MovieLens Report

Riya Michael

01/07/2021

```
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret")
if(!require(data.table)) install.packages("data.table")
if(!require(lubridate)) install.packages("lubridate")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(dplyr)) install.packages("dplyr")
if(!require(recosystem)) install.packages("recosystem")
if(!require(recommenderlab)) install.packages("recommenderlab")

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(ggplot2)
library(dplyr)
library(recosystem)
library(recommenderlab)
library(markdown)
library(knitr)

# MovieLens 10M data set:
# https://grouplens.org/data sets/movielens/10m/
# http://files.grouplens.org/data sets/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)
```

```
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)
######
```

I. Introduction

Machine Learning has proved to be a prominent and significant analytical tool which is beneficial to every organization irrespective of the sector it belongs to. A relevant use of machine learning algorithms can be seen in the form of recommendation systems especially implemented by Netflix. The main purpose of a recommendation system is to build a model which uses historical data pertaining to the movies various users have watched and rated as an input. This input helps the data scientist to perform different machine learning techniques and detect any possible trends or patterns and finally generate an output in the form of a recommendation system. This recommendation system essentially suggests movies to various users based on their past ratings which are detected through patterns by the built model.

The aim of this project is to build a recommendation system on a large data set, which is typically time consuming and tedious. This is done with the help of machine learning techniques such as Regularization and Matrix Factorization along with a few basic models.

Data The data set can be downloaded from the grouplens.org site (https://grouplens.org/datasets/movielens/10m/). Depicted below is the basic information about the data set which gives us a basic understanding about the data.

```
dim(edx)
```

```
## [1] 9000061       6
```

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                            genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
```

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

```
tail(edx)
```

```
##    userId movieId rating  timestamp                              title
## 1:  56915   62245    4.0 1223985447 Music Room, The (Jalsaghar) (1958)
## 2:  59269   59680    3.0 1229014701           One Hour with You (1932)
## 3:  59269   64325    3.0 1229014646             Long Night, The (1947)
## 4:  59342   61768    0.5 1230070861          Accused (Anklaget) (2005)
## 5:  60713    4820    2.0 1119156754       Won't Anybody Listen? (2000)
## 6:  68986   61950    3.5 1223376391                    Boot Camp (2007)
##                                    genres
## 1:                                  Drama
## 2:                 Comedy|Musical|Romance
## 3: Crime|Drama|Film-Noir|Romance|Thriller
## 4:                                  Drama
## 5:                            Documentary
## 6:                               Thriller
```

```
summary(edx)
```

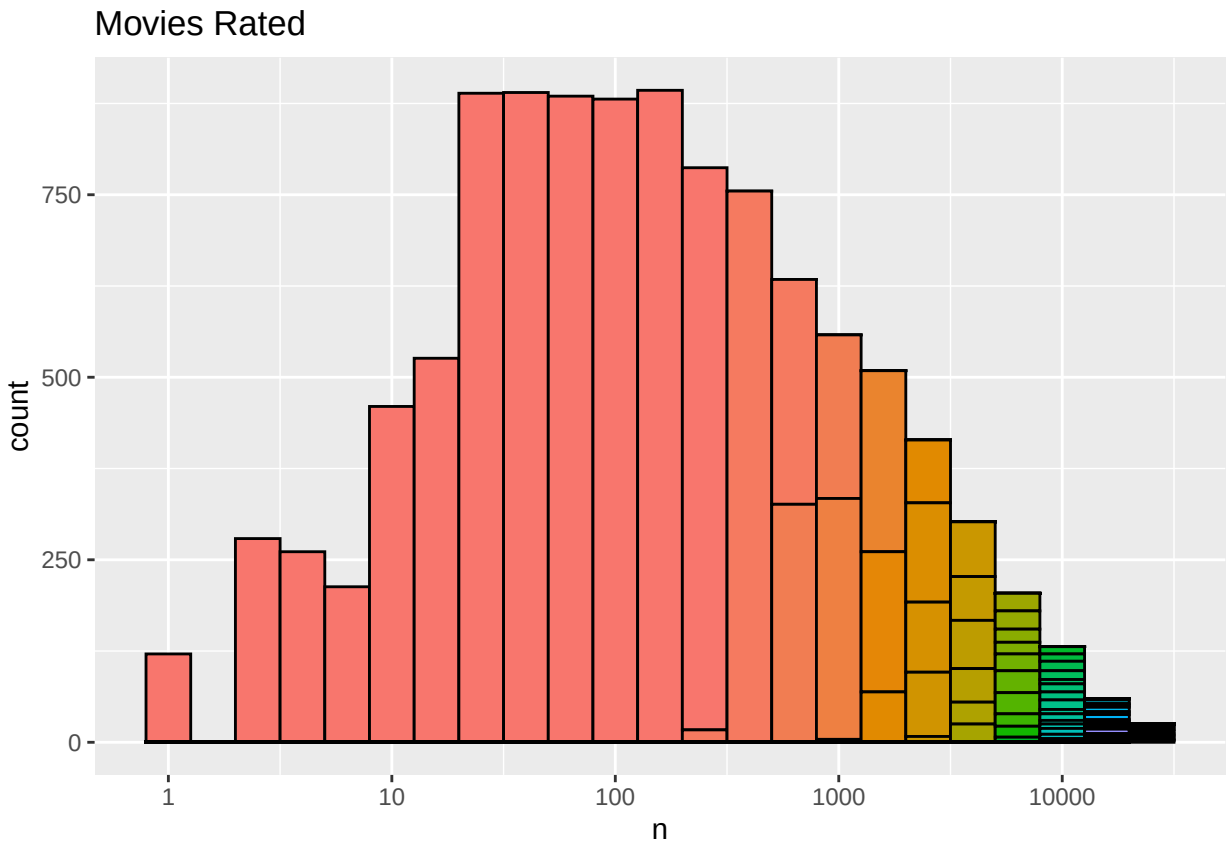
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

From the above mentioned details, we can observe that the data set contains 10 million ratings across. The rating is marked on a scale of 1 to 5, where 5 is the highest rating a movie can get.

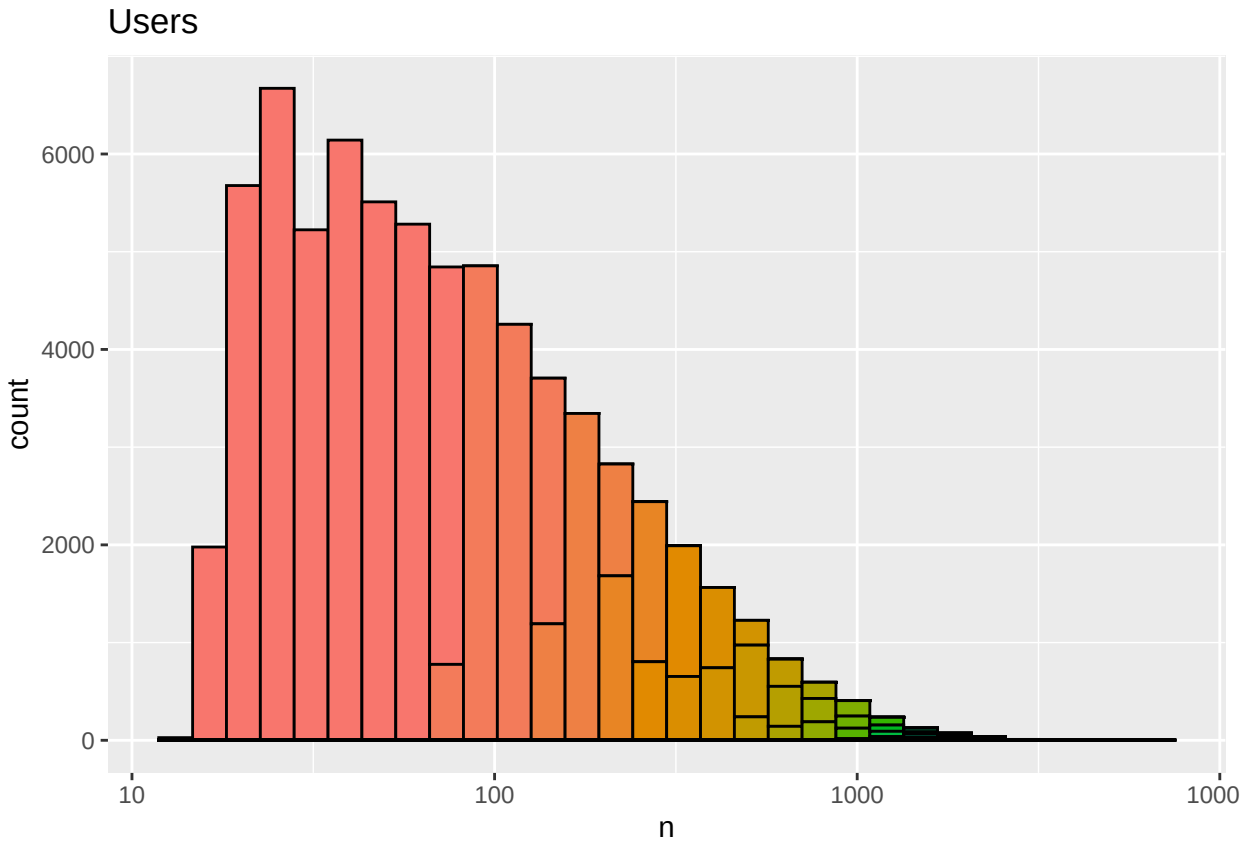
```
## [1] FALSE
```

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

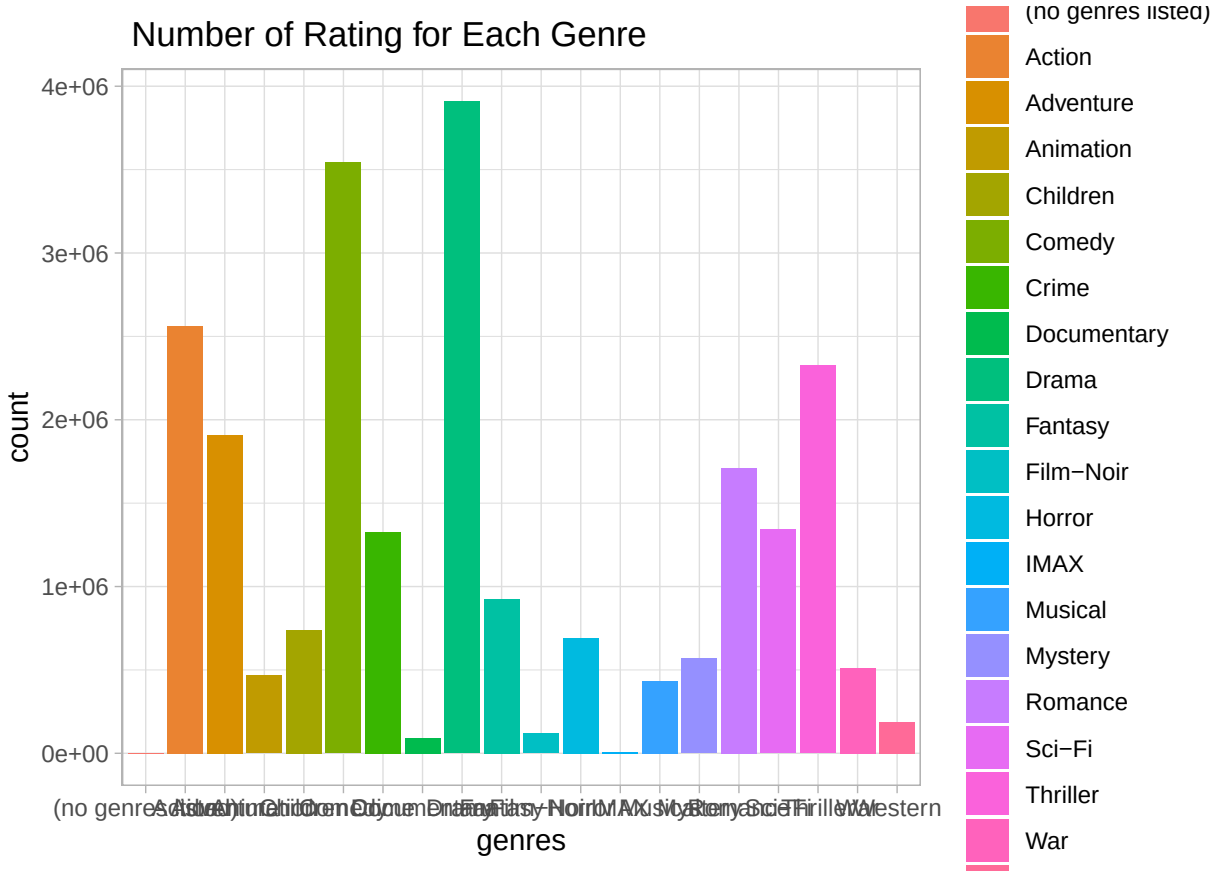
From the above figures, we can observe that the number of users that have rated different movies is close to 70,000 and the number of movies that have been rated bu multiple users is around 10,000.



The graph above depicts the number of ratings received by 'n' number of movies.



The graph above shows us the number of ratings that 'n' number of users have given.



The above graph depicted the number ratings given to movies (based on their genres). We can see that Dramas received the highest number of ratings while IMAX received the lowest number of ratings.

II. Analysis

For the sake of evaluating our model, we will split our movielens data set into train (edx)and test (validation). For a more detailed analysis, in this project, the edx set has been further split into train and test sets where we will use our train set to fit the model and test out the accuracy of the model by running the model on the test set. Since, we are running multiple models, we will track the accuracy of the same on the test sets (of edx) and finally arrive at the optimal model. This optimal model will be used to finally fit on the validation set (final hold - out set). Hence, we will compare the predicted ratings and actual ratings of the validation set.

To verify the accuracy of every model, we will be using RMSE (Root Mean Squared Error) as our success indicator. A lower RMSE is preferred as it indicates that our model generates lesser errors.

1. Simple Average The first model applied here is the simple average across every movie and every user to predict our ratings. The model follows the equation:

$$Y_{u,i} = \mu, \tag{1}$$

Here, $Y_{u,i}$ = Predicted Rating for user u and movie i μ = Average of all the ratings across every movie and user

```r
#Calculating the basic average of ratings
mu <- mean(train$rating)
mu
```

```
## [1] 3.51238
```

```r
#predicting the ratings for the test set using the average and thereby getting the RMSE
avg_RMSE <- RMSE(test$rating,mu)
avg_RMSE
```

```
## [1] 1.059643
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

2. Movie Effect Model From the RMSE reported above, we can observe that it is on the higher side and hence to reduce our RMSE, we consider movie bias term b_u.

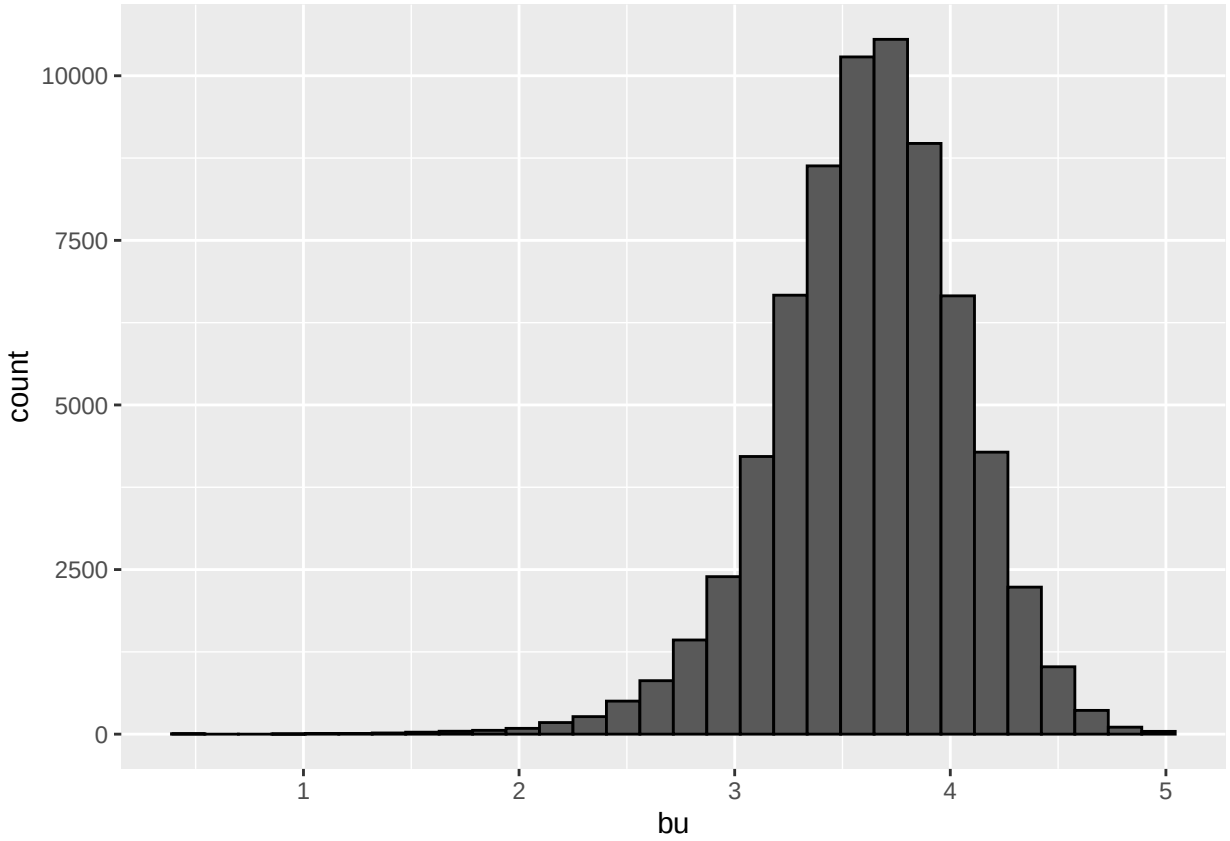
$$Y_{u,i} = \mu + b_i. \tag{2}$$

```r
#Now if we consider movie bias effects
mu <- mean(train$rating)

movie_avg <- train %>%
  group_by(movieId) %>%
  summarize(bi = mean(rating - mu))

bi <- train %>%
  group_by(movieId) %>%
  summarize(bi = mean(rating - mu))

#Predicting ratings with mu and bi
predicted_ratings <- mu + test %>%
  left_join(movie_avg, by = 'movieId') %>%
  pull(bi)
```

```
## # A tibble: 2 x 2
##   Method          RMSE
##   <chr>          <dbl>
## 1 Simple average 1.06
## 2 Movie Effect   0.943
```

3. Movie + User Effect Model
From the RMSE generated above, we can observe that the value has reduced. To improve our accuracy we can further account for the user bias too, which is denoted as b_u.

$$Y_{u,i} = \mu + b_i + b_u. \tag{3}$$




```
#Including user bias into the algorithm
user_avg <- train %>%
  left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi))

#Checking whether there is any improvement in the RMSE
predicted_ratings <- test %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  mutate(pred = mu + bi + bu) %>%
  .$pred
```

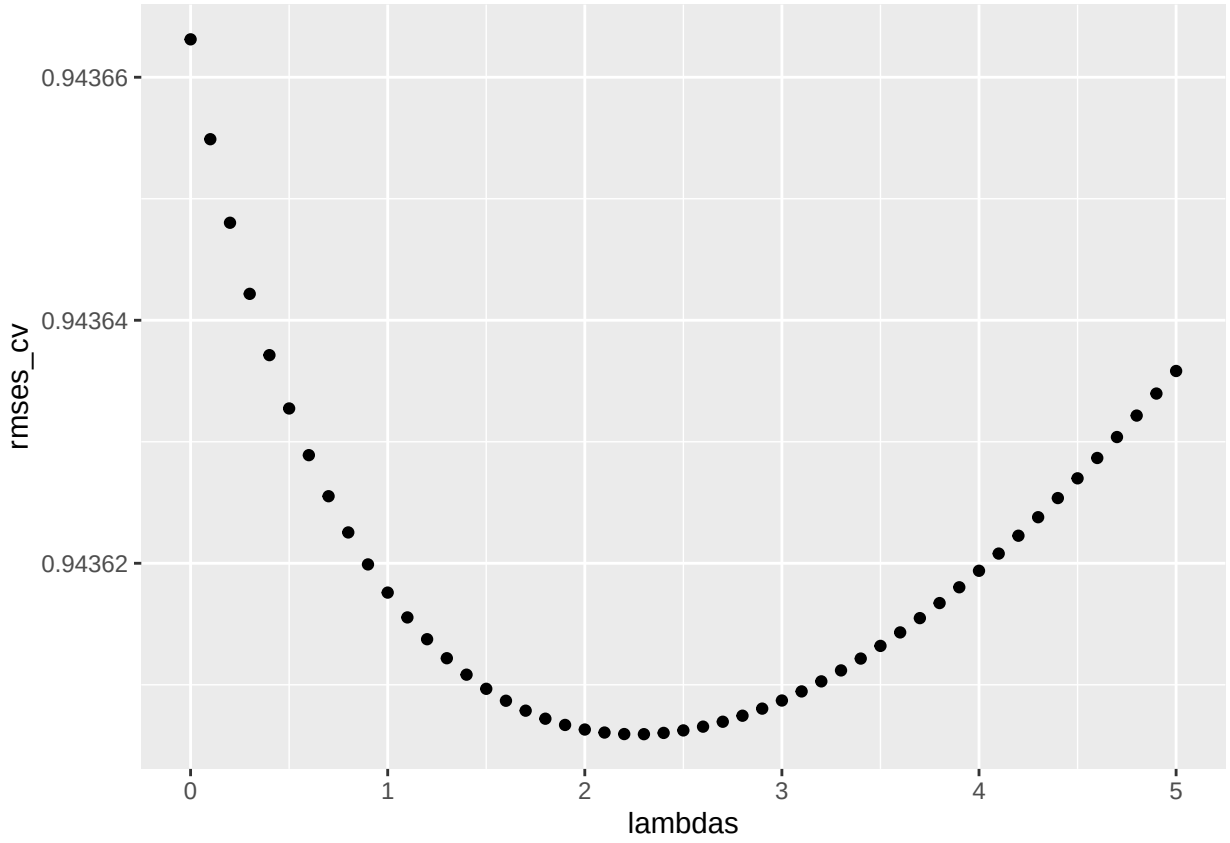
```
## # A tibble: 3 x 2
##   Method                     RMSE
##   <chr>                     <dbl>
## 1 Simple average            1.06
## 2 Movie Effect              0.943
## 3 Movie + User Effects Model 0.866
```

Now we can see a considerable improvement in our reported RMSE value.

4. Regularization with Movie Effects We carry out regularization in order to eliminate any obscure predictions. Suppose our model generates a list of top 10 movies based on the predicted ratings, this may

not necessarily be correct. This is because this situation may arise out of just 1 user rating a particular movie 5 and hence that automatically comes under the list of top 10 movies. However, this occurs as a result of insufficient or negligible number of users rating these movies. Hence, we perform Regularization to avoid such mistakes.

```
#REGULARISATION OF MOVIE EFFECTS ONLY

#Start by splitting the data by cross - validation
#Use 10-fold cross validation to pick a lambda for movie effects regularization
#Split the data into 10 parts
set.seed(2019, sample.kind = "Rounding")
cv_splits <- createFolds(edx$rating, k=10, returnTrain =TRUE)

#Define a matrix to store the results of cross validation
rmses <- matrix(nrow=10, ncol=51)
lambdas <- seq(0, 5, 0.1)

#Perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
train <- edx[cv_splits[[k]],]
 test <- edx[-cv_splits[[k]],]

#Make sure userId and movieId in test set are also in the train set
test_final <- test %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

removed <- anti_join(test, test_final)
train_final <- rbind(train, removed)

mu <- mean(train_final$rating)
just_the_sum <- train_final %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), ni = n())

rmses[k,] <- sapply(lambdas, function(l){
  predicted_ratings <- test_final %>%
  left_join(just_the_sum, by='movieId') %>%
  mutate(b_i = s/(ni+l)) %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
  return(RMSE(predicted_ratings, test_final$rating))})
 }

rmses_cv <- colMeans(rmses)
qplot(lambdas,rmses_cv)
```




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

#2.2 happens to be the optimal value for lamda where the RMSE is the lowest.
#Hence, tuning the train set with a lamda of 2.2 for movie effect and check the RMSE generated.

#Model generation and prediction (on the test set)
mu <- mean(train_final$rating)
movie_reg_avgs <- train_final %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
predicted_ratings <- test_final %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

bi_reg_rmse <- RMSE(predicted_ratings, test_final$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularization using only movie",
                                     RMSE = bi_reg_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Simple average                 1.06
```

```
## 2 Movie Effect                    0.943
## 3 Movie + User Effects Model      0.866
## 4 Regularization using only movie 0.944
```

```
#We can see a negligible improvement after performing regularization.
#Applying regularization with movie and user effects to check whether that improves the RMSE.
```

On evaluating the RMSE generated from the regularization model using movie effects only, we can see a decrease in our RMSE value. It is also evident that the optimal lamda value i.e. the point at which RMSE is lowest is when lamda is 2.2.

5. Regularization with Movie Effects and User Effects By running regularization with movie and user effects we can try to optimize our model and improve our RMSE by account for the user effect also.

```
#REGULARISATION OF MOVIE + USER EFFECTS

#Start by setting a value for lamda for movie and user effects each
lambda_i <- 2.2
lambda_u <- seq(0, 8, 0.1)
rmses_2 <- matrix(nrow=10,ncol=length(lambda_u))

#Perform 10-fold cross validation to determine the optimal lambda
for(k in 1:10) {
  train <- edx[cv_splits[[k]],]
  test <- edx[-cv_splits[[k]],]

#Make sure userId and movieId in test set are also in the train set
  test_final <- test %>%
    semi_join(train, by = "movieId") %>%
    semi_join(train, by = "userId")

  removed <- anti_join(test, test)
  train_final <- rbind(train, removed)

  mu <- mean(train_final$rating)

  rmses_2[k,] <- sapply(lambda_u, function(l){
    b_i <- train_final %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+lambda_i))
    b_u <- train_final %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
      test_final %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      pull(pred)
    return(RMSE(predicted_ratings, test_final$rating))
  })
```

```
}
rmses_2
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,] 0.8656631 0.8656439 0.8656254 0.8656075 0.8655903 0.8655736 0.8655575
##  [2,] 0.8646600 0.8646395 0.8646197 0.8646006 0.8645821 0.8645642 0.8645469
##  [3,] 0.8649624 0.8649424 0.8649231 0.8649044 0.8648863 0.8648688 0.8648519
##  [4,] 0.8667099 0.8666900 0.8666708 0.8666521 0.8666341 0.8666167 0.8665998
##  [5,] 0.8643749 0.8643524 0.8643306 0.8643095 0.8642890 0.8642692 0.8642499
##  [6,] 0.8665943 0.8665737 0.8665537 0.8665344 0.8665157 0.8664976 0.8664801
##  [7,] 0.8644561 0.8644365 0.8644175 0.8643991 0.8643814 0.8643643 0.8643477
##  [8,] 0.8663181 0.8662991 0.8662807 0.8662630 0.8662458 0.8662293 0.8662133
##  [9,] 0.8645782 0.8645594 0.8645414 0.8645239 0.8645071 0.8644909 0.8644753
## [10,] 0.8660842 0.8660633 0.8660430 0.8660233 0.8660042 0.8659858 0.8659680
##            [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
##  [1,] 0.8655420 0.8655270 0.8655126 0.8654987 0.8654853 0.8654725 0.8654601
##  [2,] 0.8645302 0.8645141 0.8644986 0.8644836 0.8644692 0.8644553 0.8644419
##  [3,] 0.8648355 0.8648198 0.8648045 0.8647899 0.8647757 0.8647621 0.8647489
##  [4,] 0.8665836 0.8665679 0.8665527 0.8665381 0.8665240 0.8665105 0.8664974
##  [5,] 0.8642313 0.8642132 0.8641957 0.8641788 0.8641624 0.8641466 0.8641313
##  [6,] 0.8664632 0.8664469 0.8664311 0.8664159 0.8664013 0.8663871 0.8663735
##  [7,] 0.8643317 0.8643163 0.8643014 0.8642871 0.8642732 0.8642599 0.8642471
##  [8,] 0.8661978 0.8661830 0.8661686 0.8661548 0.8661415 0.8661287 0.8661164
##  [9,] 0.8644602 0.8644458 0.8644318 0.8644184 0.8644056 0.8643932 0.8643814
## [10,] 0.8659507 0.8659340 0.8659179 0.8659024 0.8658873 0.8658728 0.8658589
##           [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
##  [1,] 0.8654483 0.8654369 0.8654260 0.8654155 0.8654055 0.8653960 0.8653869
##  [2,] 0.8644290 0.8644166 0.8644047 0.8643933 0.8643824 0.8643719 0.8643619
##  [3,] 0.8647363 0.8647241 0.8647125 0.8647013 0.8646905 0.8646802 0.8646703
##  [4,] 0.8664849 0.8664728 0.8664612 0.8664500 0.8664393 0.8664291 0.8664193
##  [5,] 0.8641165 0.8641022 0.8640884 0.8640750 0.8640622 0.8640498 0.8640379
##  [6,] 0.8663604 0.8663477 0.8663356 0.8663239 0.8663127 0.8663020 0.8662917
##  [7,] 0.8642348 0.8642230 0.8642116 0.8642007 0.8641903 0.8641803 0.8641707
##  [8,] 0.8661046 0.8660933 0.8660824 0.8660720 0.8660620 0.8660525 0.8660434
##  [9,] 0.8643700 0.8643592 0.8643488 0.8643389 0.8643294 0.8643203 0.8643117
## [10,] 0.8658454 0.8658324 0.8658199 0.8658078 0.8657963 0.8657852 0.8657745
##           [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
##  [1,] 0.8653782 0.8653699 0.8653620 0.8653545 0.8653475 0.8653408 0.8653344
##  [2,] 0.8643523 0.8643431 0.8643344 0.8643260 0.8643181 0.8643106 0.8643034
##  [3,] 0.8646608 0.8646518 0.8646432 0.8646350 0.8646271 0.8646197 0.8646126
##  [4,] 0.8664099 0.8664010 0.8663924 0.8663843 0.8663765 0.8663691 0.8663622
##  [5,] 0.8640264 0.8640154 0.8640048 0.8639946 0.8639848 0.8639754 0.8639665
##  [6,] 0.8662818 0.8662724 0.8662634 0.8662547 0.8662465 0.8662387 0.8662313
##  [7,] 0.8641616 0.8641529 0.8641445 0.8641366 0.8641291 0.8641220 0.8641152
##  [8,] 0.8660347 0.8660264 0.8660186 0.8660111 0.8660040 0.8659973 0.8659909
##  [9,] 0.8643036 0.8642958 0.8642884 0.8642815 0.8642749 0.8642687 0.8642629
## [10,] 0.8657643 0.8657545 0.8657451 0.8657361 0.8657275 0.8657193 0.8657115
##           [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
##  [1,] 0.8653285 0.8653229 0.8653176 0.8653127 0.8653082 0.8653040 0.8653001
##  [2,] 0.8642967 0.8642903 0.8642843 0.8642786 0.8642733 0.8642683 0.8642637
##  [3,] 0.8646059 0.8645996 0.8645936 0.8645880 0.8645827 0.8645778 0.8645732
##  [4,] 0.8663555 0.8663493 0.8663434 0.8663379 0.8663327 0.8663278 0.8663233
##  [5,] 0.8639579 0.8639496 0.8639418 0.8639343 0.8639272 0.8639205 0.8639141
```

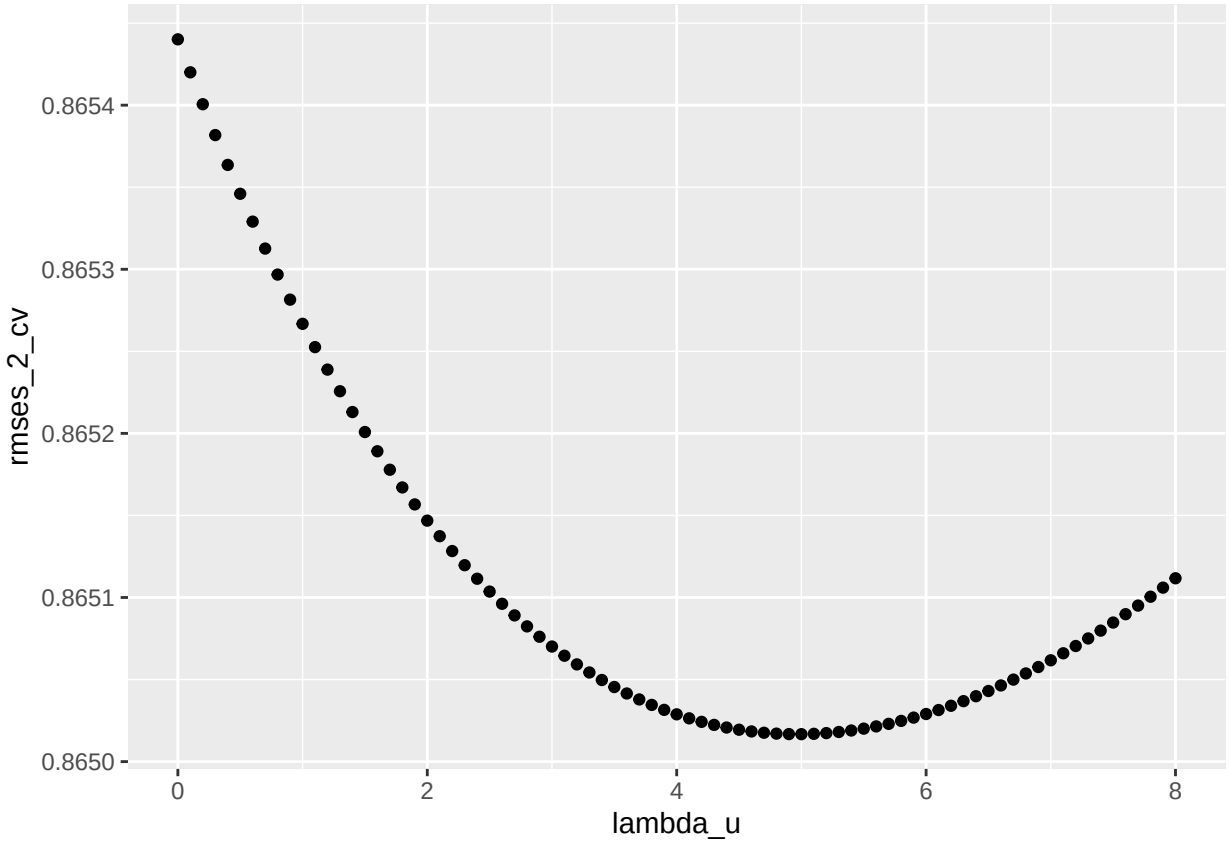
```
##  [6,] 0.8662242 0.8662176 0.8662113 0.8662053 0.8661997 0.8661944 0.8661895
##  [7,] 0.8641088 0.8641028 0.8640971 0.8640918 0.8640868 0.8640821 0.8640778
##  [8,] 0.8659850 0.8659793 0.8659741 0.8659691 0.8659645 0.8659603 0.8659563
##  [9,] 0.8642575 0.8642524 0.8642477 0.8642433 0.8642392 0.8642355 0.8642321
## [10,] 0.8657041 0.8656971 0.8656904 0.8656841 0.8656781 0.8656725 0.8656672
##           [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
##  [1,] 0.8652965 0.8652932 0.8652903 0.8652877 0.8652853 0.8652833 0.8652815
##  [2,] 0.8642594 0.8642554 0.8642518 0.8642485 0.8642454 0.8642427 0.8642403
##  [3,] 0.8645689 0.8645649 0.8645613 0.8645579 0.8645548 0.8645521 0.8645496
##  [4,] 0.8663191 0.8663152 0.8663116 0.8663083 0.8663053 0.8663027 0.8663003
##  [5,] 0.8639080 0.8639022 0.8638968 0.8638917 0.8638869 0.8638825 0.8638783
##  [6,] 0.8661849 0.8661807 0.8661767 0.8661731 0.8661698 0.8661667 0.8661640
##  [7,] 0.8640738 0.8640701 0.8640668 0.8640637 0.8640609 0.8640585 0.8640563
##  [8,] 0.8659527 0.8659494 0.8659464 0.8659437 0.8659413 0.8659392 0.8659373
##  [9,] 0.8642291 0.8642263 0.8642239 0.8642217 0.8642199 0.8642183 0.8642170
## [10,] 0.8656622 0.8656576 0.8656533 0.8656493 0.8656456 0.8656422 0.8656391
##           [,43]     [,44]     [,45]     [,46]     [,47]     [,48]     [,49]
##  [1,] 0.8652800 0.8652788 0.8652779 0.8652772 0.8652768 0.8652766 0.8652767
##  [2,] 0.8642382 0.8642363 0.8642347 0.8642334 0.8642324 0.8642317 0.8642312
##  [3,] 0.8645474 0.8645455 0.8645439 0.8645425 0.8645414 0.8645406 0.8645400
##  [4,] 0.8662982 0.8662964 0.8662948 0.8662935 0.8662925 0.8662918 0.8662913
##  [5,] 0.8638744 0.8638709 0.8638676 0.8638646 0.8638618 0.8638594 0.8638572
##  [6,] 0.8661616 0.8661594 0.8661575 0.8661559 0.8661546 0.8661535 0.8661527
##  [7,] 0.8640544 0.8640527 0.8640514 0.8640503 0.8640495 0.8640489 0.8640486
##  [8,] 0.8659358 0.8659345 0.8659335 0.8659327 0.8659322 0.8659320 0.8659320
##  [9,] 0.8642160 0.8642153 0.8642149 0.8642147 0.8642148 0.8642151 0.8642157
## [10,] 0.8656363 0.8656337 0.8656315 0.8656295 0.8656278 0.8656264 0.8656252
##           [,50]     [,51]     [,52]     [,53]     [,54]     [,55]     [,56]
##  [1,] 0.8652771 0.8652777 0.8652785 0.8652796 0.8652809 0.8652824 0.8652841
##  [2,] 0.8642309 0.8642309 0.8642312 0.8642317 0.8642324 0.8642334 0.8642346
##  [3,] 0.8645397 0.8645396 0.8645397 0.8645401 0.8645407 0.8645416 0.8645427
##  [4,] 0.8662910 0.8662910 0.8662912 0.8662917 0.8662924 0.8662934 0.8662945
##  [5,] 0.8638553 0.8638536 0.8638522 0.8638510 0.8638501 0.8638494 0.8638490
##  [6,] 0.8661522 0.8661519 0.8661518 0.8661520 0.8661524 0.8661531 0.8661540
##  [7,] 0.8640485 0.8640487 0.8640491 0.8640498 0.8640507 0.8640518 0.8640531
##  [8,] 0.8659322 0.8659327 0.8659335 0.8659344 0.8659356 0.8659370 0.8659387
##  [9,] 0.8642165 0.8642175 0.8642188 0.8642204 0.8642221 0.8642241 0.8642263
## [10,] 0.8656243 0.8656236 0.8656232 0.8656230 0.8656231 0.8656234 0.8656239
##           [,57]     [,58]     [,59]     [,60]     [,61]     [,62]     [,63]
##  [1,] 0.8652861 0.8652883 0.8652907 0.8652933 0.8652961 0.8652991 0.8653023
##  [2,] 0.8642360 0.8642377 0.8642395 0.8642416 0.8642439 0.8642464 0.8642491
##  [3,] 0.8645440 0.8645455 0.8645472 0.8645491 0.8645513 0.8645536 0.8645561
##  [4,] 0.8662959 0.8662975 0.8662993 0.8663014 0.8663036 0.8663060 0.8663087
##  [5,] 0.8638487 0.8638488 0.8638490 0.8638495 0.8638501 0.8638510 0.8638521
##  [6,] 0.8661551 0.8661565 0.8661580 0.8661598 0.8661618 0.8661640 0.8661664
##  [7,] 0.8640547 0.8640565 0.8640585 0.8640607 0.8640631 0.8640657 0.8640685
##  [8,] 0.8659405 0.8659426 0.8659449 0.8659474 0.8659501 0.8659529 0.8659560
##  [9,] 0.8642287 0.8642314 0.8642342 0.8642372 0.8642405 0.8642439 0.8642476
## [10,] 0.8656246 0.8656256 0.8656268 0.8656282 0.8656298 0.8656316 0.8656336
##           [,64]     [,65]     [,66]     [,67]     [,68]     [,69]     [,70]
##  [1,] 0.8653057 0.8653092 0.8653130 0.8653170 0.8653211 0.8653254 0.8653299
##  [2,] 0.8642520 0.8642551 0.8642584 0.8642619 0.8642655 0.8642694 0.8642734
##  [3,] 0.8645589 0.8645618 0.8645649 0.8645682 0.8645717 0.8645754 0.8645792
##  [4,] 0.8663115 0.8663145 0.8663177 0.8663211 0.8663247 0.8663285 0.8663324
```

```
##  [5,] 0.8638534 0.8638549 0.8638567 0.8638586 0.8638607 0.8638629 0.8638654
##  [6,] 0.8661689 0.8661717 0.8661747 0.8661779 0.8661813 0.8661848 0.8661885
##  [7,] 0.8640715 0.8640747 0.8640781 0.8640816 0.8640854 0.8640893 0.8640934
##  [8,] 0.8659593 0.8659627 0.8659664 0.8659702 0.8659742 0.8659784 0.8659828
##  [9,] 0.8642514 0.8642554 0.8642596 0.8642640 0.8642685 0.8642733 0.8642782
## [10,] 0.8656358 0.8656383 0.8656409 0.8656437 0.8656467 0.8656498 0.8656532
##           [,71]     [,72]     [,73]     [,74]     [,75]     [,76]     [,77]
##  [1,] 0.8653346 0.8653394 0.8653444 0.8653496 0.8653549 0.8653604 0.8653660
##  [2,] 0.8642776 0.8642820 0.8642866 0.8642913 0.8642962 0.8643012 0.8643064
##  [3,] 0.8645832 0.8645874 0.8645918 0.8645963 0.8646010 0.8646058 0.8646108
##  [4,] 0.8663365 0.8663408 0.8663453 0.8663499 0.8663547 0.8663596 0.8663647
##  [5,] 0.8638681 0.8638709 0.8638739 0.8638771 0.8638805 0.8638840 0.8638877
##  [6,] 0.8661924 0.8661965 0.8662008 0.8662052 0.8662098 0.8662145 0.8662194
##  [7,] 0.8640977 0.8641021 0.8641067 0.8641115 0.8641164 0.8641215 0.8641268
##  [8,] 0.8659873 0.8659920 0.8659968 0.8660019 0.8660070 0.8660124 0.8660179
##  [9,] 0.8642832 0.8642885 0.8642939 0.8642994 0.8643052 0.8643110 0.8643171
## [10,] 0.8656567 0.8656604 0.8656643 0.8656683 0.8656726 0.8656769 0.8656815
##           [,78]     [,79]     [,80]     [,81]
##  [1,] 0.8653718 0.8653777 0.8653838 0.8653900
##  [2,] 0.8643118 0.8643173 0.8643230 0.8643288
##  [3,] 0.8646160 0.8646213 0.8646267 0.8646323
##  [4,] 0.8663700 0.8663754 0.8663809 0.8663866
##  [5,] 0.8638916 0.8638956 0.8638998 0.8639041
##  [6,] 0.8662245 0.8662297 0.8662351 0.8662406
##  [7,] 0.8641322 0.8641378 0.8641435 0.8641493
##  [8,] 0.8660235 0.8660293 0.8660353 0.8660413
##  [9,] 0.8643233 0.8643296 0.8643361 0.8643427
## [10,] 0.8656862 0.8656910 0.8656960 0.8657012
```

```
rmses_2_cv <- colMeans(rmses_2)
rmses_2_cv
```

```
##  [1] 0.8654401 0.8654200 0.8654006 0.8653818 0.8653636 0.8653460 0.8653290
##  [8] 0.8653126 0.8652968 0.8652815 0.8652668 0.8652526 0.8652389 0.8652257
## [15] 0.8652130 0.8652008 0.8651891 0.8651779 0.8651671 0.8651567 0.8651468
## [22] 0.8651374 0.8651283 0.8651197 0.8651114 0.8651036 0.8650962 0.8650891
## [29] 0.8650824 0.8650761 0.8650701 0.8650645 0.8650592 0.8650543 0.8650497
## [36] 0.8650455 0.8650415 0.8650379 0.8650346 0.8650315 0.8650288 0.8650264
## [43] 0.8650242 0.8650224 0.8650208 0.8650194 0.8650184 0.8650176 0.8650171
## [50] 0.8650168 0.8650167 0.8650169 0.8650174 0.8650180 0.8650190 0.8650201
## [57] 0.8650214 0.8650230 0.8650248 0.8650268 0.8650290 0.8650314 0.8650340
## [64] 0.8650368 0.8650398 0.8650430 0.8650464 0.8650500 0.8650537 0.8650576
## [71] 0.8650617 0.8650660 0.8650704 0.8650750 0.8650798 0.8650847 0.8650898
## [78] 0.8650951 0.8651005 0.8651060 0.8651117
```

```
qplot(lambda_u,rmses_2_cv)
```




```
lambda_u <-lambda_u[which.min(rmses_2_cv)]

#4.6 happens to be the optimal lambda value for user bias.
#Hence, we can generate our final model based on regularization of movie and users with their respectiv

#Model generation and prediction
lambda_i <- 2.2
lambda_u <- 4.6
rmses_3 <- matrix(nrow=10,ncol=length(lambda_u))
mu <- mean(train_final$rating)
b_i_reg <- train_final %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_i))
b_u_reg <- train_final %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_u))
predicted_ratings <- test_final %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

biu_reg_rmse <- RMSE(predicted_ratings, test_final$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie & User Effects",
```

```
                                    RMSE = biu_reg_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Simple average                 1.06
## 2 Movie Effect                   0.943
## 3 Movie + User Effects Model     0.866
## 4 Regularization using only movie  0.944
## 5 Regularized Movie & User Effects 0.866
```

Until now, regularization using movie + user effects seems to be the most suitable model as it generates the lowest RMSE of 0.866.

6. Matrix Factorization Matrix Factorization is carried out by grouping different variables to form patterns. For example, we are aware that certain types of users will like certain movies. In other words, a user who liked the movie 'Sleepless in Seattle' has a higher chance of liking 'You've got mail' as compared to a user who liked the movie 'Star Wars'. Hence, in this way, we can form clusters or groups of users depending upon the movies they like and detect patterns. This is done through Matrix Factorization. We start by calculating residuals. Then we use the recommender system to carry out Matrix Factorization.

```
set.seed(123, sample.kind = "Rounding") # This is a randomized algorithm

#Convert the train, test and validation sets into recosystem input format
train_data <-  with(train, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating = rating))
test_data  <-  with(test, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating = rating))
validation_data  <-  with(validation, data_memory(user_index = userId,
                                      item_index = movieId,
                                      rating = rating))
#Create the model object
r <-  recosystem::Reco()

#Select the best tuning parameters
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))

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

```
## iter      tr_rmse          obj
##    0       0.9808   1.1019e+07
##    1       0.8755   8.9836e+06
##    2       0.8422   8.3432e+06
##    3       0.8206   7.9552e+06
```

```
##    4       0.8047   7.6903e+06
##    5       0.7926   7.5043e+06
##    6       0.7825   7.3589e+06
##    7       0.7739   7.2449e+06
##    8       0.7664   7.1501e+06
##    9       0.7598   7.0690e+06
##   10       0.7540   7.0018e+06
##   11       0.7487   6.9431e+06
##   12       0.7440   6.8895e+06
##   13       0.7399   6.8457e+06
##   14       0.7360   6.8057e+06
##   15       0.7324   6.7694e+06
##   16       0.7294   6.7417e+06
##   17       0.7264   6.7108e+06
##   18       0.7236   6.6846e+06
##   19       0.7211   6.6625e+06
```

```
#Calculate the predicted values for the test set
predicted_ratings_mf <-  r$predict(test_data, out_memory())
head(predicted_ratings_mf, 10)
```

```
##  [1] 3.113500 3.812977 3.873547 3.225301 3.973977 3.894884 4.108405 4.433609
##  [9] 3.194830 3.765643
```

```
rmse_mf <- RMSE(test$rating,predicted_ratings_mf)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Matrix Factorization",
                                     RMSE = rmse_mf))
rmse_results
```

```
## # A tibble: 6 x 2
##   Method                             RMSE
##   <chr>                             <dbl>
## 1 Simple average                    1.06
## 2 Movie Effect                      0.943
## 3 Movie + User Effects Model        0.866
## 4 Regularization using only movie   0.944
## 5 Regularized Movie & User Effects 0.866
## 6 Matrix Factorization              0.786
```

On performing Matrix Factorization, we can observe that we get the lowest RMSE value.

III. Results

After performing the simple average model, movie effect model, movie + user effect model, regularization using movie effect, regularization using movie + user effect and finally the matrix factorization model, we can conclude that Matrix Factorization is the most suitable machine learning algorithm. The same can be confirmed by verifying the RMSE value after running the algorithm on the validation set.

```
#Calculate the predicted values for the validation set (final hold - out set)
predicted_ratings_mf <-  r$predict(validation_data, out_memory())
head(predicted_ratings_mf, 10)
```

```
##  [1] 5.018804 3.937582 2.867634 3.901809 3.355236 4.032077 4.295608 3.019508
##  [9] 4.847739 3.290251
```

```
rmse_mf <- RMSE(validation$rating,predicted_ratings_mf)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Matrix Factorization (Validation Set)",
                                     RMSE = rmse_mf))
rmse_results
```

```
## # A tibble: 7 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Simple average                       1.06
## 2 Movie Effect                         0.943
## 3 Movie + User Effects Model           0.866
## 4 Regularization using only movie      0.944
## 5 Regularized Movie & User Effects     0.866
## 6 Matrix Factorization                 0.786
## 7 Matrix Factorization (Validation Set) 0.786
```

Since, our main aim while building the recommendation system was to minimize RMSE, we have achieved our goal by applying the Matrix Factorization and hence this is the optimal model.

IV. Conclusion

With the increasing need for machine learning algorithms required for various tasks, recommendation systems being one of them, data scientists have the opportunity to apply numerous algorithms depending on their data sets. When it comes to handling large data sets, there are limited algorithms that can aid in this process such as Regularization, Dimension Reduction and Matrix Factorization to name a few.

Among the various models that this project has implemented, matrix factorization has proved to be the most suitable model. Some of the key reasons may be since this model not only accounts for the movie and user bias effect, but also takes into consideration the residuals, i.e. the differences in user patterns and movie patterns that are detected in the data set.

A more thorough and detailed report can be formed by generating a model that accounts for the genres of various movies or even the release years of these movies. Moreover, other models like Dimension Reduction may be applied to improve the overall RMSE.