MovieLens Project Submission in R

Farouk Ibrahim

March 30, 2024

Contents

1 Introduction

Welcome to my first movie recommendation system! This report is one of the project submission requirements for the capstone course of the HarvardX Data Science series[1]. The R markdown file for this report and the PDF are available on GitHub[2].

Recommendation systems are used extensively and is crucial in online streaming shopping services like Netflix, YouTube, Amazon, etc. These system provides the right recommendation for the customer based on his preference and similar customer profiles. These system has a dramatic economic impact on how online business are being developed and business are trying their best to improve their algorithm, for example Netflix announced in 2006 that they would give a one million dollar award to whoever could make their suggestion system at least 10% better. This is an example of how important such system are for businesses.

A common way for recommendation systems to work is by using a rating scale from 1 to 5 grades or stars, where 1 means least satisfied and 5 means most satisfied. The primary objective of a recommendation system is to assist users in finding what they need based on their likes, previous actions by predicting the rating of a new item, which is movie in our case.

I have created my own recommendation system using all the tools I have seen throughout the courses in this series.

More about my profile can be found in this link [3].

2 Dataset

In this project, I have created a movie recommendation system using the MovieLens dataset [4]. I used the 10M version [5] of the MovieLens dataset to make the computation a little

[1] https://pll.harvard.edu/series/professional-certificate-data-science
[2] https://github.com/FaroukIbrahim/harvard
[3] https://www.linkedin.com/in/farouk-ibrahim-72394b7/
[4] https://grouplens.org/datasets/movielens/latest/
[5] http://grouplens.org/datasets/movielens/10m/

simpler. I downloaded the MovieLens data and run code provided by the capstone course to generate my datasets. This code is also available in the submitted R code.

I developed my machine learning algorithm using the "edx" dataset which is 90% of the 10M version which is further split for training "train_set" and validation "test_set". For a final test of my final algorithm, I predicted movie ratings in the "final_holdout_test" set which is the remaining 10% of the 10M version as if they were unknown.

3 Evaluation

Root Mean Squared Error (RMSE) metric function was used to evaluate the difference between predicted values and actual values. RMSE was used to evaluate how close my predictions are to the true values in the "final_holdout_test" set.

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

The goal of this project is to build a recommendation system with RMSE less than 0.8649.

4 Methods & Analysis

4.1 Data Preparation

In the following section, the code is provided by the course and it allows you to download the dataset and split it into two parts edx and final_handout_test.

edx dataset constitute 90% of the data and is used for training where as final_handout_test constitute of 10% and is used for validation.

```
##############################################
# Create edx and final_holdout_test sets
#----------------------------------------------
# In the following section, the code is provided by the course and it allows
# you to download the dataset and split it into two parts edx and
# final_handout_test.
# edx dataset constitute 90% of the data and is used for training where as
# final_handout_test constitute of 10% and is used for validation.
##############################################

# Note: this process could take a couple of minutes
```

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

4.2 Data Exploration

I want to first explore the data to understand its parameters using the below code.

```
##############################################
# Data Exploration
#--------------------------------------------
# I want to first explore the data to understand its parameters using the below code.
##############################################


# The dataset includes 6 columns useid, moveid, rating, timestamp, title, and genres.
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
# The total number of training & test dataset is 9M.
nrow(edx)
```

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

```r
#The total number of validation dataset is 1M.
nrow(final_holdout_test)
```

```
## [1] 999999
```

4.3 Data Splitting

In this section, we split the edx datatset to training 80% and test 20% sets.

Training dataset has 7.2M records whereas test dataset has 1.8M records.

To make sure we don't include users and movies in the test set that do not appear in the training set, we removed these using the semi_join function.

```r
################################################
# Training Data Splitting
#-----------------------------------------------
# In this section, we split the edx datatset to training 80% and test 20% sets
# Training dataset has 7.2M records whereas test dataset has 1.8M records
# To make sure we don't include users and movies in the test set that do not
# appear in the training set, we removed these using the semi_join function.
################################################

# First we will split the edx data set into training and test sets
# Loaded the caret package to be able split the data use "createDataPartition"
library(caret)

# Set the random seed in order to be able to repeat the results.
set.seed(123)

# Split the edx data 1 time into 80% for training and 20% for testing.
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# To make sure we don't include users and movies in the test set that do not
# appear in the training set, we removed these using the semi_join function.
test_set <- test_set %>%
```

```
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

```
# 7.2M records for train_set
nrow(train_set)
```

```
## [1] 7200043
```

```
# 1.8M records for test_set
nrow(test_set)
```

```
## [1] 1799965
```

4.4 Model Evaluation

Here we define the function for Root Mean Squared Error

```
##############################################
# Model Evaluation
#--------------------------------------------
# Here we define the function for Root Mean Squared Error
##############################################

# Here we define the function for Root Mean Squared Error.
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

4.5 Methods used on edx Dataset

My main reference from the series for building a recommendation system was a blog [6] for winning the Netflix prize from Section 6.2 Machine Learning course, which also provides us with a sample code we can build on top of.

The first method I used is the first Linear approach in the blog which is normalization of global Effects known as Bias. Some Bias like movie and user effect contributed to improving RMSE where as other didn't.

In the following sections, we will talk through them all.

[6] https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

4.5.1 Bias Linear Method: Overall Average

Lets start building the first most basic recommendation system using baseline rating which is the mean overall user-movie ratings.

This will be used as a naive prediction along with a corresponding RMSE value.

```
##############################################
# Bias Linear Method:  Overall Average
#---------------------------------------------
# Lets start building the first most basic recommendation system using
# baseline rating which is the mean overall user-movie ratings
# This will be used as a naive prediction along with a corresponding
# RMSE value
##############################################

# We simply calculate the mean of all the rating (0-5) in the train_set
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.512371
```

```
# the mean mu is 3.5 which is higher than the median 2.5, lets calculate its RMSE
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse
```

```
## [1] 1.060013
```

```
# the RMSE is 1.06, lets check the RMSE if we use the median of 2.5 (5/2).
# we created nrow of test set of value 2.5 and calculated RMSE

predictions <- rep(2.5, nrow(test_set))
median_rating<- RMSE(test_set$rating, predictions)
```

As you can see RMSE is 1.466 which is much higher than 1.06, hence lets log both RMSE in a table, along with a Delta column which will be used subsequently to compare the improvements we are making for the RMSE

```
# as you can see RMSE is 1.466 which is much higher than 1.06, hence lets log
# both RMSE in a table, along with a Delta column which will be used subsequently
# to compare the improvements we are making for the RMSE

rmse_results <- data_frame(
```

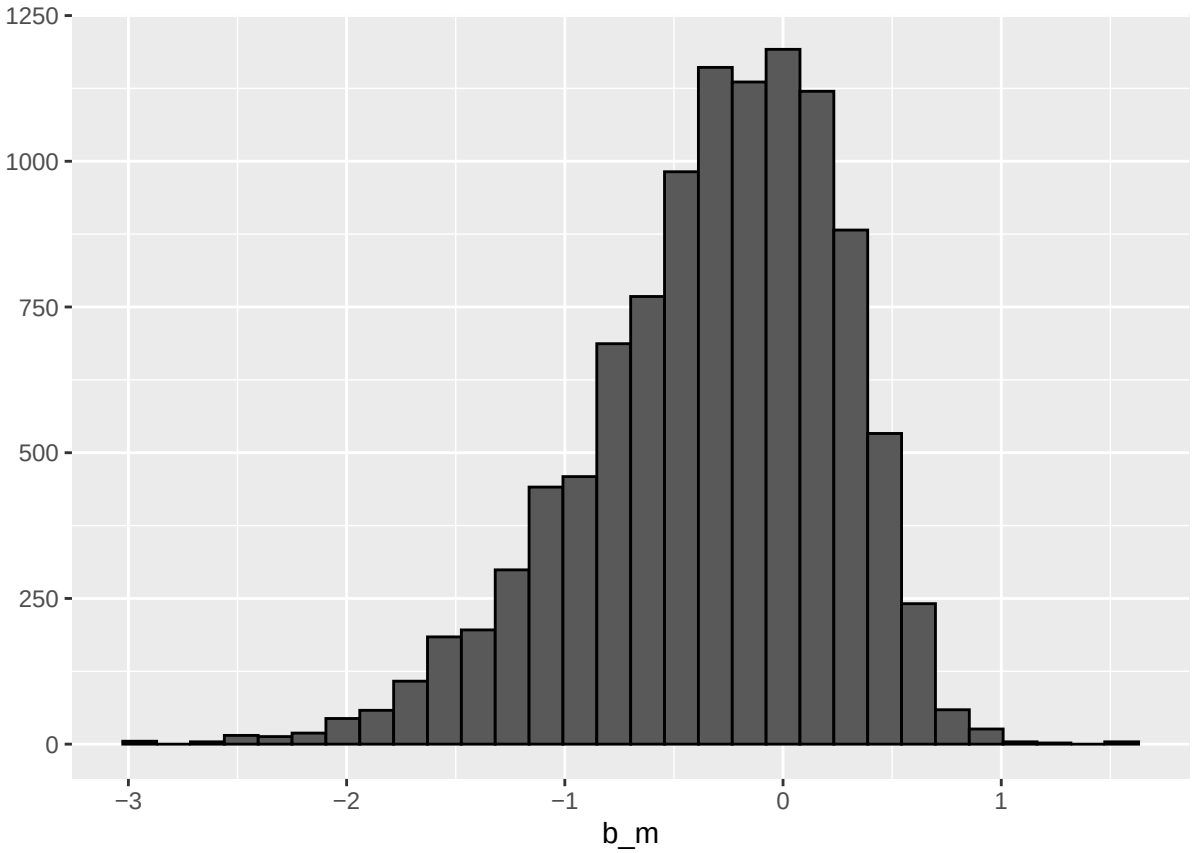
```
  Method_on_EDX = "Median Rating 2.5",
  RMSE = median_rating,
  Improvement = median_rating-median_rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Overall Average",
  RMSE = naive_rmse,
  Improvement = median_rating-naive_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.466115	0.0000000
Overall Average	1.060013	0.4061023

4.5.2 Bias Linear Method: Movie Effect

In this section, we are modeling movie specific effects; a particular movie might be more successful hence its ratings can be higher/lower than average.

```
##############################################
# Bias Linear Method:  Movie Effect
#--------------------------------------------
# In this section, we are modeling movie specific effects; a particular movie
# might be more successful hence its ratings can be higher/lower than average.
##############################################


# for each movieid in train_set using group_by function, calculate movie effect
# bias b_m using summarize. b_m is the mean difference between mu and rating.
# we then save the results in table movie_avgs
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu))

# to visualize this table we use a histogram with bin =30 , as you can see its
# normally distributed with most of the bias between -1 and +0.5
movie_avgs %>% qplot(b_m, geom ="histogram", bins = 30, data = ., color = I("black"))
```




```
# lets create the prediction, by simply adding the movie effect bias m_b to mu
# to do that we have to left join test_set with movie average
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_m)

# lets do a quick summary check to see how many NA, as you can see there are no
# due to the left join which is good.
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.217   3.586   3.512   3.878   4.750
```

```
# now for the interesting part, lets calculate the RMSE, save and print it
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie Effect",
  RMSE = model_1_rmse ,
  Improvement = naive_rmse - model_1_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927

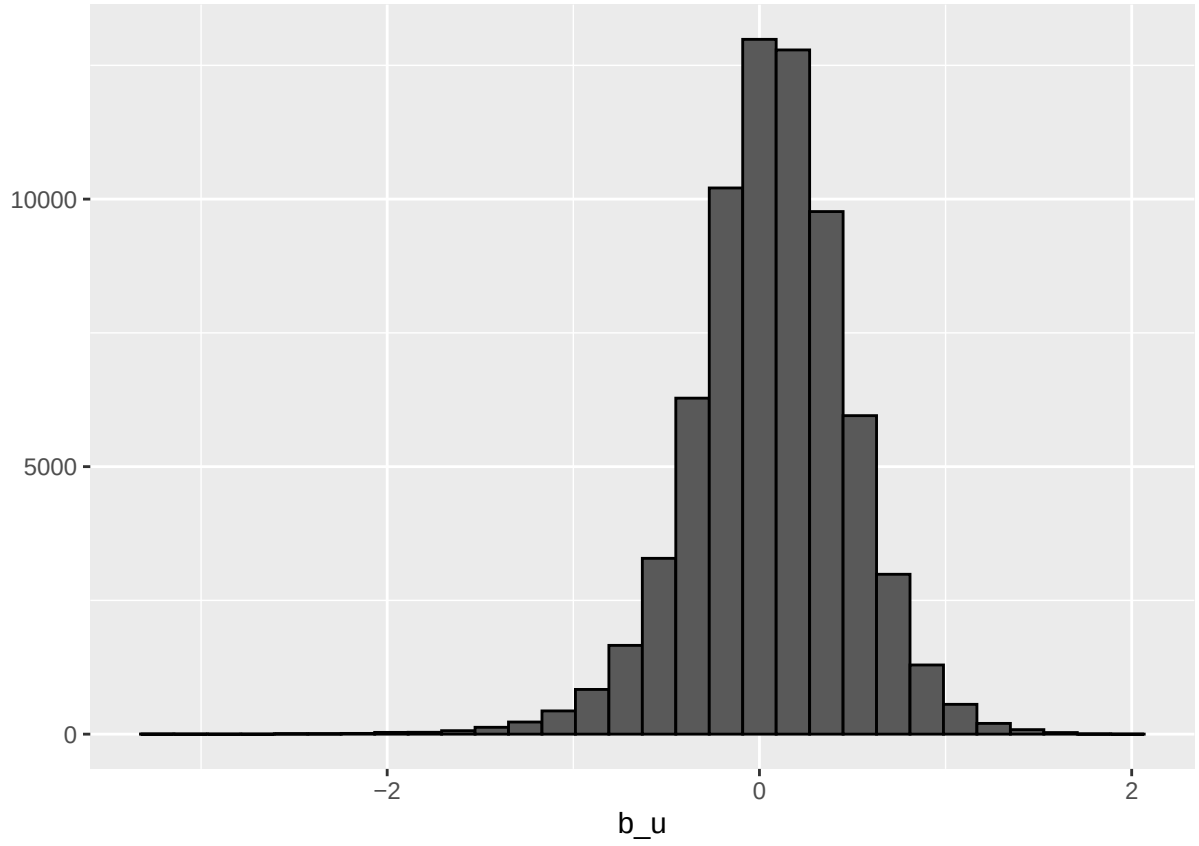
```
# as you can see RMSE reduced significantly by 0.1163 due to the movie effect
```

As you can see RMSE reduced significantly by 0.1163 due to the movie effect.

4.5.3 Bias Linear Method: User Effect

In this section we are modeling user specific effects, a particular user tends to rate movies lower or higher than the average user.

```
##############################################
# Bias Linear Method:  User Effect
#--------------------------------------------
# In this section we are modeling user specific effects, a particular user
# tends to rate movies lower or higher than the average user.
##############################################


# for each userid in train_set using group_by function, calculate user effect
# bias b_u using summarize. b_u is the mean of (rating - mu - b_m).
# we then save the results in table movie_avgs
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_m))


# to visualize this table we use a histogram with bin =30 , as you can see its
# normally distributed with most of the bias between -0.5 and +0.75
user_avgs %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```




```
# lets create the prediction, we use test_set to test our prediction and left
# join it with movie_avg and user_avg by movieid and userid respectively to
# be able to calculate pred as a sum of overall rating average, movie bias,
# user bias

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


# lets do a quick summary check to see how many NA, as you can see there are no
# due to the left join which is good.
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.7161  3.1353  3.5651  3.5127  3.9450  6.0547
```

```
# now for the interesting part, lets calculate the RMSE, save and print it
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie + User Effects",
  RMSE = model_2_rmse ,
  Improvement = model_1_rmse - model_2_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238

```
# As you can see RMSE reduced significantly by 0.077 due to the user effect.
```

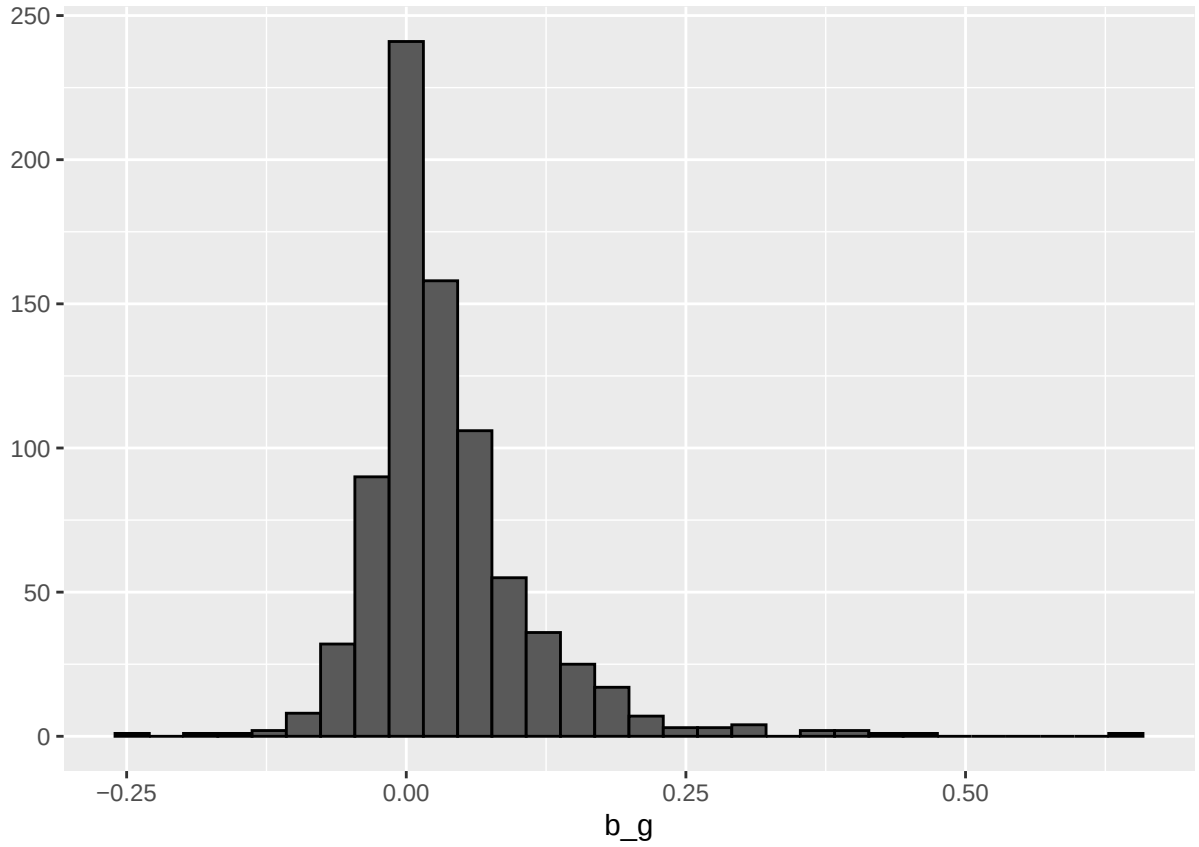
As you can see RMSE reduced significantly by 0.077 due to the user effect.

4.5.4 Bias Linear Method: Genres Effect

In this section, we are modeling genres effects, a particular type of movie or genres tends to be rated lower or higher than the average rating.

```
##############################################
# Bias Linear Method:  Genres Effect
#---------------------------------------------
# In this section, we are modeling genres effects, a particular type of movie
# or genres tends to be rated lower or higher than the average rating.
##############################################



# for each genres in train_set using group_by function, calculate genres effect
# bias b_g using summarize. b_g is the mean of (rating - mu - b_m - b_u).
# we then save the results in table movie_avgs

genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  filter(n()>=1) %>%
```

```r
  summarize(b_g = mean(rating - mu - b_m - b_u ))

# to visualize this table we use a histogram with bin =30 , as you can see its
# normally distributed with most of the bias between -0.1 and +0.2
genres_avgs %>% qplot(b_g, geom ="histogram", bins = 30, data = ., color = I("black"))
```



As you can see its normally distributed with most of the bias between -0.1 and +0.2

```r
# lets create the prediction, we use test_set to test our prediction and left
# join it with movie_avg, user_avg, genres_avg by movieid, userid, genres
# respectively to be able to calculate pred as a sum of overall rating average,
# movie bias, user bias, genres bias

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu + b_m + b_u + b_g) %>%
  pull(pred)
```

```r
# lets do a quick summary check to see how many NA, as you can see there are no
# due to the left join which is good.
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.7216  3.1322  3.5648  3.5126  3.9482  6.0745
```

```r
# the below code can be removed, however it is useful incase the left join
# has some NA, we can replace NA with average of predicted rating
#mean_predicted_rating <- mean(predicted_ratings, na.rm = TRUE)
#predicted_ratings[is.na(predicted_ratings)] <- mean_predicted_rating


# now for the interesting part, lets calculate the RMSE, save and print it
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie + User + Genres Effects",
  RMSE = model_3_rmse ,
  Improvement = model_2_rmse - model_3_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678

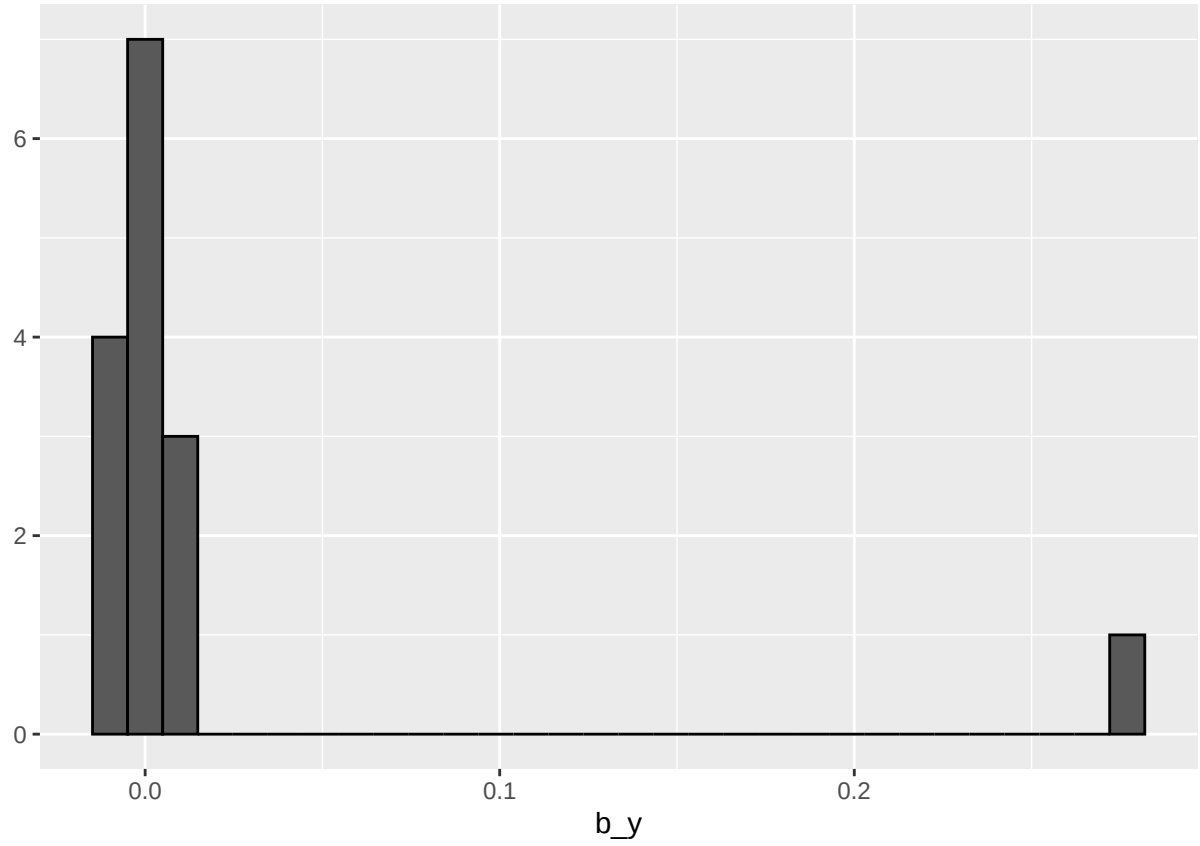
```r
# As you can see RMSE reduced slightly by 0.000367.
```

As you can see RMSE reduced slightly by 0.000367.

4.5.5 Bias Linear Method: Year Effect

In this section We would like to model year effects, we are looking for correlating that the year derived from time stamp might have an effect in reducing RMSE.

```
##############################################
# Bias Linear Method:  Year Effect
#---------------------------------------------
# In this section We would like to model year effects, we are looking for correlating
# that the year derived from time stamp might have an effect in reducing RMSE.
##############################################


# Install the following to allow date functions
if (!require("lubridate")) install.packages("lubridate")
library(lubridate)
library(dplyr)

# Use as_date_time function to convert to a date
train_set$date  <- as_datetime(train_set$timestamp, origin = "1970-01-01")

# Extract year and save it in the training and test set
train_set$year <- year(train_set$date)
test_set$date  <- as_datetime(test_set$timestamp, origin = "1970-01-01")
test_set$year <- year(test_set$date)


# for each year in train_set using group_by function, calculate year effect
# bias b_y using summarize. b_y is the mean of (rating - mu - b_m - b_u - b_g).
# we then save the results in table year_avgs
year_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_m - b_u - b_g))

# to visualize this table we use a histogram with bin =30 , we can see very
# very little effect +- 0.01
year_avgs %>% qplot(b_y, geom ="histogram", bins = 30, data = ., color = I("black"))
```



We can see very very little effect +- 0.01

```
# lets create the prediction, we use test_set to test our prediction and left
# join it with movie_avg, user_avg, genres_avg, year_avg by movieid, userid,
# genres, year respectively to be able to calculate pred as a sum of
# overall rating average, movie bias, user bias, genres bias, year bias
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
  pull(pred)

# lets do a quick summary check to see how many NA, as you can see there are no
# due to the left join which is good.
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.7195  3.1321  3.5650  3.5126  3.9482  6.0879
```

```r
# now for the interesting part, lets calculate the RMSE, save and print it
model_4_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie + User + Genres + Year Effects",
  RMSE = model_4_rmse ,
  Improvement = model_3_rmse - model_4_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161

```r
# As expected RMSE reduced very slightly.
```

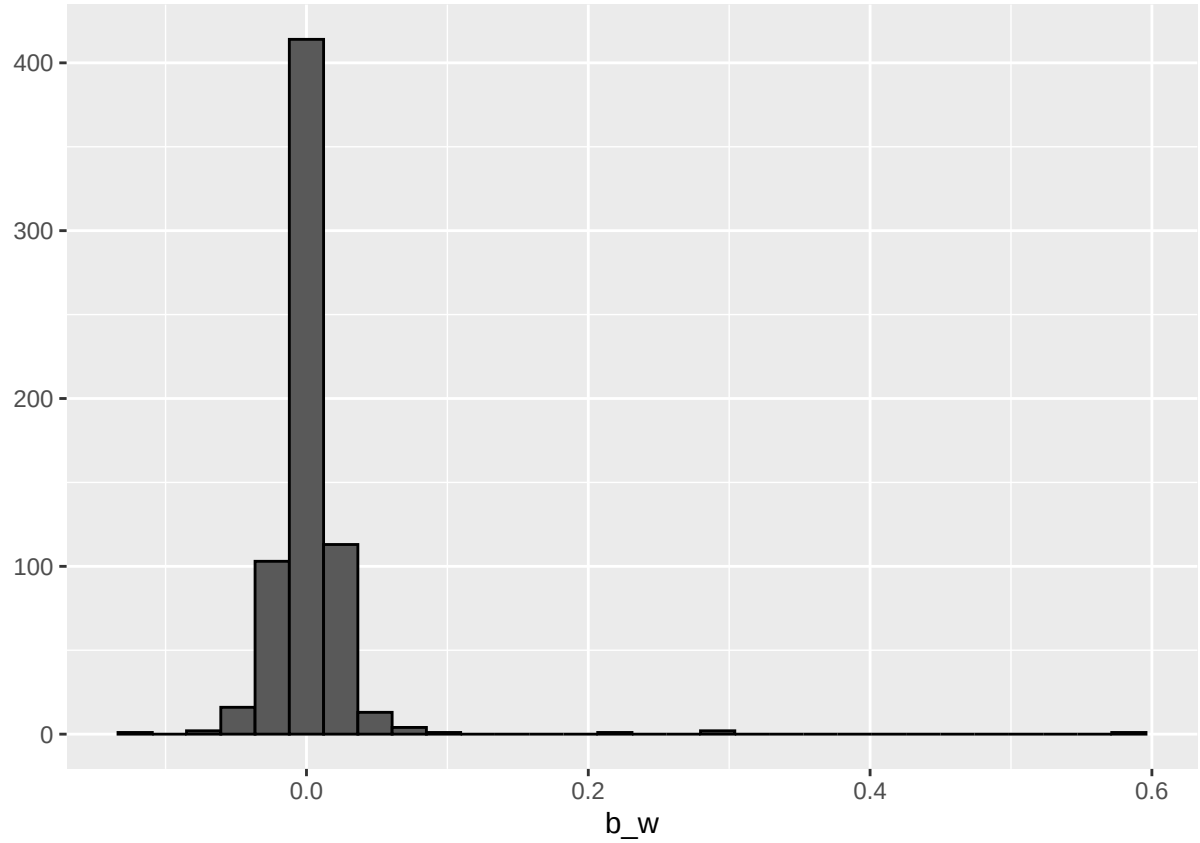
As expected RMSE reduced very slightly.

4.5.6 Bias Linear Method: Week Effect

In this section We would like to model week effects, we are looking for correlating that the week derived from time stamp might have an effect in reducing RMSE.

```r
##############################################
# Bias Linear Method:  Week Effect
#---------------------------------------------
# In this section We would like to model week effects, we are looking for correlating
# that the week derived from time stamp might have an effect in reducing RMSE.
##############################################



# for the train_set derive the week from timestamp/date
# 1. Calculate day of the week
train_set$day_of_week <- weekdays(train_set$date)

# 2. Round date to the nearest week
train_set$week <- round_date(train_set$date, unit = "week")
```

```r
# for the test_set derive the week from timestamp/date
# 1. Calculate day of the week
test_set$day_of_week <- weekdays(test_set$date)

# 2. Round date to the nearest week
test_set$week <- round_date(test_set$date, unit = "week")



# To make sure we don't include records in the test set that do not
# appear in the training set, we removed these using the semi_join function by
# week and day_week.This is optional
test_set <- test_set %>%
  semi_join(train_set, by = "week") %>%
  semi_join(train_set, by = "day_of_week")

# for each week in train_set using group_by function, calculate week effect
# bias b_w using summarize. b_w is the mean of (rating - mu - b_m - b_u - b_g -b_y).
# we then save the results in table week_avgs
week_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
    group_by(week) %>%
  summarize(b_w = mean(rating - mu - b_m - b_u - b_g - b_y  ))

# to visualize this table we use a histogram with bin =30 , we can see very
# very little effect +- 0.01
week_avgs %>% qplot(b_w, geom ="histogram", bins = 30, data = ., color = I("black"))
```



We can see very little effect +- 0.01

```
# lets create the prediction, we use test_set to test our prediction and left
# join it with movie_avg, user_avg, genres_avg, year_avg, week_avg by movieid,
# userid, genres, year, week respectively to be able to calculate pred as a sum
# of overall rating average, movie bias, user bias, genres bias, year bias, week
# bias
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y  + b_w) %>%
  pull(pred)

# lets do a quick summary check to see how many NA, as you can see there are no
# due to the left join which is good.
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -0.718   3.132   3.565   3.513   3.948   6.094
```

```
# now for the interesting part, lets calculate the RMSE, save and print it
model_5_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie + User + Genres + Year + Week Effects",
  RMSE = model_5_rmse ,
  Improvement = model_4_rmse - model_5_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161
Movie + User + Genres + Year + Week Effects	0.8658375	0.0000753

```
# Surprisingly RMSE for week reduced more than year.
```

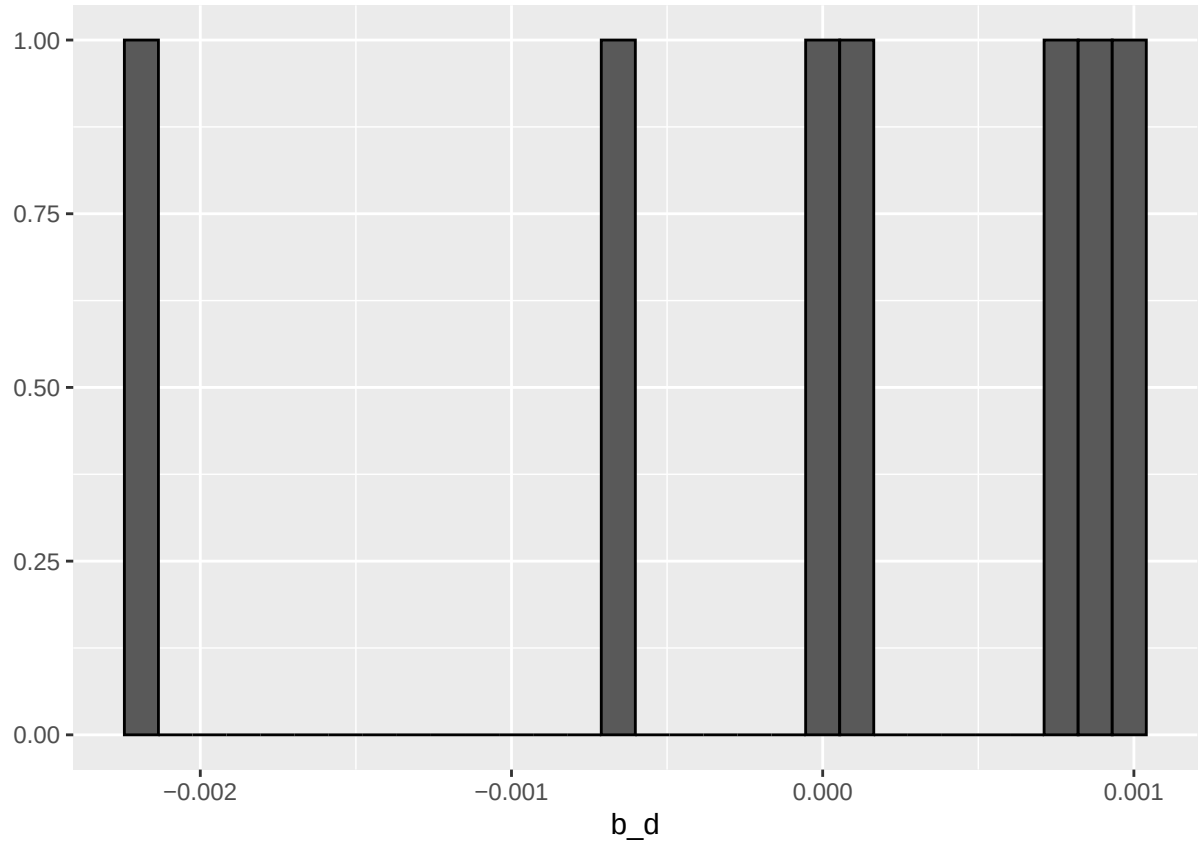
Surprisingly RMSE for week reduced more than year.

4.5.7 Bias Linear Method: Day Effect

In this section We would like to model day of week effects, we are looking for correlating that day_of week derived from time stamp might have an effect in reducing RMSE.

```
################################################
# Bias Linear Method:  Day Effect
#-----------------------------------------------
# In this section We would like to model day of week effects, we are looking for corre
# that day_of week derived from time stamp might have an effect in reducing RMSE
################################################



# for each day_of_week in train_set using group_by function, calculate its effect
# bias b_d using summarize. b_d is the mean of (rating - mu - b_m - b_u - b_g
# -b_y - b_w) we then save the results in table day_avgs
day_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  group_by(day_of_week) %>%
  summarize(b_d = mean(rating - mu - b_m - b_u - b_g - b_y - b_w))

# to visualize this table we use a histogram with bin =30 , we can see very
# very little effect +- 0.001
day_avgs %>% qplot(b_d, geom ="histogram", bins = 30, data = ., color = I("black"))
```



We can see very little effect +- 0.001

```
# lets create the prediction, we use test_set to test our prediction and left
# join it with movie_avg, user_avg, genres_avg, year_avg, week_avg, day_avg
# by movieid, userid, genres, year, week, day respectively to be able to
# calculate pred as a sum of overall rating average, movie bias, user bias,
# genres bias, year bias, week bias, day bias
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
```

```
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  left_join(day_avgs, by='day_of_week') %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y + b_w + b_d) %>%
  pull(pred)

# lets do a quick summary check to see how many NA, as you can see there are no
# due to the left join which is good.
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -0.718   3.132   3.565   3.513   3.948   6.092
```

```
# now for the interesting part, lets calculate the RMSE, save and print it
model_6_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie + User + Genres + Year + Week + Day Effects",
  RMSE = model_6_rmse,
  Improvement = model_5_rmse - model_6_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161
Movie + User + Genres + Year + Week Effects	0.8658375	0.0000753
Movie + User + Genres + Year + Week + Day Effects	0.8658364	0.0000010

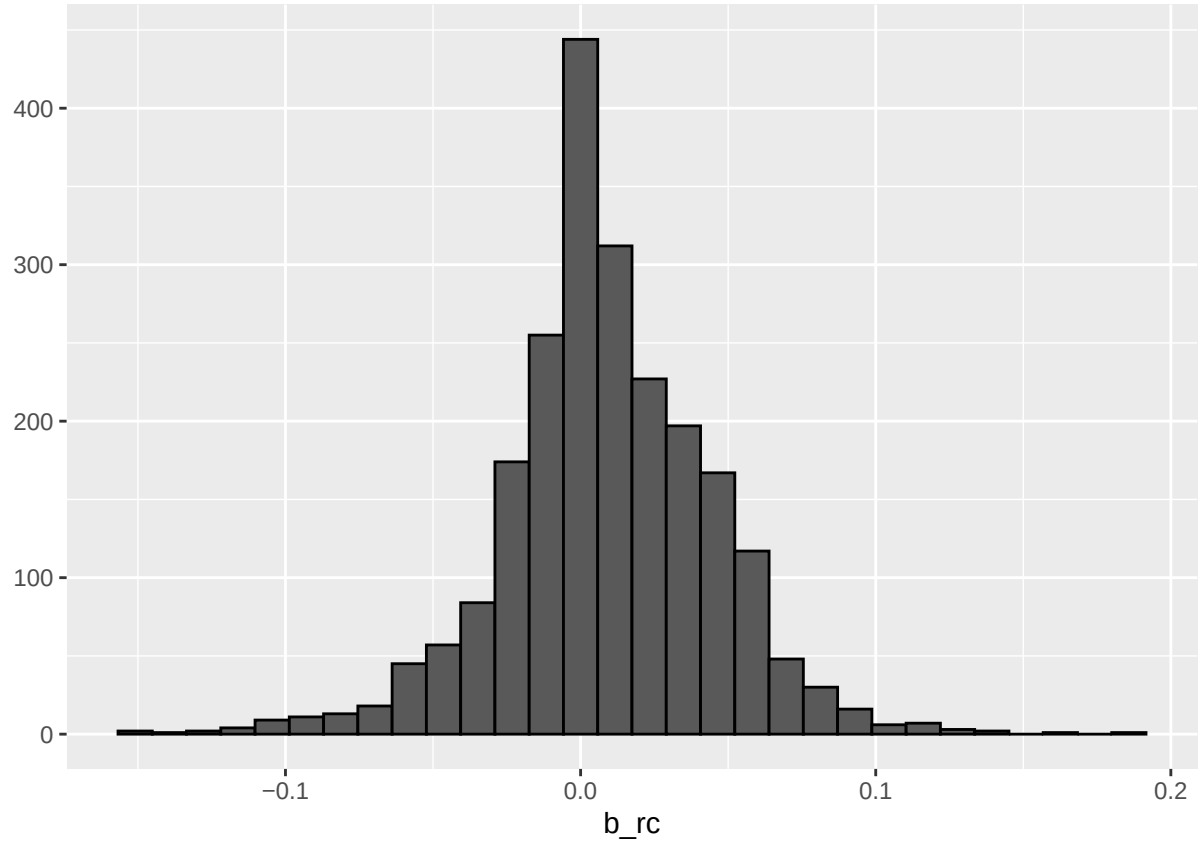
```
# As expected RMSE reduced very slightly.
```

As expected RMSE reduced very slightly.

4.5.8 Bias Linear Method: Rating Effect

In this section, we would like to model rating count per movie effects as it might have an effect in reducing RMSE. However as it turns out that it does not and hence will no be used subsequent models.

```
##############################################
# Bias Linear Method: Rating Effect
#--------------------------------------------
# In this section, we would like to model rating count per movie effects as it might h
# an effect in reducing RMSE. However as it turns out that it does not and hence
# will no be used subsequent models.
##############################################



# in training and test set, create rating_count per movie
test_set <- test_set %>% group_by(movieId) %>% mutate( rating_count = n())
train_set <- train_set %>% group_by(movieId) %>% mutate( rating_count = n())

# for each rating_count in train_set using group_by function, calculate its effect
# bias b_rc using summarize. b_rc is the mean of (rating - mu - b_m - b_u - b_g
# -b_y - b_w -b_d) we then save the results in table day_avgs
rating_count_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  left_join(day_avgs, by='day_of_week') %>%
  group_by(rating_count) %>%
  summarize(b_rc = mean(rating - mu - b_m - b_u - b_g - b_y - b_w - b_d ))

# to visualize this table we use a histogram with bin =30 , we can see very
# very little effect +- 0.05
rating_count_avgs %>% qplot(b_rc, geom ="histogram", bins = 30, data = ., color = I("bla
```



We can see very little effect +- 0.05

```
# lets create the prediction, we use test_set to test our prediction and left
# join it with movie_avg, user_avg, genres_avg, year_avg, week_avg, day_avg,
# rating_count_avg by movieid, userid, genres, year, week, day, rating_count
# respectively to be able to calculate pred as a sum of overall rating average,
# movie bias, user bias, genres bias, year bias, week bias, day bias, rating bias
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  left_join(day_avgs, by='day_of_week') %>%
  left_join(rating_count_avgs, by='rating_count') %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y + b_w + b_d + b_rc) %>%
  pull(pred)

# lets do a quick summary check to see how many NA, but we see many NAs
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
```

```
##    -0.7     3.1     3.5     3.4     3.9     6.1  590450
```

```
# lets replace them with the mean
mean_predicted_rating <- mean(predicted_ratings, na.rm = TRUE)
predicted_ratings[is.na(predicted_ratings)] <- mean_predicted_rating
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6556  3.2881  3.4390  3.4390  3.6872  6.1033
```

```
model_7_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Movie + User + Genres + Year + Week + Day + Rating Effects "
  RMSE = model_7_rmse,
  Improvement = model_6_rmse - model_7_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161
Movie + User + Genres + Year + Week Effects	0.8658375	0.0000753
Movie + User + Genres + Year + Week + Day Effects	0.8658364	0.0000010
Movie + User + Genres + Year + Week + Day + Rating Effects	0.9298308	-0.0639944

```
# RMSE got worse, so we will be ignoring this effect going forward.
```

RMSE got worse, so we will be ignoring this effect going forward. I did the same for (square root of the) number of days since user first rating as per below and the results didn't improve, hence we also disregarded it. test_set <- test_set %>% group_by(userId) %>% mutate(user_sqrt_fdate = sqrt(max(timestamp)-min(timestamp))) Going forward I have done my best to use bias to reduce the RMSE and will not use the rating effect.

4.5.9 Regularization + Linear Method

Regularization is applied for all the Linear bias methods above to prevent overfitting on the data set, in attempt to improve RMSE. The idea is to find an optimal lambda that will reduce RMSE.

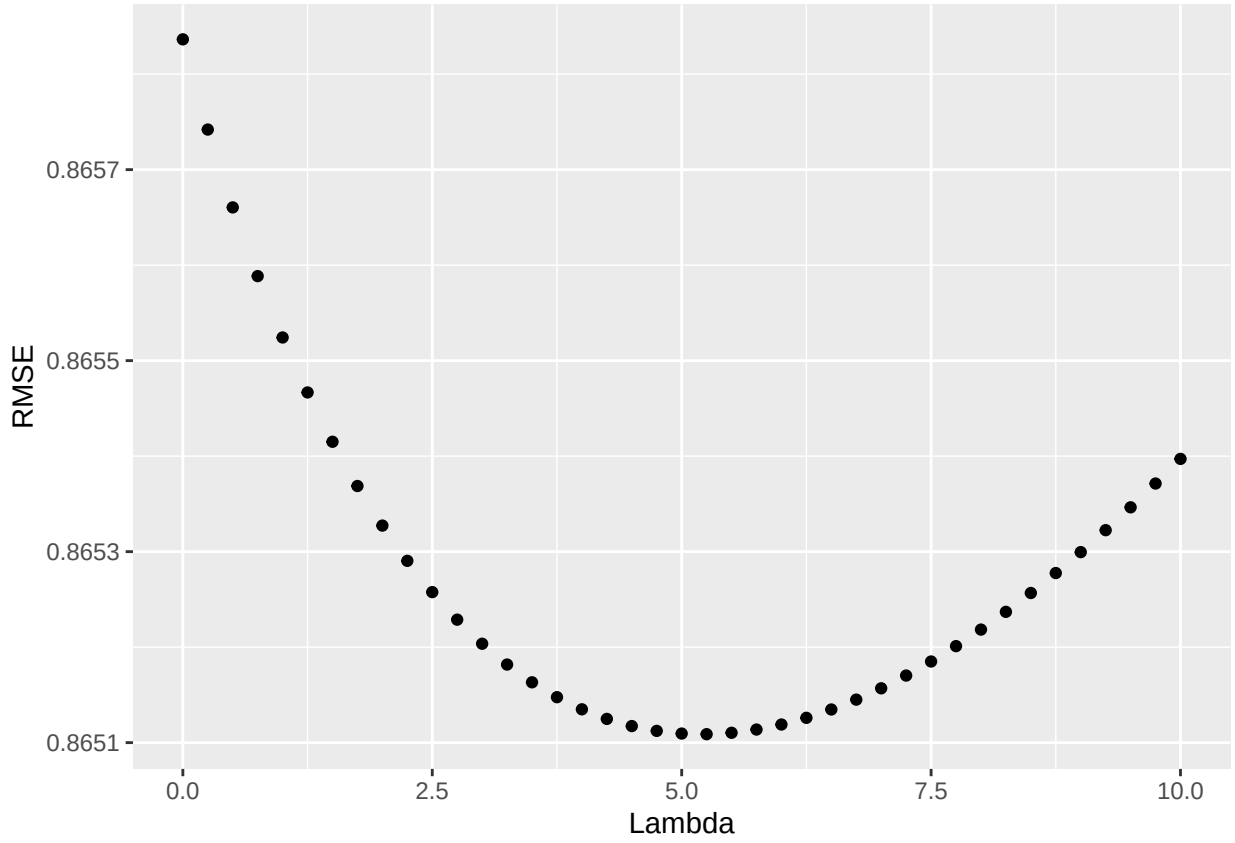
In regularization, instead of relying on mean to calculate the bias, we calculate the mean ourselves using sum and dividing the number of records however the number of records is adjusted by an optimal lambda parameter in attempt to optimize RMSE as you will see below.

```
##################################################
# Regularization + Linear Method
#-------------------------------------------
# Regularization is applied for all the Linear bias methods above to prevent
# overfitting on the data set, in attempt to improve RMSE.
# The idea is to find an optimal lambda that will reduce RMSE.
# In regularization, instead of relying on mean to calculate the bias, we
# calculate the mean ourselves using sum and dividing the number of records
# however the number of records is adjusted by an optimal lambda parameter
# in attempt to optimize RMSE as you will see below.
##################################################


# First Define a regularization function that takes lambda and the training and
# test set as an input which then runs through all the bias models and returns
# an RSME

regularization <- function(lambda, trainset, testset){

  # Mean (mu)
  mu <- mean(train_set$rating)

  # Movie effect (b_m)
  movie_avgs <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+lambda))

  # User effect (b_u)
  user_avgs <- train_set %>%
    left_join(movie_avgs, by="movieId") %>%
        group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu)/(n()+lambda))

  # genres effect (b_g)
```

```r
genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_m - b_u)/(n()+lambda))

# year effect (b_y)
year_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_m - b_u - b_g)/(n()+lambda))

# week effect (b_w)
week_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  group_by(week) %>%
  summarize(b_w = sum(rating - mu - b_m - b_u - b_g - b_y)/(n()+lambda))

# day effect (b_w)
day_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  group_by(day_of_week) %>%
  summarize(b_d = sum(rating - mu - b_m - b_u - b_g - b_y - b_w)/(n()+lambda))

# create prediction
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  left_join(day_avgs, by='day_of_week') %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y + b_w + b_d) %>%
  pull(pred)
```

```r
  return(RMSE(predicted_ratings, testset$rating))
}

# Define a set of lambdas as a sequence to try.
# you need to try a few ranges.e.g I tried 0 to 100, and narrowd down 0 to 10
#lambdas <- seq(0, 100, 2)
lambdas <- seq(0, 10, 0.25)
```

```r
# Use sapply like a for loop to try all the lambdas and calculate the final RMSE
# this will take quite some time around 10 to 15 min on an i7 13th gen with 16 gb.


rmses <- sapply(lambdas,
                regularization,
                trainset = train_set,
                testset = test_set)

# Plot the lambda vs RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
  geom_point()
```



```
# We pick the lambda that returns the lowest RMSE, which turns out to be 5.25
lambda <- lambdas[which.min(rmses)]
#lambda <- 5.25
lambda
```

```
## [1] 5.25
```

```
# retrieve the lowest RMSE, and add it.
lowest_rmse_index<- which.min(rmses)
model_7_rmse<- rmses[lowest_rmse_index]
model_7_rmse
```

```
## [1] 0.8651089
```

```
rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Regularization (Movie + User + genres + year + week + day) "
  RMSE = model_7_rmse ,
  Improvement = model_6_rmse - model_7_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161
Movie + User + Genres + Year + Week Effects	0.8658375	0.0000753
Movie + User + Genres + Year + Week + Day Effects	0.8658364	0.0000010
Movie + User + Genres + Year + Week + Day + Rating Effects	0.9298308	-0.0639944
Regularization (Movie + User + genres + year + week + day)	0.8651089	0.0007276

```
# Using Regularization, we are able to achieve another RMSE reduction, which is
# larger than all the effects of genres + year + week + day combined
# the best results I got using bias and regularization on edx data set
# was validated with RMSE 0.8651089, obviously I still didn't try it on final_handout_
```

Using Regularization, we are able to achieve another RMSE reduction, which is larger than all the effects of Genres + Year + Week + Day combined the best results I got using bias and regularization on edx dataset which was tested with RMSE 0.8651089, obviously I still didn't validate on final_handout_set.

4.5.10 Matrix factorization Method:

I started testing with recommenderlab [7] as its referenced in 33.11.2 Connection to SVD and PCA "we recommend trying the recommenderlab package".

I started exploring this package, but I faced issues that I need around 34 GB of ram to run the code, which my pc doesn't have. So I dropped this package, and researched for another similar approach using matrix factorization and stubble upon the following reconsystem [8]

I read the github readme and code and adjusted my the below code to fit my need to achieve the goal. The below approach is only for evaluation, as we can use my previous model Regularization + Linear which achieved RMSE 0.8651089 on edx dataset.

```
##########################################
# Matrix factorization Method: Recommenderlab/Recosystem
#------------------------------------------
# I started testing with recommenderlab ^[https://github.com/mhahsler/recommenderlab]
# as its referenced in 33.11.2 Connection to SVD and PCA "we recommend trying the
# recommenderlab package".
# I started exploring this package, but I faced issues that I need around 34 GB
# of ram to run the code, which my pc doesn't have.
# So I dropped this package, and researched for another similar approach using
# matrix factorization and stubble upon the following recosystem
# ^[https://github.com/yixuan/recosystem]
# I read the github readme and code and adjusted my the below code to fit my
# need to achieve the goal. The below approach is only for evaluation, as we can
# use my previous model Regularization + Linear which achieved RMSE 0.8651089 on edx d
##########################################


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

# set seed to allow for recreation of the same results
set.seed(123, sample.kind = "Rounding")


# Convert the train, test and validation sets into reconsystem input format
train_data <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))
test_data <- with(test_set, data_memory(user_index = userId,
```

[7] https://github.com/mhahsler/recommenderlab
[8] https://github.com/yixuan/recosystem

```
                                          item_index = movieId,
                                          rating = rating))

validation_reco <- with(final_holdout_test, data_memory(user_index = userId,
                                                        item_index = movieId,
                                                        rating = rating))

# Create the Recon model object
r <- recosystem::Reco()

# Identifying the best tuning parameters (I used the default ranges)
# this might take several minutes
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread = 4, niter = 10))
```

```
# Run the training the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9915   1.0016e+07
##    1       0.8789   8.0825e+06
##    2       0.8474   7.5100e+06
##    3       0.8260   7.1630e+06
##    4       0.8092   6.9128e+06
##    5       0.7962   6.7342e+06
##    6       0.7854   6.5931e+06
##    7       0.7757   6.4789e+06
##    8       0.7674   6.3862e+06
##    9       0.7601   6.3077e+06
##   10       0.7537   6.2373e+06
##   11       0.7480   6.1802e+06
##   12       0.7429   6.1308e+06
##   13       0.7383   6.0861e+06
##   14       0.7342   6.0471e+06
##   15       0.7305   6.0146e+06
##   16       0.7270   5.9822e+06
##   17       0.7239   5.9550e+06
##   18       0.7210   5.9301e+06
##   19       0.7183   5.9062e+06
```

```
# validating the model predictions
y_hat_final_reco <- r$predict(test_data, out_memory())


# now for the interesting part, lets calculate the RMSE, save and print it
model_8_rmse <- RMSE(y_hat_final_reco, test_set$rating)

rmse_results <- bind_rows(
  rmse_results,
  data_frame(Method_on_EDX="Matrix Factorization using Recosystem ",
  RMSE = model_8_rmse ,
  Improvement = model_7_rmse - model_8_rmse))
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161
Movie + User + Genres + Year + Week Effects	0.8658375	0.0000753
Movie + User + Genres + Year + Week + Day Effects	0.8658364	0.0000010
Movie + User + Genres + Year + Week + Day + Rating Effects	0.9298308	-0.0639944
Regularization (Movie + User + genres + year + week + day)	0.8651089	0.0007276
Matrix Factorization using Recosystem	0.7904119	0.0746970

```
# As you can see the results are significantly improved similar to that of user
# effects. This is great results.
```

As you can see the results are significantly improved similar to that of user effects. This is great results.

4.6 Methods used on final_holdout_test Dataset

Since we are satisfied with our results (close to 0.865) using our edx dataset, now let's predict on the final_holdout_test dataset using the following 2 methods: A) Regularization + Linear Method B) Matrix factorization: Recommenderlab/Recosystem

4.6.1 Regularization + Linear Method

First we need to add date, year, day_of_week, week in the final_holdout_test this will not affect the accuracy. Let's use the optimal lambda we extracted earlier from the train_set

```
##############################################
# Regularization + Linear Method
# on final_holdout_test
##############################################

# First we need to add date, year, day_of_week, week in the final_holdout_test
# this will not affect the accuracy
final_holdout_test$date  <- as_datetime(final_holdout_test$timestamp, origin = "1970-01-
final_holdout_test$year <- year(final_holdout_test$date)
final_holdout_test$day_of_week <- weekdays(final_holdout_test$date)
final_holdout_test$week <- round_date(final_holdout_test$date, unit = "week")

# let's use the optimal lambda we extracted earlier from the train_set
lambda<-5.25

# Run the below Regularization + Linear Method

# Mean (mu)
mu <- mean(train_set$rating)

# Movie effect (b_m)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - mu)/(n()+lambda))

# User effect (b_u)
user_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - mu)/(n()+lambda))

# genres effect (b_g)
genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_m - b_u)/(n()+lambda))

# year effect (b_y)
year_avgs <- train_set %>%
```

```
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_m - b_u - b_g)/(n()+lambda))

# week effect (b_w)
week_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  group_by(week) %>%
  summarize(b_w = sum(rating - mu - b_m - b_u - b_g - b_y)/(n()+lambda))

# day effect (b_w)
day_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  group_by(day_of_week) %>%
  summarize(b_d = sum(rating - mu - b_m - b_u - b_g - b_y - b_w)/(n()+lambda))

# create prediction
predicted_ratings <- final_holdout_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(week_avgs, by='week') %>%
  left_join(day_avgs, by='day_of_week') %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y + b_w + b_d) %>%
  pull(pred)

# summary show us there are 10 NA entires
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -0.406   3.138   3.564   3.512   3.941   5.979      10
```

```
# lets replace them with the mean
mean_predicted_rating <- mean(predicted_ratings, na.rm = TRUE)
predicted_ratings[is.na(predicted_ratings)] <- mean_predicted_rating
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -0.406   3.138   3.564   3.512   3.941   5.979
```

```
# now for the interesting part, lets calculate the RMSE, save and print it
model_9_rmse<- (RMSE(predicted_ratings, final_holdout_test$rating))



rmse_results_f <- data_frame(
  Method_on_final_holdout_test = "Regularization (Movie + User + Genres + Year + Week +
  RMSE_Final = model_9_rmse,
  RMSE_EDX = model_7_rmse,
  Comparison = model_7_rmse - model_9_rmse)
rmse_results_f %>% knitr::kable()
```

Method_on_final_holdout_test	RMSE_Final	RMSE_EDX	Comparison
Regularization (Movie + User + Genres + Year + Week + Day)	0.8652253	0.8651089	-0.0001165

```
# as you can see the results is 0.8652253 using regularization with linear bias
# on the final_holdout_test which is very similar to 0.8651089 on edx data.
```

As you can see the results is 0.8652253 using regularization with linear bias on the final_holdout_test which is very similar to 0.8651089 on edx data.

4.6.2 Matrix factorization Method:

We will rerun our prediction using object r (Recosystem) that was trained earlier but compare and predict using final_handout_test dataset.

```
##########################################
# Matrix Factorization using recosystem
# on final_holdout_test
##########################################
```

```
# validating the model predictions
y_hat_final_reco_2 <- r$predict(validation_reco, out_memory())


# now for the interesting part, lets calculate the RMSE, save and print it
model_10_rmse <- RMSE(y_hat_final_reco_2 , final_holdout_test$rating)

rmse_results_f <- bind_rows(
  rmse_results_f,
  data_frame(Method_on_final_holdout_test="Matrix Factorization using recosystem",
  RMSE_Final = model_10_rmse,
  RMSE_EDX = model_8_rmse,
  Comparison = model_8_rmse - model_10_rmse))
rmse_results_f %>% knitr::kable()
```

Method_on_final_holdout_test	RMSE_Final	RMSE_EDX	Comparison
Regularization (Movie + User + Genres + Year + Week + Day)	0.8652253	0.8651089	-0.0001165
Matrix Factorization using recosystem	0.7905823	0.7904119	-0.0001704

```
# As you can see the results is 0.7906104 using Matrix Factorization using
# recosystem on the final_holdout_test which is very similar to 0.7906201 on edx data.
```

As you can see the results is 0.7906104 using Matrix Factorization using recosystem on the final_holdout_test which is very similar to 0.7906201 on edx data.

5 Summary Results

As we can see from the below table we have achieved satisfactory results using Regularization + linear method on many parameters (Movie, user, genres, year, week,day_of_week). We also explored the usage of Matrix factorization which even provided even better results.

Results below on edx dataset:

```
rmse_results %>% knitr::kable()
```

Method_on_EDX	RMSE	Improvement
Median Rating 2.5	1.4661154	0.0000000
Overall Average	1.0600131	0.4061023

Method_on_EDX	RMSE	Improvement
Movie Effect	0.9436204	0.1163927
Movie + User Effects	0.8662966	0.0773238
Movie + User + Genres Effects	0.8659288	0.0003678
Movie + User + Genres + Year Effects	0.8659128	0.0000161
Movie + User + Genres + Year + Week Effects	0.8658375	0.0000753
Movie + User + Genres + Year + Week + Day Effects	0.8658364	0.0000010
Movie + User + Genres + Year + Week + Day + Rating Effects	0.9298308	-0.0639944
Regularization (Movie + User + genres + year + week + day)	0.8651089	0.0007276
Matrix Factorization using Recosystem	0.7904119	0.0746970

Results below on final_holdout_test dataset, we also provided comparison with edx dataset.

```
rmse_results_f %>% knitr::kable()
```

Method_on_final_holdout_test	RMSE_Final	RMSE_EDX	Comparison
Regularization (Movie + User + Genres + Year + Week + Day)	0.8652253	0.8651089	-0.0001165
Matrix Factorization using recosystem	0.7905823	0.7904119	-0.0001704

6 Conclusion

As in any data science project, we started with preparing the dataset, and explored its size and possible predictors.

I had to review the machine learning course particularly the recommendation system section, which gave valuable insights on how to tackle this project. The main reference was the Netflix blog [9] which helped me explore different approaches.

We started with linear random model, and kept on adding all the columns/ parameters as predictors and excluded those that didn't contribute in reducing RMSE.

We then used Regularization which improved the RMSE slightly and finally evaluated the matrix factorization model which significantly improved the RMSE.

Lastly we validated our models on the final_handout_test dataset, whereby regularization + linear bias achieved an RMSE of 0.8652253 and matrix factorization with an RMSE of 0.7906104.

[9] https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

6.1 Limitation

As I am currently using an i7 13th generation 16 GB laptop without GPU, I was not able to run Matrix factorization with recommenderlab as it required 34 GB. I would have liked to explore that option. There are also other models described in the Netflix blog that I would like to explore but this would be compute intensive and would not run practically on my laptop.

6.2 Future Work

As I have knowledge in azure cloud, it would be interesting to run the same project on the cloud using different algorithm provided by Azure Machine Learning studio and explore performance/cost/accuracy of such models.