by adding weights to all the biases.

A parameter for the weights is found through cross-validation

```
#finding optimal lambda
lambdas <- seq(0, 10, 0.25)

rmses <- 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))

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

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

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    pull(pred)

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

reg_rmse <- rmses[which.min(rmses)]
rmse_results <- add_row(rmse_results, method = "Regularized Model", RMSE = reg_rmse)
reg_rmse
```

```
## [1] 0.8647544
```

```
lambdas[which.min(rmses)]
```

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

The target is reched with regularization and tuning parameter of 4.5.

Validation

Validation set is used to test the performance of the best fitting model.

```r
#defining the average
mu <- mean(train_set$rating)

#defining lambda for regulization
l <- lambdas[which.min(rmses)]

#calculating regularized movie bias
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))

#calculating regularized user bias
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))

#defining year bias
#adding column year to validation set
validation <- validation %>%
  mutate(year = str_extract(validation$title, "\\((\\d{4}\\))"))
validation$year <- as.numeric(gsub("\\(|\\)", "", validation$year))

#calculating the regularized year bias
year_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - b_i - mu - b_u)/(n()+l))

#calculating regularized genre bias
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "year") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - mu - b_u - b_y)/(n()+l))

#calculating the prediction
predicted_ratings <-
  validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "year") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  pull(pred)

#calculating rmse
RMSE(validation$rating, predicted_ratings)
```

```
## [1] 0.8651557
```

Conclusion

The best performing model was developed using regularization on the model of biases.

```
as.data.frame(rmse_results)
```

```
##                                                 method      RMSE
## 1                                          Proportions 1.4987471
## 2                                      Just the Average 1.0599043
## 3                                      Avg + Movie Bias 0.9437429
## 4                            Avg + Movie Bias + User Bias 0.8659319
## 5                Avg + Movie Bias + User Bias + Year Bias 0.8656117
## 6 Avg + Movie Bias + User Bias + Year Bias + Genre Bias 0.8653661
## 7                                      Regularized Model 0.8647544
```

The author finds the number of variables to be insufficient. For example, some people are more drawn to certain kind of actors while others are not. Also, the ratings from critics could help set the tone for the users, as some users tend to agree and others not. For future work, one could experiment with gradient boosted decision trees as was used in the Netflix Competition (https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf). Furthermore, matrix factorization could also be used, but this solution needs a considerably bigger computing power than was available to the author. On the other hand, one can also apply reservoir sampling, which would reduce the size of the data, but keep the statistical characteristics.

This assignment has been very interesting and beneficial for the author.