MovieLens Project

Yu-Wei Hung

8/17/2020

Introduction

In today's technologically advanced society with more and more e-commerce companies making their sales online, a recommendation system to guide consumers towards the right product is extremely valuable to both the consumers and the corporations. They allow predictions for the next product a consumer's most likely to buy, the next movie a consumer's most likely to watch, or even a restaurant a consumer's most likely to visit. Therefore, e-commerce companies often invest heavily on recommendation systems that best analyse and capture users preference and readily predicts the next item of purchase. For example, in 2006, Netflix, a popular online movie streaming company, placed out an open competition with a 1$ million USD prize to the developer that can produce an algorithm that increases the accuracy of the company's recommendation system by at least 10%. The use of recommendation systems in today's economy is clear: better recommendation systems equates to higher sales and customer satisfaction.

The main goal of this project is to build a similar recommendation system to Netflix's challenge winners and predict what users would most likely watch based on preferences.

In this document we first take a look at the data sets and goals of this project. Then, we explain the process of data exploration, visualization, and analysis. After, we present the results and discuss the performance of the model.

Data set

The data set that is used in this project is from Grouplens, that has gathered a large amount of movie rating data, which is in the Movielens website.

The complete database can be found here: (https://grouplens.org/datasets/movielens/10m/) The subset of the complete database that is used can be found here:(http://files.grouplens.org/datasets/movielens/ml-10m.zip)

The version of R used in this project and any subsequent code is for the version R 4.0.

Model Explanation

The basis of recommendation systems, or machine learning in general, is to compare the predicted value to the actual value (y to y-hat). The loss functions that is used in this model is explained here.

Mean Squared Error (MSE)

$$MSE = \frac{1}{N}\sum_{i=1}(\hat{y}_i - y_i)^2$$

The mean squared error, or MSE, is a statistical estimator that measures the averages of squared errors. The value is always positive, and values approaching 0 are better estimators.

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}(\hat{y}_i - y_i)^2}$$

The root mean squared error, or RMSE, is the square root of MSE. It is the target metric we will use in evaluating our recommendation model.

Method/Analysis

Data Preparation

The following code is provided by EdX Capstone Module to get the model started. The code first splits the original dataset into a training set (edx) and a evaluating (validation) set.

```r
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t",
                             readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                            title = as.character(title),
                                            genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
# 'Validation' set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating,
                                  times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
# Make sure userId and movieId in 'validation' set are also in 'edx' set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
# Add rows removed from 'validation' set back into 'edx' set
removed <- anti_join(temp, validation)
```

```
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

After that, we split edx set into training and test sets. Model building is done in the training set, and the test set is for testing the model. The last set, Validation set, is used later to calculate RMSE.

```
set.seed(1, sample.kind = "Rounding")
test_ind<- createDataPartition(y=edx$rating,p=0.1,times=1, list= FALSE)
train_set<- edx[-test_ind,]
pre_test<-edx[test_ind,]
test_set<- pre_test %>% semi_join(train_set, by = "movieId")%>%
                        semi_join(train_set, by = "userId")
removed<-anti_join(pre_test,test_set)
train_set<-rbind(train_set,removed)

rm(test_ind,pre_test,removed)
```

General statistics/Analysis

Before we get started, lets take a look at the dataset:

```
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
```

This is what the first few lines of the dataset looks like.

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

From this line of code, we can see that the dataset consists of 6 columns, which includes: userId, movieId, rating, timestamp, title, and genres.

```
dim(edx)
```

```
## [1] 9000055       6
```

There are 6 column (from above), and 9000055 rows of data.

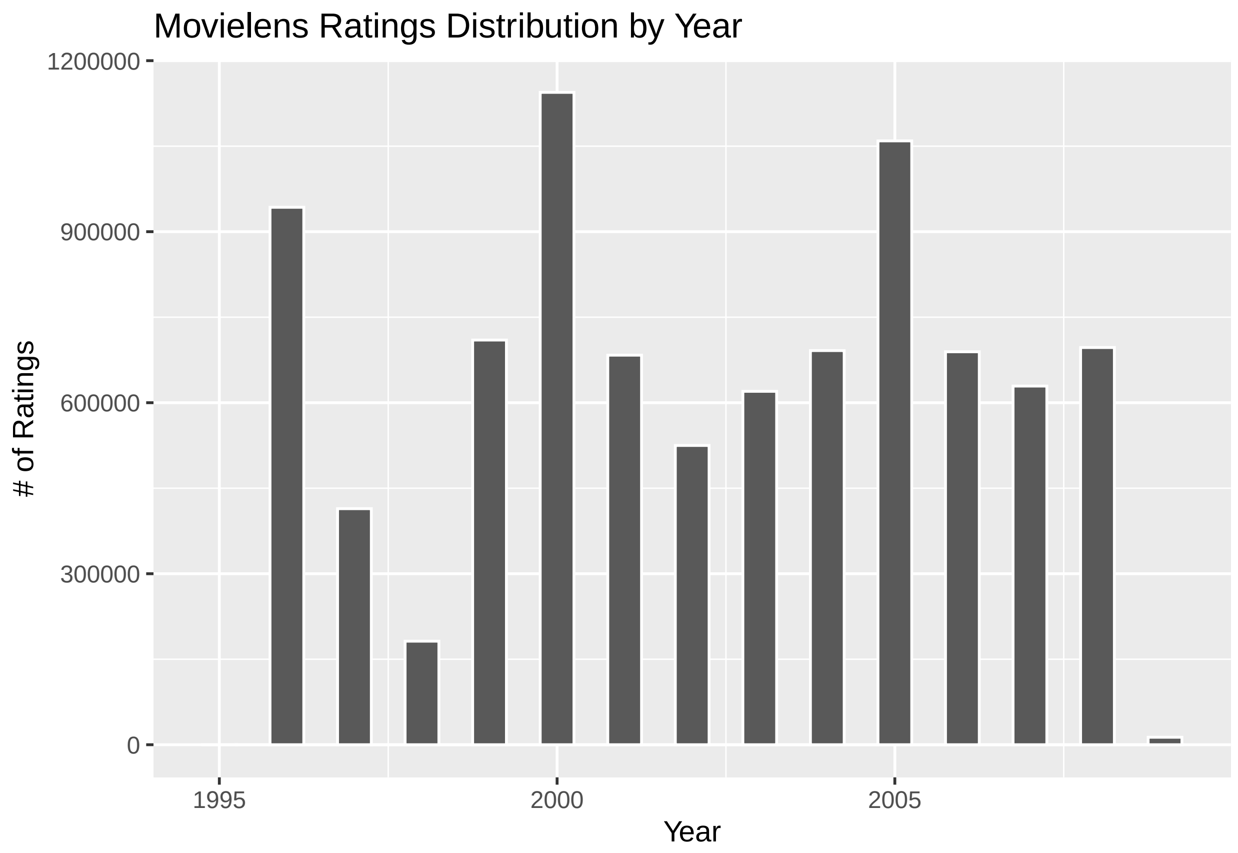
Date

First and foremost, the data collection period is shown below.

```
library(lubridate)
tibble(`Start Date` = date(as_datetime(min(edx$timestamp), origin="1970/01/01")),
       `End Date` = date(as_datetime(max(edx$timestamp), origin="1970/01/01"))) %>%
  mutate(Collection_Period = duration(max(edx$timestamp)-min(edx$timestamp)))
```

```
## # A tibble: 1 x 3
##   `Start Date` `End Date` Collection_Period
##   <date>       <date>     <Duration>
## 1 1995-01-09   2009-01-05 441479727s (~13.99 years)
```

```
edx_timestamp_dist<- edx %>% mutate(year = year(as_datetime(timestamp,origin="1970/01/01"))) %>%
                              ggplot(aes(x=year)) + scale_y_continuous() +
                              geom_histogram(binwidth = 0.50, color = "white") +
                              ggtitle("Movielens Ratings Distribution by Year") +
                              ylab("# of Ratings") + xlab("Year")
edx_timestamp_dist
```



Graph above shows the number of ratings collected during the collection period. Other than spikes in the year 2000 and 2005, not much significance can be drawn from the graph.

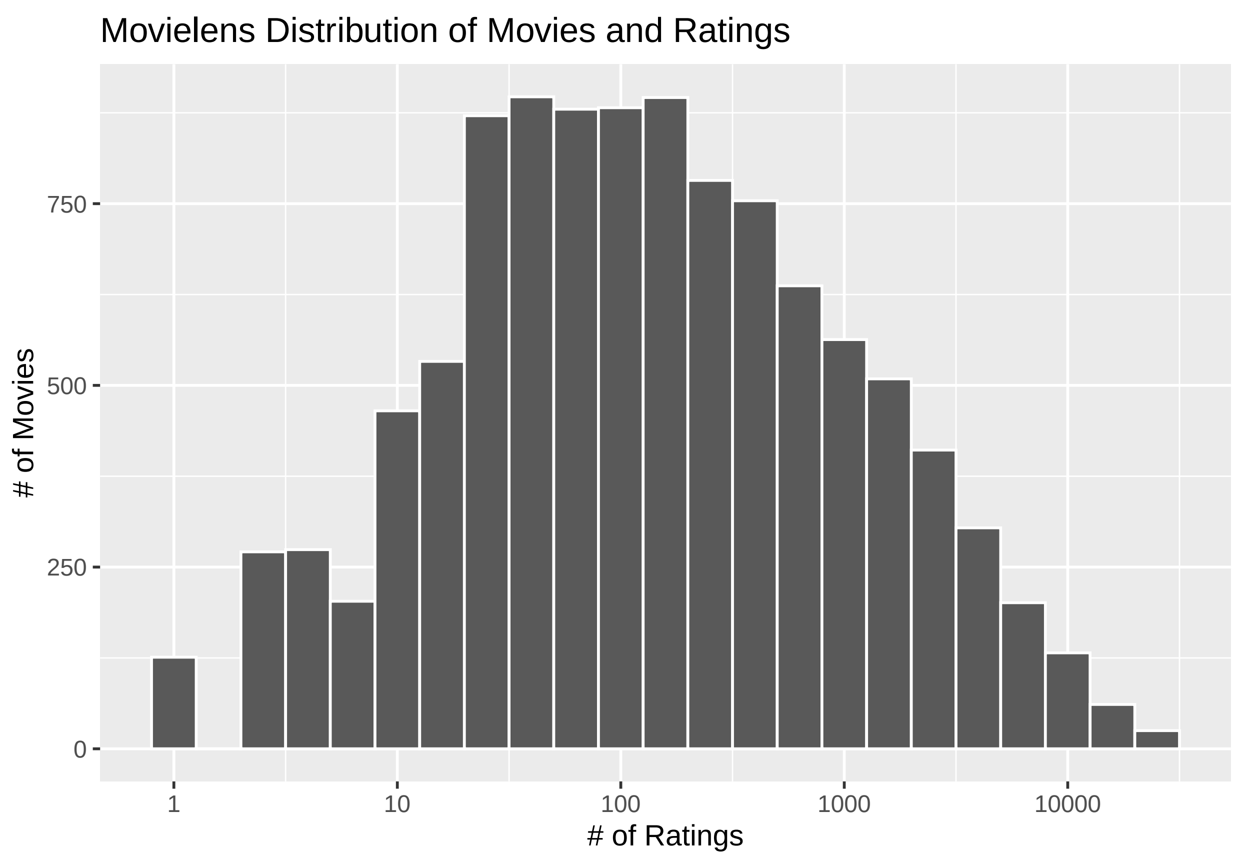
Movies

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

There are a total of 10677 different movies in the dataset. Ideally, all of them would have the same number of ratings. However, that is rarely the case.

```
edx_movie_dist <- edx %>% group_by(movieId) %>%
                          summarize(n=n()) %>%
                          ggplot(aes(n)) + scale_x_log10() +
                          geom_histogram(binwidth = .20, color = "white") +
                          ggtitle("Movielens Distribution of Movies and Ratings") +
                          ylab("# of Movies") + xlab("# of Ratings")
edx_movie_dist
```



Based on the distribution above,it is shown that the distribution is close to symmetric, with some that have a lot of ratings (or vice versa). This is important for later modelling in which we will have to account for weight on movies that have few ratings.

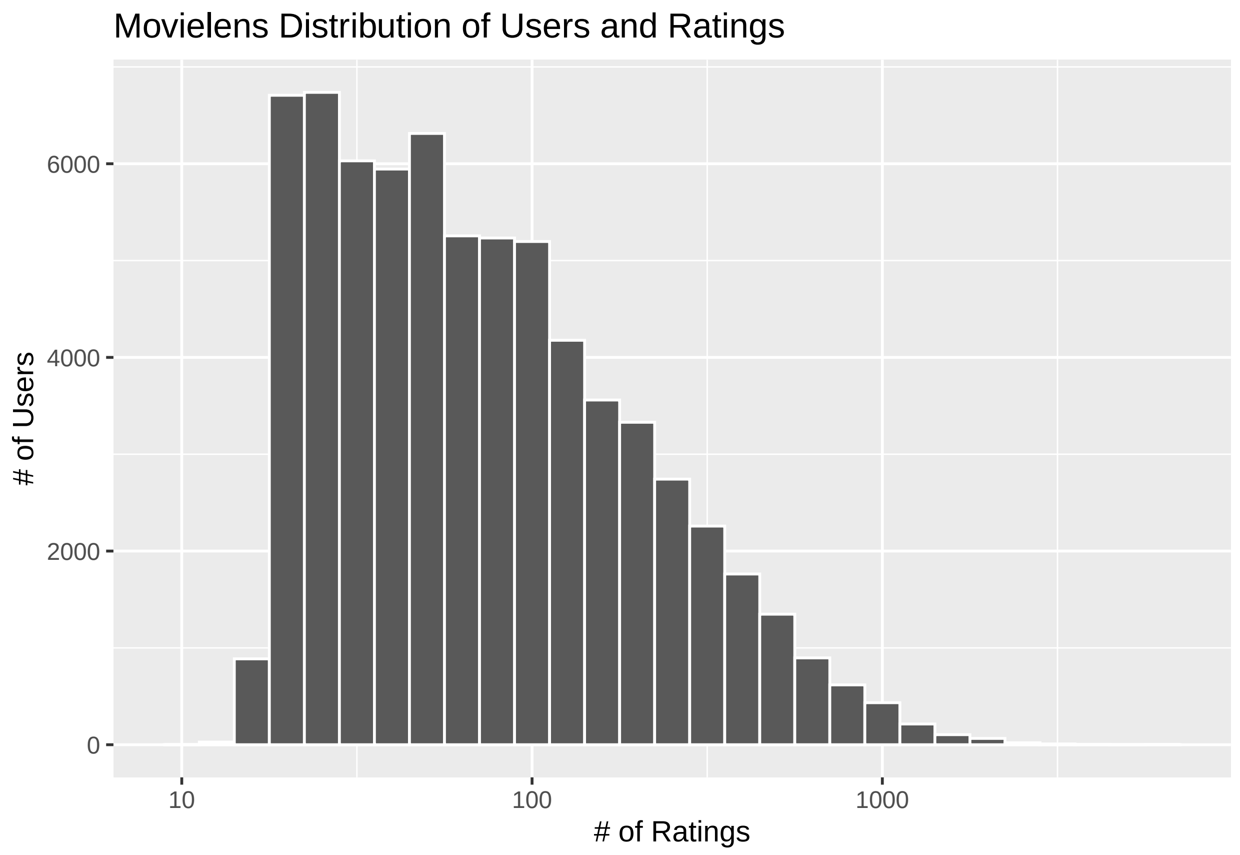
Users

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

There are 69878 unique movie raters in the edx dataset.

```
edx_userid_dist <- edx %>% group_by(userId) %>%
                             summarise(n = n()) %>%
                             ggplot(aes(n)) + scale_x_log10() +
                             geom_histogram(binwidth = 0.10, color = "white") +
                             ggtitle("Movielens Distribution of Users and Ratings") +
                             ylab("# of Users") + xlab("# of Ratings")
edx_userid_dist
```



Based on the distribution above,it is shown that the distribution is rightly skewed, which means that the majority rated few movies. This is important later on as weight needs to be put appropriately on raters that are more active than not.

Ratings

```
n_distinct(edx$rating)
```

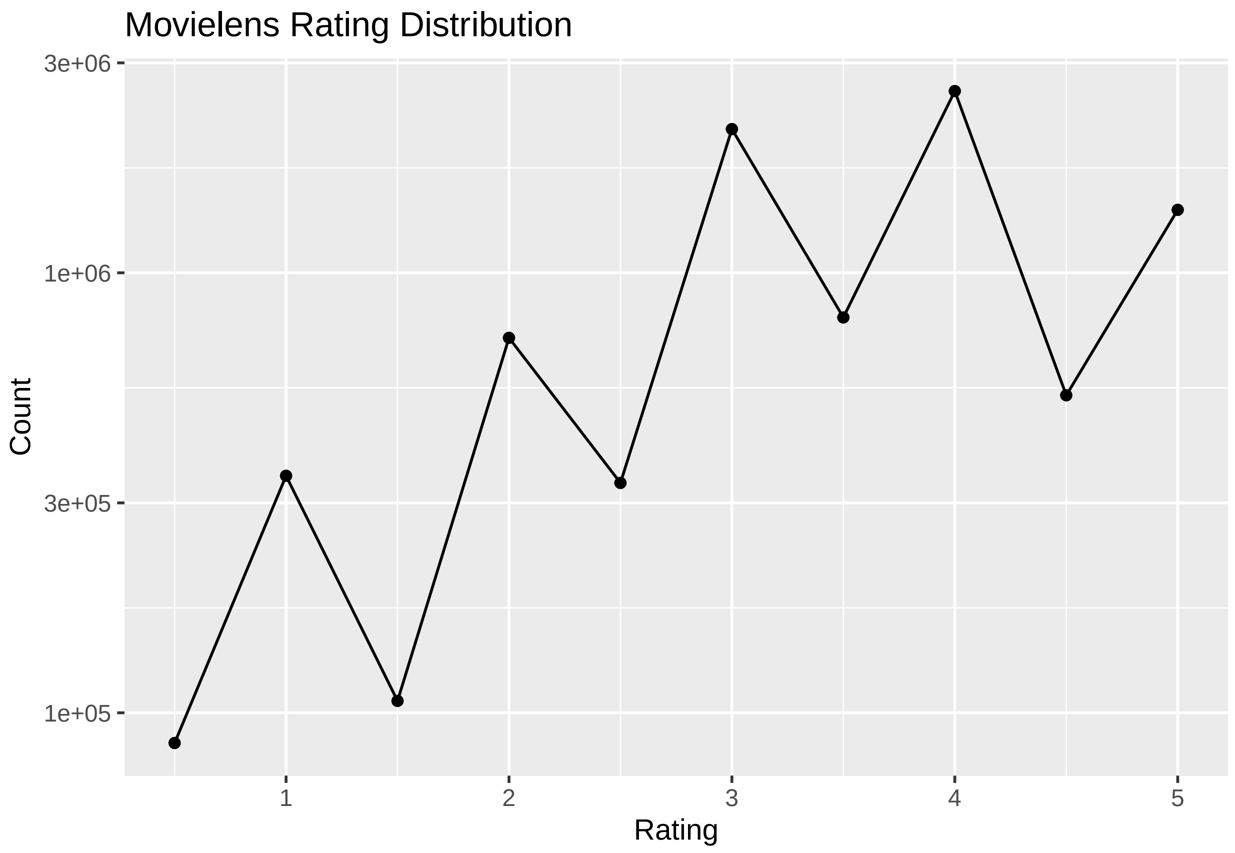
```
## [1] 10
```

```
edx %>% group_by(rating) %>% summarize(n = n())
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
## 10    5   1390114
```

Numbers above show that users(raters) had the option to pick from 10 different possible values, ranging from 0.5 (lowest possible) to 5.0(highest possible).

```r
edx_rating_dist <- edx %>% group_by(rating) %>% summarize(n = n()) %>%
                                                ggplot(aes(x = rating, y = n)) +
                                                geom_point() + geom_line() +
                                                scale_y_log10() +
                                                ggtitle("Movielens Rating Distribution") +
                                                ylab("Count") + xlab("Rating")
edx_rating_dist
```



This graph above shows that rounded values are received more than decimal values. It also shows that higher value ratings are more commonly occurring than lower value ratings.

Genres

```r
n_distinct(edx$genres)
```

```
## [1] 797
```

There are 797 different combination of genres in this dataset. However, genre is not focused on in this project.

In this specific model, we will be focusing specifically on the movie and rater's information to reduce complexity and over training.

```r
train_set<- train_set %>% select(userId, movieId, title, rating)
test_set<- test_set %>% select(userId, movieId, title, rating)
```

Model Preparation

Model Loss Functions

Here we code in the loss functions that we are going to use for evaluation:

```r
#MSE
MSE <- function(true_ratings, predicted_ratings){
  mean((true_ratings - predicted_ratings)^2)
}

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

It is helpful to first determine the MSE in order to see its correlation to the RMSE, which is the value that we will be using.

Random Prediction

The simplest prediction would be to randomly predict using the probability distribution based on the data observation above. This prediction acts as a baseline for improvement since the error value should be the worst it can be.

Since we've created a sample training set and distribution of rating is unknown, we revert back to a simple Monte-Carlo simulation for a gross approximation of the distribution.

```r
set.seed(1234, sample.kind = "Rounding")
prob <- function(x , y) mean(y == x)
ratings <- seq(0.5, 5, 0.5)

#Monte Carlo Simulation
B <- 1000
MC <-replicate(B, {
  sim <- sample(100, replace = TRUE, train_set$rating)
  sapply(ratings, prob, y = sim)
})

probs<- sapply( 1:nrow(MC), function(x) mean(MC[x,]) )

 #Random Prediction
y_random <- sample(ratings, size = nrow(test_set), replace=TRUE, prob = probs)

 #Create tibble table to present errors
results<- tibble()

 #Combine the first random prediction into the table
```

```r
results<- bind_rows(results,
                            tibble(Methods ="Random Prediction",
                                   RMSE = RMSE(test_set$rating,y_random),
                                   MSE = MSE(test_set$rating,y_random)))

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

Methods	RMSE	MSE
Random Prediction	1.828939	3.345019

Note that the results of the RMSE is way above target. This gives us a starting point since random predictions are rarely the right method.

Linear Model

The simplest model uses the concept that all users will give the same ratings to all of the movies, regardless of the user and the movie.

The equation is

$$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$$

and the code is expressed below:

```r
#Only the mean of values
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

```r
#Combine the mean method into the table
results <- bind_rows(results,
                            tibble(Methods = "Mean Only Model",
                                   RMSE = RMSE(test_set$rating,mu),
                                   MSE = MSE(test_set$rating, mu)))

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

Methods	RMSE	MSE
Random Prediction	1.828939	3.345019
Mean Only Model	1.060054	1.123714

Movie Effect

Obviously, different movies are rated differently. This can be simply explained as some movies are more popular than others, such as ones you see in theaters. This is known as the movie effect(bias), and is expressed as b_i.

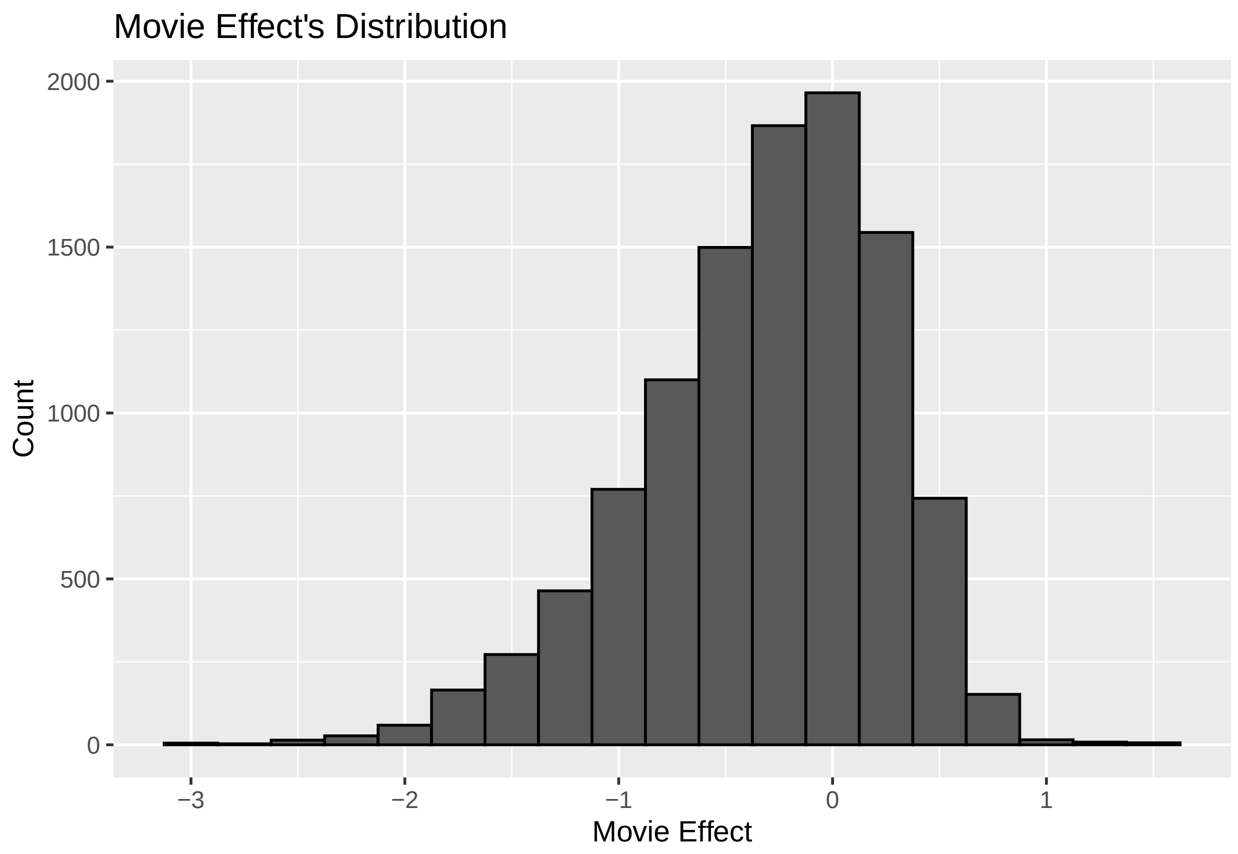
The movie effect can be calculated as the mean of the difference of the mean and observed rating.

The equation is

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$$

and the code is expressed below:

```r
movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
#Plot histogram to show distribution
movie_avgs %>% ggplot(aes(x = b_i)) +
                geom_histogram(binwidth = .25, color =I("black")) +
                ggtitle("Movie Effect's Distribution") +
                ylab("Count") + xlab("Movie Effect") +
                scale_y_continuous()
```



The histogram above shows that the movie effect distribution is skewed to the left.

We can update the prediction as follows:

```r
#Predict the rating with b_i and the mean
yh_b_i <- mu + test_set %>% left_join(movie_avgs, by="movieId") %>% .$b_i
#Calculate the RMSE and input into table
results <- bind_rows(results,
                          tibble(Methods = "Movie Effect Model",
                                 RMSE = RMSE(test_set$rating,yh_b_i),
                                 MSE = MSE(test_set$rating,yh_b_i)))

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

Methods	RMSE	MSE
Random Prediction	1.8289393	3.3450188
Mean Only Model	1.0600537	1.1237138
Movie Effect Model	0.9429615	0.8891764

User Effect

Similarly to the movie effect, different people have different rating patterns or opinions. Some users generously give out high ratings, while other users critically rate and dislike most movies. This, just like the movie effect, is called the user effect(bias), and is expressed as b_u.

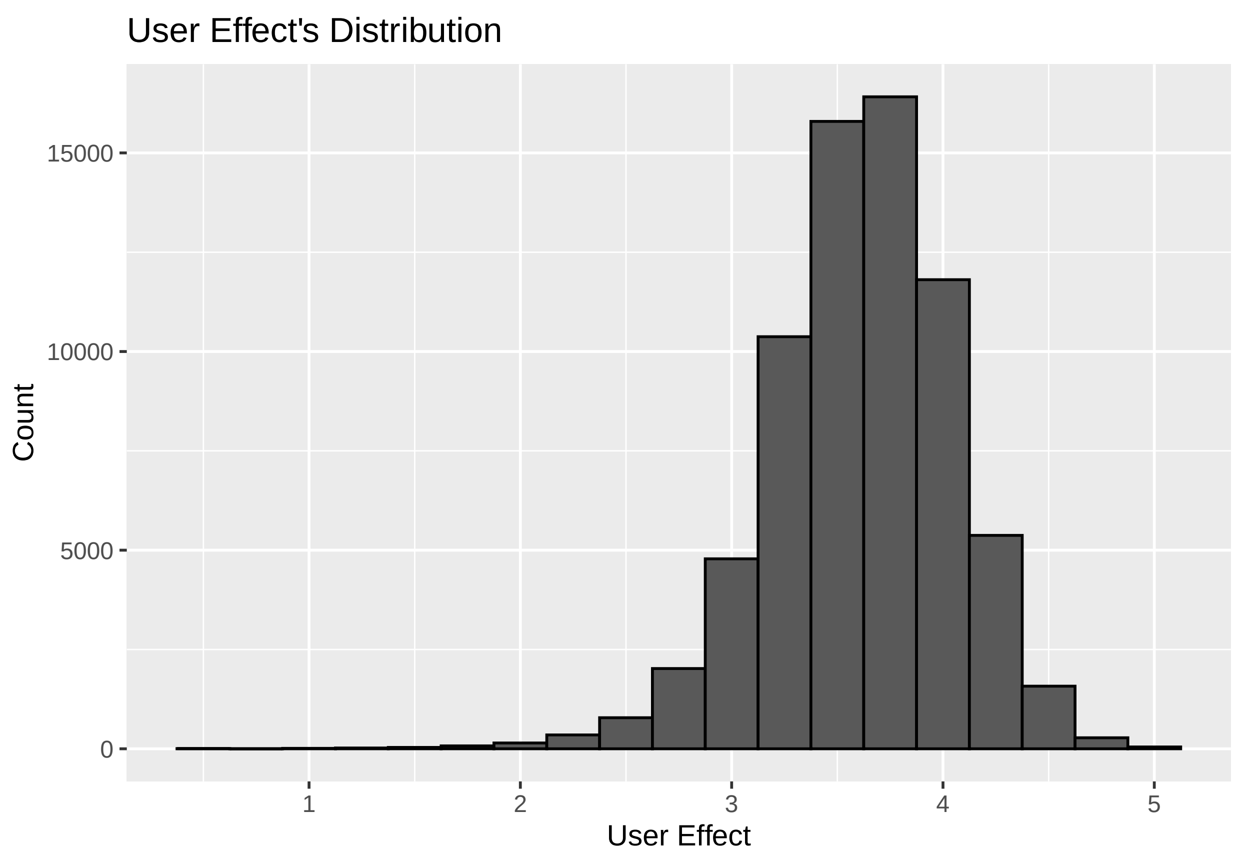
Movies can also be grouped into different genres and have different distributions. Those of the same genre tend to receive similar ratings. However, we will not be covering genre effect in this project.

The equation finalizes to become:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

and the code is shown below:

```r
#User Effect
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#Plot histogram to show distribution
train_set %>% group_by(userId) %>% summarize(b_u = mean(rating)) %>%
                                   ggplot(aes(x = b_u)) +
                                   geom_histogram(binwidth=.25, color = I("black")) +
                                   ggtitle("User Effect's Distribution") +
                                   ylab("Count") + xlab("User Effect") + scale_y_continuous()
```



The histogram above shows us that the user effect is normally distributed. We can now update the prediction again as follows:

```
yh_b_i_u <- test_set %>% left_join(movie_avgs, by = "movieId") %>%
                          left_join(user_avgs, by = "userId") %>%
                          mutate(predictions = mu + b_i + b_u) %>% .$predictions

results<- bind_rows(results,
                            tibble(Methods = "Mean with Movie and User Effect Model",
                                   RMSE = RMSE(test_set$rating, yh_b_i_u),
                                   MSE = MSE(test_set$rating, yh_b_i_u)))

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

Methods	RMSE	MSE
Random Prediction	1.8289393	3.3450188
Mean Only Model	1.0600537	1.1237138
Movie Effect Model	0.9429615	0.8891764
Mean with Movie and User Effect Model	0.8646843	0.7476789

Regularization

In the previous approach, the results give out good estimates. However, they are prone to over fitting due to excessive noise. Noises in data are data that aren't reflective to the data itself, but are from chances.

For example, movies rated very few times should not be weighted the same as movies that have been rated many times. In regularization, we prevent the risk of over fitting by selecting a tuning parameter, lambda, to control the impact of variance and bias in a model.

The equation aims to minimize an equation that adds penalty. Therefore, the movie and user effect can be calculated with:

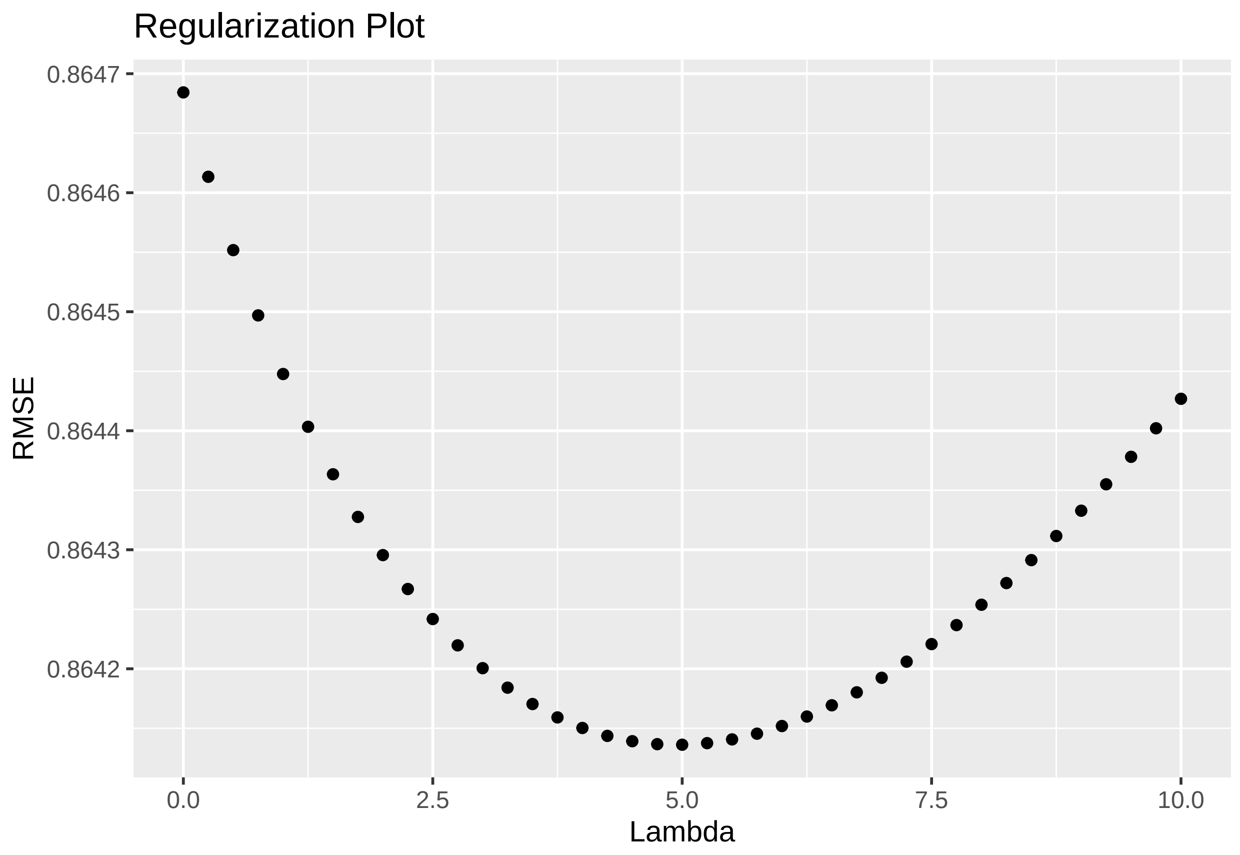
$$\hat{b}_i = \frac{1}{n_i + \lambda} \sum_{u=1}^{n_i} (y_{u,i} - \hat{\mu})$$

$$\hat{b}_u = \frac{1}{n_u + \lambda} \sum_{i=1}^{n_u} (y_{u,i} - \hat{b}_i - \hat{\mu})$$

We must also choose a tuning parameter, lambda, that minimizes the RMSE by running simulations.

Here are the regularized user and movie effect adding lambda in code:

```r
regularization_func <- function(lambda, train, test){

  #mean stays the same as it does not change
  mu<- mean(train$rating)

  #Movie Effect
  b_i <- train %>%  group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n() + lambda))

  #User Effect
  b_u <- train %>% left_join(b_i, by = "movieId") %>%
                           group_by(userId) %>%
                           summarize(b_u = sum( rating - b_i - mu )/(n() + lambda))

  #Prediction
  pred_rate <- test %>% left_join(b_i, by="movieId") %>%
                         left_join(b_u,by="userId") %>%
                         mutate(preds = mu + b_i + b_u) %>%
                         pull(preds)

  return(RMSE(pred_rate, test$rating))
}
  #define lambda
  lambdas <- seq( 0, 10, 0.25 )

  #Tune lambda
  rmses <- sapply(lambdas, regularization_func, train= train_set, test = test_set)

  #Plot lambda vs RMSE to find lowest RMSE penalization
  tibble(Lambda = lambdas, RMSE = rmses) %>% ggplot(aes(Lambda, RMSE)) +
                                              geom_point()+
                                              ggtitle("Regularization Plot")
```



Based on this graph, the lambda with the lowest RMSE value is chosen.

```
#Pick the lowest RMSE lambda
lambda <- lambdas[which.min(rmses)]

#Prediction is then calculated using regularized parameters
mu<- mean(train_set$rating)

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

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

#Prediction and add to table
yh_reg_u_i <- test_set %>% left_join(movie_avgs, by = "movieId") %>%
                          left_join(user_avgs, by = "userId") %>%
                          mutate(pred_rate = mu + b_i + b_u) %>%
                          pull(pred_rate)

results <- bind_rows(results,
                           tibble(Methods = "Regularized User and Movie Effect Model",
                                  RMSE = RMSE(test_set$rating,yh_reg_u_i),
                                  MSE = MSE(test_set$rating,yh_reg_u_i)))
```

```r
results %>% knitr::kable()
```

Methods	RMSE	MSE
Random Prediction	1.8289393	3.3450188
Mean Only Model	1.0600537	1.1237138
Movie Effect Model	0.9429615	0.8891764
Mean with Movie and User Effect Model	0.8646843	0.7476789
Regularized User and Movie Effect Model	0.8641362	0.7467313

Matrix Factorization

As you can see, the regularized user + movie effect model has RMSE reduced to below the 0.8649 target. However, there is one more factor that we did not account for that may reduce RMSE further. The previous model neglects the possible source of variation that groups of movies have similar rating patterns and group of users have similar rating pattern as well.

Matrix factorization converts data into matrices where each movie is a column and rating a cell, then have algorithm fill in missing values. By using approximation of a large rating matrix into 2 lower dimension matrix P and Q.

A R package called recosystem allows us to break down rating matrix and estimate ratings using parallel matrix factorization.

Following code was derived from Recosystem developer's instructions to allow us to set up recosystem.

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

set.seed(1234, sample.kind = "Rounding")

#1. Convert train and test sets to recosystem format
train_dat <- with(train_set, data_memory(user_ind = userId, item_ind = movieId, rating = rating))
test_dat <- with(test_set, data_memory(user_ind = userId, item_ind = movieId, rating = rating))

#2. Create object
r<- Reco()

#3. Select tuning parameter
opts <- r$tune(train_dat, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                      costp_l1 = c(0.01, 0.1),
                                      costq_l1 = c(0.01, 0.1),
                                      nthread = 4, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0.01
##
## $min$costp_l2
## [1] 0.1
##
## $min$costq_l1
## [1] 0.01
##
## $min$costq_l2
## [1] 0.01
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.8108696
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10     0.01     0.01     0.01     0.01   0.1 0.8261381
## 2   20     0.01     0.01     0.01     0.01   0.1 0.8139335
## 3   30     0.01     0.01     0.01     0.01   0.1 0.8121322
## 4   10     0.10     0.01     0.01     0.01   0.1 0.8419334
## 5   20     0.10     0.01     0.01     0.01   0.1 0.8271648
## 6   30     0.10     0.01     0.01     0.01   0.1 0.8312663
## 7   10     0.01     0.10     0.01     0.01   0.1 0.8285348
## 8   20     0.01     0.10     0.01     0.01   0.1 0.8154050
## 9   30     0.01     0.10     0.01     0.01   0.1 0.8108696
## 10  10     0.10     0.10     0.01     0.01   0.1 0.8474870
## 11  20     0.10     0.10     0.01     0.01   0.1 0.8256062
## 12  30     0.10     0.10     0.01     0.01   0.1 0.8319958
## 13  10     0.01     0.01     0.10     0.01   0.1 0.8463959
## 14  20     0.01     0.01     0.10     0.01   0.1 0.8347575
## 15  30     0.01     0.01     0.10     0.01   0.1 0.8391538
## 16  10     0.10     0.01     0.10     0.01   0.1 0.8507334
## 17  20     0.10     0.01     0.10     0.01   0.1 0.8575180
## 18  30     0.10     0.01     0.10     0.01   0.1 0.8701188
## 19  10     0.01     0.10     0.10     0.01   0.1 0.8494099
## 20  20     0.01     0.10     0.10     0.01   0.1 0.8449338
## 21  30     0.01     0.10     0.10     0.01   0.1 0.8539776
## 22  10     0.10     0.10     0.10     0.01   0.1 0.8542065
## 23  20     0.10     0.10     0.10     0.01   0.1 0.8642843
## 24  30     0.10     0.10     0.10     0.01   0.1 0.8731115
## 25  10     0.01     0.01     0.01     0.10   0.1 0.8435415
## 26  20     0.01     0.01     0.01     0.10   0.1 0.8358816
## 27  30     0.01     0.01     0.01     0.10   0.1 0.8282874
## 28  10     0.10     0.01     0.01     0.10   0.1 0.8573742
## 29  20     0.10     0.01     0.01     0.10   0.1 0.8527406
## 30  30     0.10     0.01     0.01     0.10   0.1 0.8517282
## 31  10     0.01     0.10     0.01     0.10   0.1 0.8536045
## 32  20     0.01     0.10     0.01     0.10   0.1 0.8537516
## 33  30     0.01     0.10     0.01     0.10   0.1 0.8521085
## 34  10     0.10     0.10     0.01     0.10   0.1 0.8744586
## 35  20     0.10     0.10     0.01     0.10   0.1 0.8660670
## 36  30     0.10     0.10     0.01     0.10   0.1 0.8645337
## 37  10     0.01     0.01     0.10     0.10   0.1 0.8539527
## 38  20     0.01     0.01     0.10     0.10    0.1 0.8387788
## 39  30     0.01     0.01     0.10     0.10    0.1 0.8360318
## 40  10     0.10     0.01     0.10     0.10    0.1 0.8608190
## 41  20     0.10     0.01     0.10     0.10    0.1 0.8630634
## 42  30     0.10     0.01     0.10     0.10    0.1 0.8734809
## 43  10     0.01     0.10     0.10     0.10    0.1 0.8755090
## 44  20     0.01     0.10     0.10     0.10    0.1 0.8643095
## 45  30     0.01     0.10     0.10     0.10    0.1 0.8576062
## 46  10     0.10     0.10     0.10     0.10    0.1 0.8831152
## 47  20     0.10     0.10     0.10     0.10    0.1 0.8760448
## 48  30     0.10     0.10     0.10     0.10    0.1 0.8817220
## 49  10     0.01     0.01     0.01     0.01    0.2 0.8213605
## 50  20     0.01     0.01     0.01     0.01    0.2 0.8695778
## 51  30     0.01     0.01     0.01     0.01    0.2 1.0151580
## 52  10     0.10     0.01     0.01     0.01    0.2 0.8764137
## 53  20     0.10     0.01     0.01     0.01    0.2 0.8372672
## 54  30     0.10     0.01     0.01     0.01    0.2 0.9367131
## 55  10     0.01     0.10     0.01     0.01    0.2 0.8206283
## 56  20     0.01     0.10     0.01     0.01    0.2 0.9634239
## 57  30     0.01     0.10     0.01     0.01    0.2 0.9638342
## 58  10     0.10     0.10     0.01     0.01    0.2 0.8765078
## 59  20     0.10     0.10     0.01     0.01    0.2 0.8370803
## 60  30     0.10     0.10     0.01     0.01    0.2 0.8905422
## 61  10     0.01     0.01     0.10     0.01    0.2 0.8410340
## 62  20     0.01     0.01     0.10     0.01    0.2 0.8893926
## 63  30     0.01     0.01     0.10     0.01    0.2 0.8470541
## 64  10     0.10     0.01     0.10     0.01    0.2 0.8756632
## 65  20     0.10     0.01     0.10     0.01    0.2 0.8768771
## 66  30     0.10     0.01     0.10     0.01    0.2 0.8836817
## 67  10     0.01     0.10     0.10     0.01    0.2 0.8645609
## 68  20     0.01     0.10     0.10     0.01    0.2 0.8762292
## 69  30     0.01     0.10     0.10     0.01    0.2 0.8805195
## 70  10     0.10     0.10     0.10     0.01    0.2 0.8817403
## 71  20     0.10     0.10     0.10     0.01    0.2 0.8850852
## 72  30     0.10     0.10     0.10     0.01    0.2 0.8850515
## 73  10     0.01     0.01     0.01     0.10    0.2 0.8219170
## 74  20     0.01     0.01     0.01     0.10    0.2 0.8109019
## 75  30     0.01     0.01     0.01     0.10    0.2 0.8584908
## 76  10     0.10     0.01     0.01     0.10    0.2 0.8454007
## 77  20     0.10     0.01     0.01     0.10    0.2 0.8583396
## 78  30     0.10     0.01     0.01     0.10    0.2 0.8721269
## 79  10     0.01     0.10     0.01     0.10    0.2 0.8454316
## 80  20     0.01     0.10     0.01     0.10    0.2 0.8396993
## 81  30     0.01     0.10     0.01     0.10    0.2 0.8401139
## 82  10     0.10     0.10     0.01     0.10    0.2 0.8587497
## 83  20     0.10     0.10     0.01     0.10    0.2 0.8648930
## 84  30     0.10     0.10     0.01     0.10    0.2 0.8711060
## 85  10     0.01     0.01     0.10     0.10    0.2 0.8454676
## 86  20     0.01     0.01     0.10     0.10    0.2 0.8399726
## 87  30     0.01     0.01     0.10     0.10    0.2 0.8405265
## 88  10     0.10     0.01     0.10     0.10    0.2 0.8696853
## 89  20     0.10     0.01     0.10     0.10    0.2 0.8768852
## 90  30     0.10     0.01     0.10     0.10    0.2 0.8771707
## 91  10     0.01     0.10     0.10     0.10    0.2 0.8622002
## 92  20     0.01     0.10     0.10     0.10    0.2 0.8738768
## 93  30     0.01     0.10     0.10     0.10    0.2 0.8824440
## 94  10     0.10     0.10     0.10     0.10    0.2 0.8893485
## 95  20     0.10     0.10     0.10     0.10    0.2 0.8928593
## 96  30     0.10     0.10     0.10     0.10    0.2 0.8927203
```

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

```
## iter      tr_rmse          obj
##    0       0.9619   1.1294e+07
##    1       0.8729   1.0018e+07
##    2       0.8455   9.6739e+06
##    3       0.8258   9.4464e+06
##    4       0.8092   9.2686e+06
##    5       0.7957   9.1299e+06
##    6       0.7842   9.0163e+06
##    7       0.7745   8.9263e+06
##    8       0.7658   8.8463e+06
##    9       0.7583   8.7787e+06
##   10       0.7516   8.7191e+06
##   11       0.7458   8.6699e+06
##   12       0.7405   8.6251e+06
##   13       0.7358   8.5851e+06
##   14       0.7315   8.5486e+06
##   15       0.7278   8.5178e+06
##   16       0.7244   8.4881e+06
##   17       0.7214   8.4632e+06
##   18       0.7186   8.4380e+06
##   19       0.7162   8.4169e+06
```

```
#Prediction and add to table
yh_r<- r$predict(test_dat, out_memory())

head(yh_r,10)
```

```
##  [1] 4.977966 3.794279 3.685266 2.782787 3.535987 3.938345 3.715642 3.405542
##  [9] 3.740240 3.325115
```

```
results <- bind_rows(results,
                          tibble(Methods = "Matrix Factorization via Recosystem Method",
                     RMSE = RMSE(test_set$rating, yh_r),
                     MSE = MSE(test_set$rating,yh_r)))

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

Methods	RMSE	MSE
Random Prediction	1.8289393	3.3450188
Mean Only Model	1.0600537	1.1237138
Movie Effect Model	0.9429615	0.8891764
Mean with Movie and User Effect Model	0.8646843	0.7476789
Regularized User and Movie Effect Model	0.8641362	0.7467313
Matrix Factorization via Recosystem Method	0.7935970	0.6297962

Validation Process

As shown in the results table, regularization with user and movie effect achieved the target, but recosystem matrix factorization was the best of them all. Therefore, we can train the original edx dataset and calculate RMSE with validation dataset.

```r
#mean only
edx_mu <- mean(edx$rating)

# b_i
edx_mov_avgs <- edx %>% group_by(movieId) %>%
                        summarize(b_i = sum(rating - edx_mu)/(n() + lambda))

# b_u
edx_use_avgs <- edx %>% left_join(edx_mov_avgs, by="movieId") %>%
                        group_by(userId) %>%
                        summarize(b_u = sum(rating - b_i - edx_mu)/(n() + lambda))

# Predictions and add to table
edx_yh_rate <- validation %>% left_join(edx_mov_avgs, by = "movieId") %>%
                              left_join(edx_use_avgs, by = "userId") %>%
                              mutate(pred_rate = edx_mu + b_i + b_u) %>%
                              pull(pred_rate)

results <- bind_rows(results,
                            tibble(Methods = "Final Regularization Model",
                            RMSE = RMSE(validation$rating, edx_yh_rate),
                            MSE  = MSE(validation$rating, edx_yh_rate)))

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

Methods	RMSE	MSE
Random Prediction	1.8289393	3.3450188
Mean Only Model	1.0600537	1.1237138
Movie Effect Model	0.9429615	0.8891764
Mean with Movie and User Effect Model	0.8646843	0.7476789
Regularized User and Movie Effect Model	0.8641362	0.7467313
Matrix Factorization via Recosystem Method	0.7935970	0.6297962
Final Regularization Model	0.8648177	0.7479097

The RMSE calculated with regularization on linear model resulted in 0.8648177, which is lower than the target of 0.8649 and slightly higher than the test set.

Next we validate with matrix factorization.

```r
set.seed(1234, sample.kind="Rounding")

edx_r <- with(edx, data_memory(user_ind = userId,
                               item_ind = movieId,
                               rating = rating))

validation_r <- with(validation, data_memory(user_ind = userId,
                                              item_ind = movieId,
                                              rating = rating))

r<- Reco()

opts <- r$tune(edx_r, 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))

r$train(edx_r,opts = c(opts$min, nthread=4, niter=20))
```

```
## iter      tr_rmse          obj
##    0       0.9713   1.1983e+07
##    1       0.8712   9.8705e+06
##    2       0.8380   9.1636e+06
##    3       0.8166   8.7525e+06
##    4       0.8006   8.4678e+06
##    5       0.7882   8.2637e+06
##    6       0.7783   8.1140e+06
##    7       0.7703   7.9922e+06
##    8       0.7635   7.8950e+06
##    9       0.7577   7.8167e+06
##   10       0.7527   7.7494e+06
##   11       0.7483   7.6933e+06
##   12       0.7444   7.6442e+06
##   13       0.7408   7.6027e+06
##   14       0.7375   7.5637e+06
##   15       0.7345   7.5313e+06
##   16       0.7317   7.5023e+06
##   17       0.7291   7.4755e+06
##   18       0.7268   7.4505e+06
##   19       0.7246   7.4285e+06
```

```
#Prediction and add to table
yh_f_r<- r$predict(validation_r, out_memory())



results <- bind_rows(results,
                            tibble(Methods = "Final Matrix Factorization via Recosystem",
                            RMSE = RMSE(validation$rating, yh_f_r),
                            MSE = MSE(validation$rating,yh_f_r)))

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

Methods	RMSE	MSE
Random Prediction	1.8289393	3.3450188
Mean Only Model	1.0600537	1.1237138
Movie Effect Model	0.9429615	0.8891764
Mean with Movie and User Effect Model	0.8646843	0.7476789

Methods	RMSE	MSE
Regularized User and Movie Effect Model	0.8641362	0.7467313
Matrix Factorization via Recosystem Method	0.7935970	0.6297962
Final Regularization Model	0.8648177	0.7479097
Final Matrix Factorization via Recosystem	0.7830395	0.6131509

The final RMSE validation for the matrix factorization via recosystem resulted in 0.7827885, which was a better outcome than the final regularization model (0.8648).

Conclusion

Throughout this entire project, we have learned to apply data exploration and data analysis on the Movielens dataset. We have also learned to create models, understand its pros and cons, and generate a RMSE error based on its predictions. We have expanded from the worst model of random predictions that yielded RMSE of 1.8721 to include in movie and user effect as well as regularization and matrix factorization to achieve the final model of 0.78 RMSE, successfully passing the 0.8649 target.

Some limitations that have popped up during the project is the inability for regular household computers to process some of the models. Some of the models and graphs in this project also took up more than a few minutes to process.

This may become an issue when new data is added in, since the entire code would have to run every time new data gets entered in, causing potential delay in the system.

In terms of future work, there is a lot more to be implemented. Genre, for example, was chosen to be left out of the modelling. However, in the future, genre could be implemented to produce predictions that has lower RMSE.