MovieLens

Victoria Famakinwa

2/12/2022

INTRODUCTION

This is the first HarvardX PH125.9x Data Science: Capstone course Project I will be working on. The purpose of this project is to create a recommendation system using the Movielens dataset which has been provided in the course. This is to demonstrate the depth of my understanding in the whole course series thus far. In this project the goal is to train a machine learning algorithm using the inputs in a subset to predict movie ratings in the validation set.

For this project the MovieLens Data set was provided by edx instructor in the given code below; Here the data is split into edx set and validation set which is 10%. noted: Validation set will only be used for final evaluation.

```
knitr::opts_chunk$set(echo = TRUE)
############################################################
# 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")
```

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

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
```

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

```
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
```

```
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
```

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

# 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")

# 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)`
```

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

```
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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

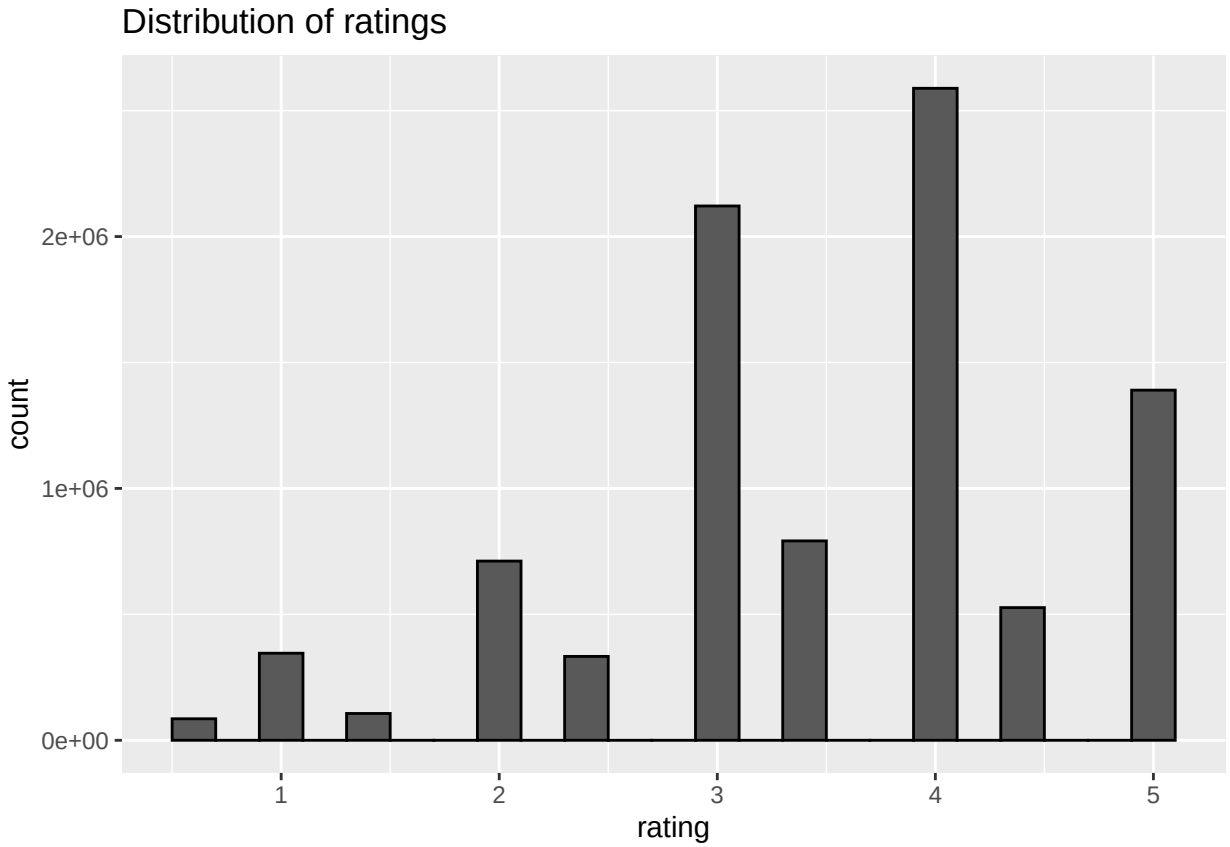
To fully comprehend the data set, we will first look at the structure of the data collection as well as the descriptive statistic.

EXPLORING THE DATA

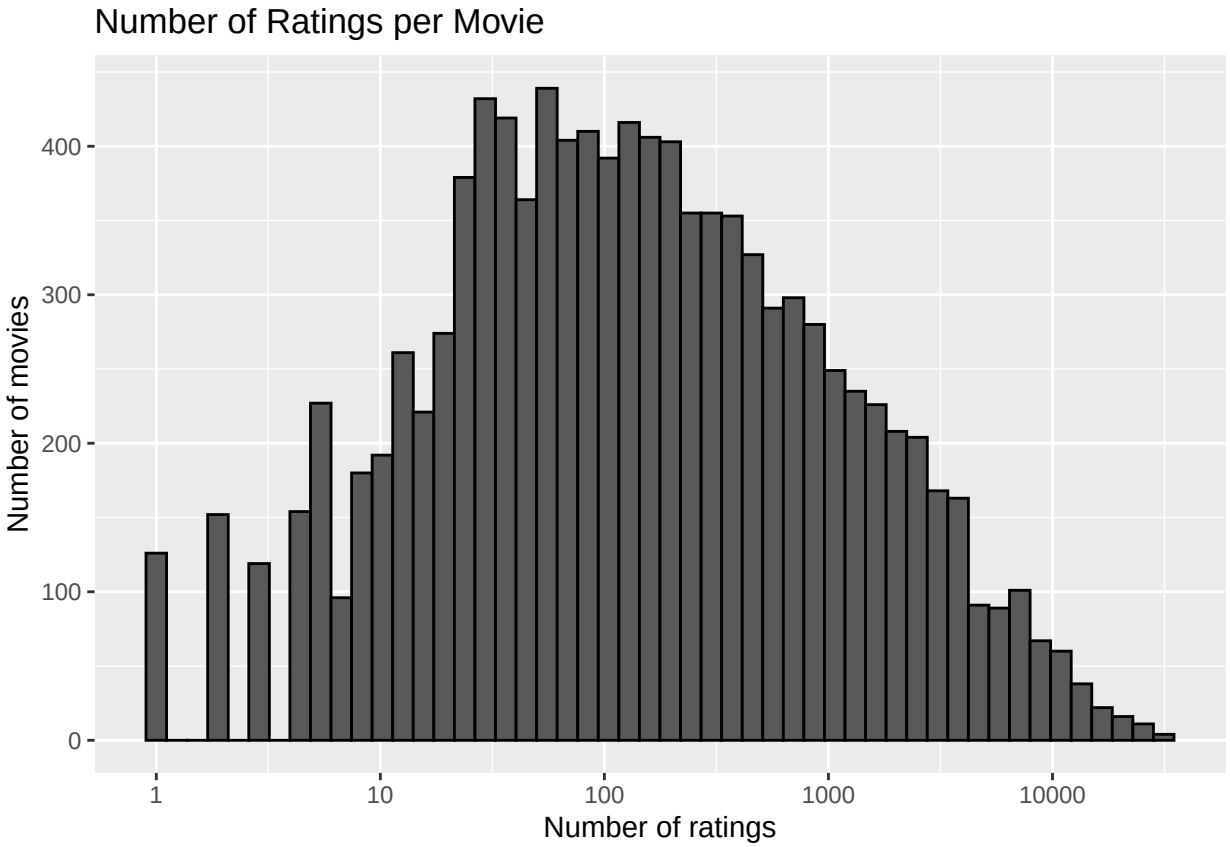
```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      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
```

```
##   Users Movies
## 1 69878  10677
```

The number of unique users is 69878, and the number of movies is 10677



A casual examination of the dataset reveals that the ratings are roughly regularly distributed, with integer values being significantly more prevalent as can be seen in the figure above.



Also as can be seen in the Figure above,there is a significant disparity between a few users who have evaluated a vast number of movies and the majority of users who have only reviewed a few.

Extracting "Year released" from Title and "Age at rating" from timestamp column

In order to convert timestamp to datetime format, there is a need to install the "Lubridate" library

```
## 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
```

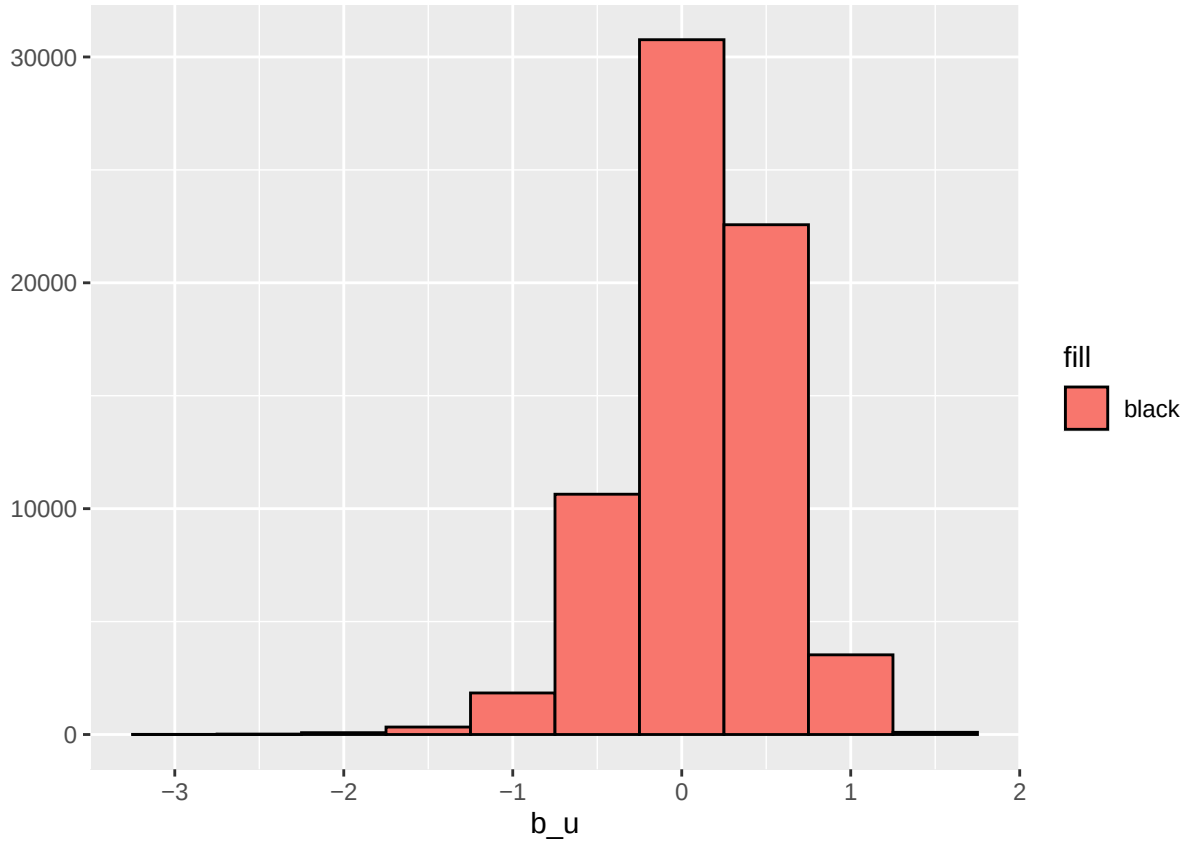
edx1 becomes the final dataset for test and train of the model after all extractions have been done. A year rated field is added to the dataset which was extracted from the timestamp column. It is later converted

into the age at rating field which indicates the difference between the Year the movie was released and the Year it was rated.

Every movie was released in a specific year, which is specified in the title. Every user assigned a rating to a film in a specific year, which is reflected in the timestamp data. The distinction between these two years shall be defined below.

Exploring User Effect on Ratings

The graph below shows the number of users vs. the average rating. Again, there is a significant user effect on rating, thus this needs to be accounted for in model creation. It's worth noting that it's a nearly normal distribution with a distinct shape than the movie effect as would be seen later on, implying that the movie and user effects have different dynamics.



Exploring Movie Effect on Ratings

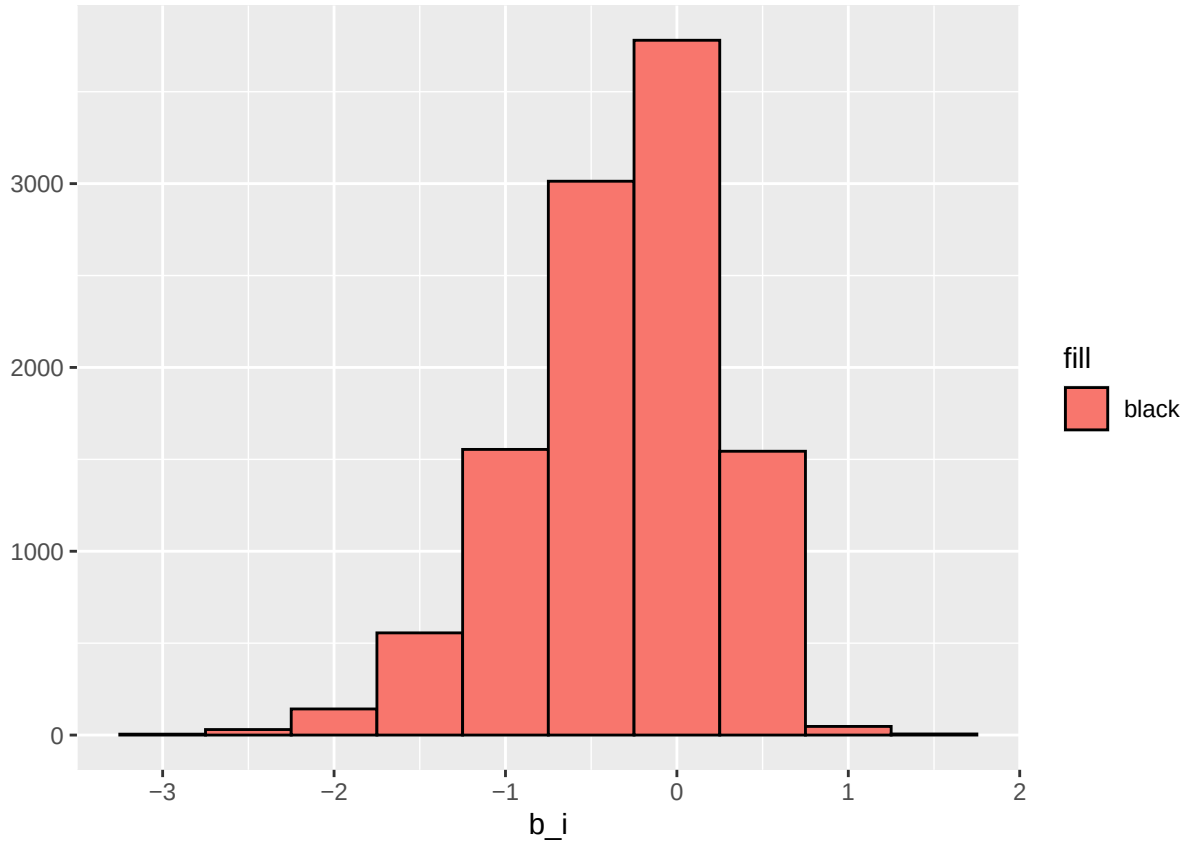
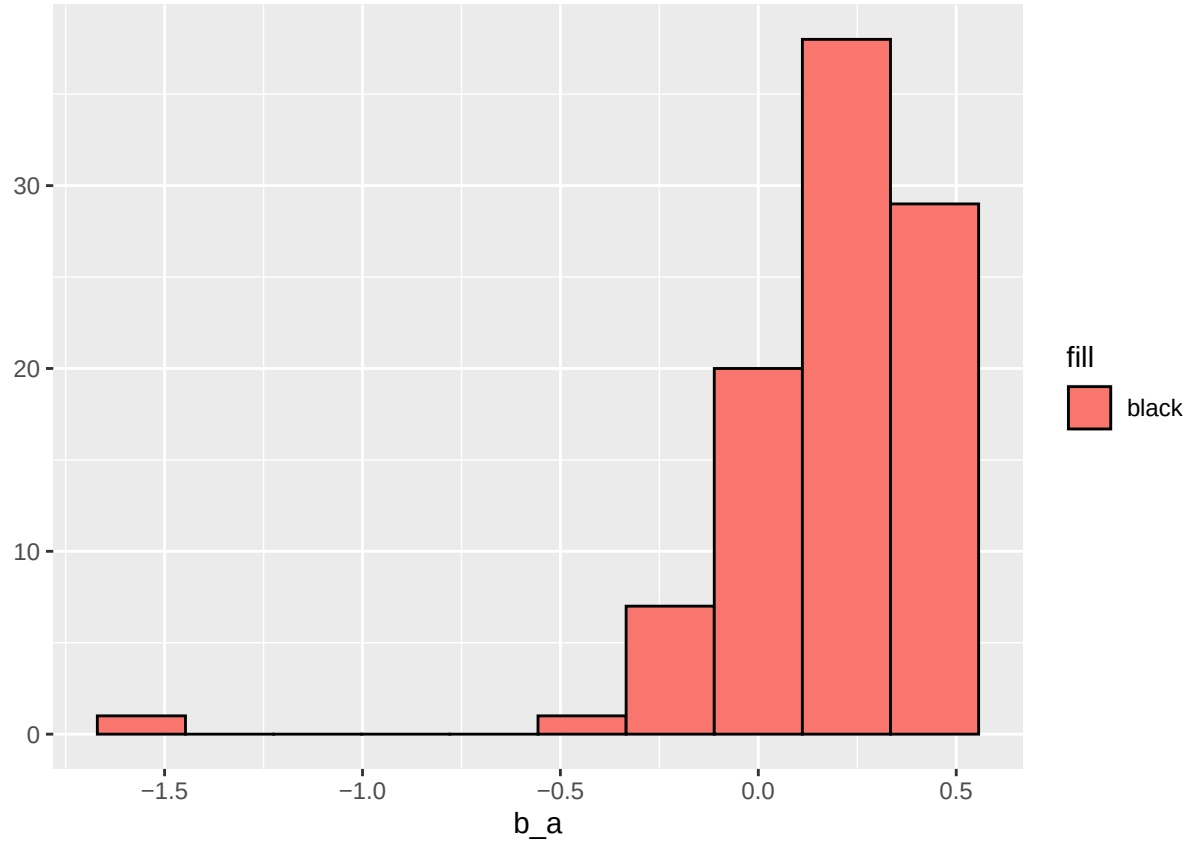


Figure above displays the mean rating distribution by movie. This shows a big movie effect on ratings, which is something I'll have to factor into my model. In comparison to a normal distribution, the distribution seems to be skewed to the left.

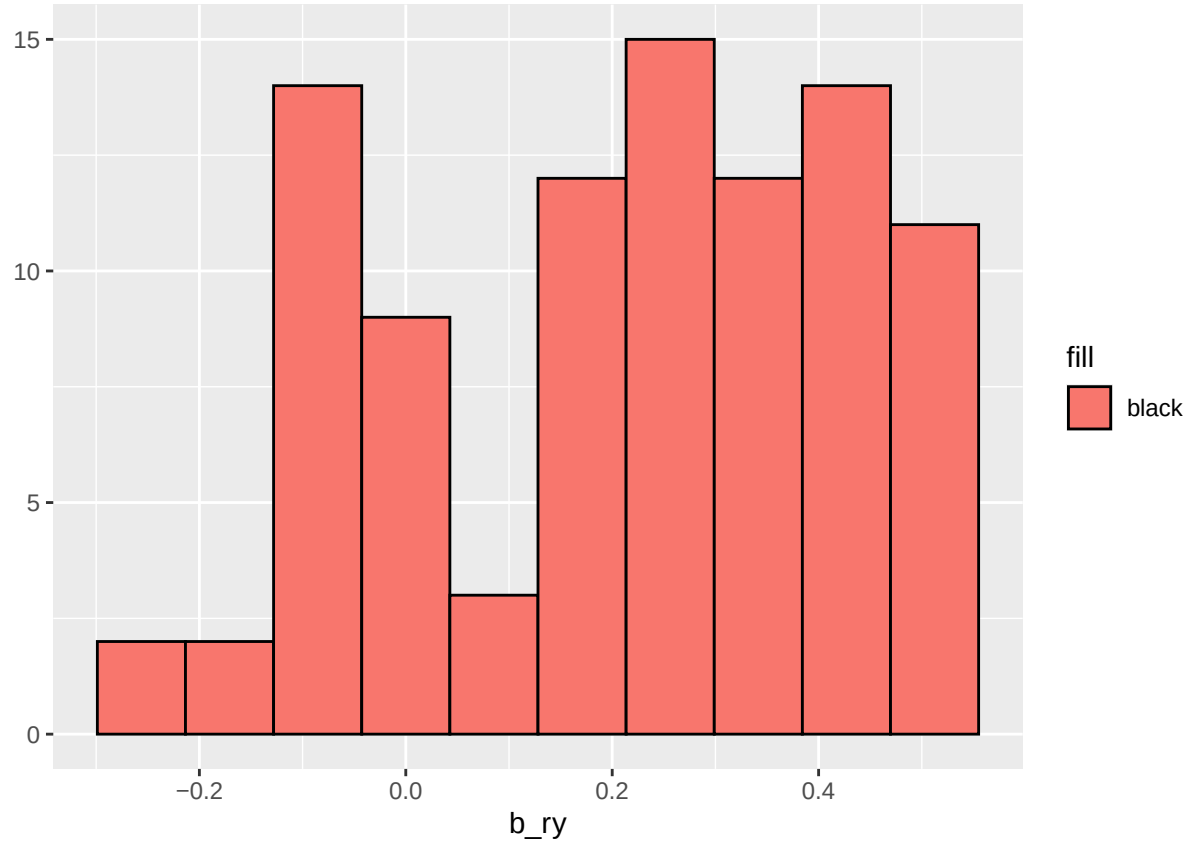
Exploring Age of Movie at Rating

We tried to see if adding a bias of age as at the time of rating (b_a) to the model could better predict ratings because we discovered that the age of movies at the time of rating seemed to affect ratings. Let's start by calculating the age bias and looking at its distribution as can be seen in the plot below. It can be observed that more rating was done in more recent years. This will definitely have an effect on the movie ratings because there is a clear bias in age at rating.



Exploring Movie Release Year on Ratings

Another element to consider is the year the movie was released. I took the year of release from the title and plotted the average rating through time. Ratings are affected by the year of release because if the movie was released in recent years more ratings would been done on it as compared to movies in early years with fewer ratings, therefore I'll include that as another model aspect.



Extracting the Genres

The exploration of genres can be aided by splitting the genre information into multiple rows. However, if we consider each row to be a single record, the preceding dataset transformation actually duplicated each record into many ones, depending on the combination of genres for each movie.We can actually visualize the genres of each movie by doing so. Each row, for example, represents a movie, whereas each column represents a genre. A unique blend of genres can be found in each film.

```
## [1] 20
```

STRATEGY FOR MODELLING

The age of the movie at rating appears to have an effect on the rating, whereas genres just add a little information but we'll use it anyhow. As a result, it would be taken into account that the effects of movie age on rating, movie effect, user effect, and movie release year as well as genres, when developing our model. A more robust model will also be built using regularization. I'll assess these models and select the best one.

DATA SPLIT

Here, the edx1 dataset will be split into train set (for training the model) and test set (for testing the model)

```
## Joining, by = c("userId", "movieId", "rating", "title", "year_released", "genres", "year_rated", "ag
```

Define RMSE: Residual Mean Squared Error

RMSE is used to evaluate the models as follows:

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

First Model: Using Mean Ratings

The overall average of all ratings will be the best forecast of ratings for each movie in the Average ratings model, simply based on the ratings themselves to minimize the RMSE.

```
mu <- mean(train$rating)
rmse_naivemean <- RMSE(test$rating, mu)
model_scores <- data.frame(method='Naive means',
rmse=rmse_naivemean)
```

The naive RMSE is 1.0611, and the average rating is mu = 3.51247.

Second Model: Adding b_i to show Movie effect on Ratings

We add the bias of movie (b_i) to the model since intrinsic aspects of a movie can plainly affect its ratings. This means that the average of the ratings for each movie will differ from the general average rating of all movies.

```
b_i <- train %>%
group_by(movieId) %>%
summarize(b_i = mean(rating - mu),
.groups='keep')

predicted_ratings_1 <- mu + test %>%
left_join(b_i, by='movieId') %>%
pull(b_i)
rmse_movie <- RMSE(test$rating, predicted_ratings_1)
model_scores <- rbind(model_scores, c("Movie effects", rmse_movie))
```

The model's RMSE is 0.9429 which shows some improvement on the model.

Third Model: Adding b_u to show User effect on Ratings

In a similar way to the movie effect, a user's intrinsic characteristics might influence a movie's rating. A stricter user, for example, might offer lower scores for all movies he or she has seen than other users have given.

```
b_u <- train %>%
left_join(b_i, by='movieId') %>%
group_by(userId) %>%
summarize(b_u = mean(rating - mu - b_i),
.groups='keep')

predicted_ratings_2 <- test %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
mutate(pred = mu + b_i + b_u) %>%
pull(pred)
rmse_user <- RMSE(test$rating, predicted_ratings_2)
model_scores <- rbind(model_scores, c("User effects", rmse_user))
```

The user bias (b_u) has now been added to the movie effect model.RMSE value is 0.8571 which shows a noticeable effect that the user has on the Ratings.

Fourth Model: Adding b_g to show Genres effect on Ratings

Let's now see if the genre type of a movie can affect it's rating, seeing that some users would prefer to rate some genres more that others.

```
b_g <- train %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
group_by(genres) %>%
summarize(b_g = mean(rating - mu - b_i - b_u),
.groups='keep')

predicted_ratings <- test %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
left_join(b_g, by='genres') %>%
mutate(pred = mu + b_i + b_u + b_g) %>%
pull(pred)
rmse_genres = RMSE(test$rating, predicted_ratings)
model_scores <- rbind(model_scores, c("Genres", rmse_genres))
```

As earlier suspected, Genres seems to have an effect on the model with RMSE value of 0.8567

Fifth Model: Adding b_ry to show Release Year effect on Ratings

Now it has to be established if the Release Year of a movie has an effect on the movie's rating.

```
b_ry <- train %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
left_join(b_g, by='genres') %>%
group_by(year_released) %>%
```

```
summarize(b_ry = mean(rating - mu - b_i - b_u - b_g),
.groups='keep')

predicted_ratings <- test %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
left_join(b_g, by='genres') %>%
left_join(b_ry, by='year_released') %>%
mutate(pred = mu + b_i + b_u + b_g + b_ry) %>%
pull(pred)
rmse_releaseyear = RMSE(test$rating, predicted_ratings)
model_scores <-rbind(model_scores, c("Release year", rmse_releaseyear))
```

RMSE value is 0.8565. Not so much of a differences in the previous value but its still a valued change in the model

Sixth Model: Adding b_a to show Age at rating effect on Ratings

We tried to see if adding a bias of age (b_a) to the model could better predict ratings because we discovered before that the age of movies at the time of rating seemed to affect ratings.

```
b_a <- train %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
left_join(b_g, by='genres') %>%
left_join(b_ry, by='year_released') %>%
group_by(age_at_rating) %>%
summarize(b_a = mean(rating - mu - b_i - b_u - b_g - b_ry),
.groups='keep')

predicted_ratings_3 <- test %>%
left_join(b_i, by='movieId') %>%
left_join(b_u, by='userId') %>%
left_join(b_g, by='genres') %>%
left_join(b_ry, by='year_released') %>%
left_join(b_a, by='age_at_rating') %>%
mutate(pred = mu + b_i + b_u + b_g + b_ry + b_a) %>%
pull(pred)
rmse_age_at_rating = RMSE(test$rating, predicted_ratings)
model_scores <-rbind(model_scores, c("Age_at_rating", rmse_age_at_rating))
```

RMSE is 0.8565. This does not seem to agree with the previous thought that age at rating has an effect on rating because compared to the previous RMSE value of 0.8565 there seem to be no difference.

REGULARIZATION

Now let's see if we can boost our model through regularization. Regularization is the process of determining a penalty term to limit effect size fluctuation. My penalty term is *lambda*, and I chose it by utilizing cross validation to minimize this function.

```
lambdas <- sort(c(seq(0,     0.1, 0.1),
                  seq(0.2,   0.6, 0.025),
                  seq(0.62,  0.8, 0.2),
                  seq(0.825, 4,   0.25)))
```

lambda is applied to function that calculates RMSE for each lambda.

```
rmses_regularization <- sapply(lambdas, function(lambda) {

  mu <- mean(train$rating)

  # Adding movie effects
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + lambda),
              .groups='keep')

  # Adding user effects
  b_u <- train %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n() + lambda),
              .groups='keep')

  # Adding genres effects
  b_g <- train %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambda),
              .groups='keep')

  # Adding year_released effects
  b_ry <- train %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    group_by(year_released) %>%
    summarize(b_ry = sum(rating - mu - b_i - b_u - b_g)/(n() + lambda),
              .groups='keep')

  # Adding age_at_rating effects
  b_a <- train %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    left_join(b_ry, by='year_released') %>%
    group_by(age_at_rating) %>%
    summarize(b_a = sum(rating - mu - b_i - b_u - b_g - b_ry)/(n() + lambda),
              .groups='keep')


  # Predicted rating
```

```
  predicted_ratings <- test %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    left_join(b_ry, by='year_released') %>%
    left_join(b_a, by='age_at_rating') %>%
    mutate(pred = mu + b_i + b_u + b_g + b_ry + b_a) %>%
    pull(pred)

  RMSE(test$rating, predicted_ratings)
})
```

Now we find the minimum value for lambda

```
lambda <- lambdas[which.min(rmses_regularization)]
```

TRAINING WHOLE DATASET

Training has to be done on the whole dataset. Training edx1 data set using all the existing features developed using test and train

Mean rating

```
mu <- mean(edx1$rating)
```

Adding movie effects

```
b_i <- edx1 %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lambda),
            .groups='keep')
```

Adding user effects

```
b_u <- edx1 %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + lambda),
            .groups='keep')
```

Adding genres effects

```
b_g <- edx1 %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambda),
            .groups='keep')
```

Adding year_released effects

```
b_ry <- edx1 %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  group_by(year_released) %>%
  summarize(b_ry = sum(rating - mu - b_i - b_u - b_g)/(n() + lambda),
            .groups='keep')
```

Adding age_at_rating effects

```
b_a <- edx1 %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  left_join(b_ry, by='year_released') %>%
  group_by(age_at_rating) %>%
  summarize(b_a = sum(rating - mu - b_i - b_u - b_g - b_ry)/(n() + lambda),
            .groups='keep')
```

Predicting the model

To enable us make predictions on the validation dataset, we need to make the same timestamp and title column extractions. Therefor our validation becomes validation1 dataset which we will be using for our final prediction.

```
validation1 <- validation %>%
  mutate(year_rated = year(as_datetime(timestamp)))%>%
  mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","year_released"),"__") %>%
  select(-timestamp) %>%
  mutate(age_at_rating= as.numeric(year_rated)-as.numeric(year_released))
```

PREDICTED RATING

```
predicted_ratings <- validation1 %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  left_join(b_ry, by='year_released') %>%
  left_join(b_a, by='age_at_rating') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_ry + b_a) %>%
  pull(pred)


rmse_target <- 0.86490
```

```
final_rmse <- RMSE(validation1$rating, predicted_ratings)
print(paste("Final RMSE is", final_rmse), quote = FALSE)
```

```
## [1] Final RMSE is 0.86436004106205
```

CONCLUSION

The goal of MovieLens project was to create a model that had an RMSE of less than **0.86490**, this was achieved with a combination of features in the model (Movie effect + User effect + Release year effect + Genres effect + Age at rating effect) which gave an RMSE value of **0.86436**.

This demonstrates that by using a Linear Regression model with regularized effects on users and movies also considering other features like Release year, genres and age at rating, you can create a good recommender system that can predict ratings.