Ivan Malig
Movielens Project EDX
June 6, 2020

Overview:

The project is similar to the Netflix Challenge introduced in the last part of the EDX Data Science course. The goal is to create a code that would have the ability to accurately predict movie ratings in a created "validation" set. The project contains several parts with each part giving ample information on what is happening with the code and the results. The challenge here is to try and figure out different factors that may result in variations in movie ratings and at the same time, incorporate these factors into our model in order to be more accurate in terms of our prediction. The dataset used was called movielens and it was provided by the EDX course. The process was to split the data into training and tests. We then perform operations that will help us get estimates for our different "predictors" and use these to make prediction ratings in our "validation" set.

A. Data

The code below is a course-provided code that will allow us to have the data set which we will be working with.

```
#Create test and validation sets
# Create edx set, validation set, and submission file
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")
# 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 <- read.table(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(levels(movieId))[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)
```

The following libraries will be useful in our analysis

library(caret)
library(anytime)
library(tidyverse)
library(lubridate)
library(ggplot2)

#We want to incerase our memory size since our current memory will prohibit us from smoothly running some snips of code later on.

memory.limit(size= 10000)

#We keep all columns except for ratings in our validation set.

validation_a <- validation
validation <- validation %>% select(-ratings)

B. The EDX DataSet

The edx data set contains 9000055 rows and 6 columns of information about different movies such as "userId"   "movieId" "rating"   "timestamp" "title"   "genres".

"userId" - a unique code given to each user.
"movieId" - a unique code given to each movie.
"rating" - rating given by users ranging from 0.5 to 5 (in increments of 0.5)
"genres" - film categories
"title" - title of the movie
"timestamp"- time when ratings were given.

Below is provided some of the information and summary statistics within the said DataSet

```
> head(edx)
  userId movieId rating timestamp                         title                        genres
1      1     122      5 838985046              Boomerang (1992)                Comedy|Romance
2      1     185      5 838983525               Net, The (1995)         Action|Crime|Thriller
4      1     292      5 838983421               Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
5      1     316      5 838983392               Stargate (1994)       Action|Adventure|Sci-Fi
6      1     329      5 838983392 Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi
7      1     355      5 838984474       Flintstones, The (1994)       Children|Comedy|Fantasy
```

```
> summary(edx)
     userId         movieId          rating        timestamp            title              genres         
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08   Length:9000055     Length:9000055    
 1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08   Class :character   Class :character  
 Median :35738   Median : 1834   Median :4.000   Median :1.035e+09   Mode  :character   Mode  :character  
 Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09                                        
 3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09                                        
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09                                        
```

2. Data Processing

For this part, our goal is to tweak the EDX set to make it easier to make use of by doing the following steps

 # First, we will define a function that will measure our RMSE.

```
RMSE <- function(actual_ratings, predicting){
  sqrt(mean((actual_ratings-predicting)^2,na.rm=TRUE))
}
```

 # Second, we will create a separate column for year for our datasets.

```
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation_a <- validation_a %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

 # Third, we will try to separate the genres column of our datasets so each row will only have one genre.

```
sep_edx  <- edx  %>% separate_rows(genres, sep = "\\|")
sep_validation <- validation   %>% separate_rows(genres, sep = "\\|")
sep_validation_a <- validation_a  %>% separate_rows(genres, sep = "\\|")
```

Shown below are the header and summary statistics of EDX DataSet after the minor adjustments

```
> head(edx)
  userId movieId rating timestamp                         title                        genres year
1      1     122      5 838985046              Boomerang (1992)                Comedy|Romance 1992
2      1     185      5 838983525               Net, The (1995)         Action|Crime|Thriller 1995
3      1     292      5 838983421               Outbreak (1995)  Action|Drama|Sci-Fi|Thriller 1995
4      1     316      5 838983392               Stargate (1994)       Action|Adventure|Sci-Fi 1994
5      1     329      5 838983392 Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi 1994
6      1     355      5 838984474       Flintstones, The (1994)       Children|Comedy|Fantasy 1994
```

```
> head(sep_edx)
# A tibble: 6 x 7
  userId movieId rating timestamp title                   genres   year
   <int>   <dbl>  <dbl>     <int> <chr>                   <chr>   <dbl>
1      1     122      5 838985046 Boomerang (1992)        Comedy   1992
2      1     122      5 838985046 Boomerang (1992)        Romance  1992
3      1     185      5 838983525 Net, The (1995)         Action   1995
4      1     185      5 838983525 Net, The (1995)         Crime    1995
5      1     185      5 838983525 Net, The (1995)         Thriller 1995
6      1     292      5 838983421 Outbreak (1995)         Action   1995
```

By running this code below, we can see how many **unique** observations there are

```
> edx %>% summarize(unique_users = n_distinct(userId), unique_movies = n_distinct(movieId))
  unique_users unique_movies
1        69878         10677
```

#We can see here which years had the highest ratings count. We notice that the 1990s, specifically the mid 90s, had the highest number of ratings. The peak was at 1995.

rate_per_year<- sep_edx %>% group_by(year) %>% summarize(count= n()) %>% arrange(desc(count))

```
    year   count
   <dbl>   <int>
 1  1995 2084327
 2  1994 1732933
 3  1996 1560847
 4  1999 1159558
 5  1997 1137289
 6  1993 1086018
 7  1998 1085810
 8  2000  930516
 9  2001  832266
10  2002  710487
```

#We can also see which genres are most rated by running this code

rate_per_genre <- sep_edx%>% group_by(genres) %>% summarize(count = n()) %>% arrange(desc(count))

```
> rate_per_genre <- sep_edx%>% group_by(genres) %>% summarize(count = n()) %>% arrange(desc(count))
`summarise()` ungrouping output (override with `.groups` argument)
> rate_per_genre
# A tibble: 20 x 2
   genres          count
   <chr>           <int>
 1 Drama         3910127
 2 Comedy        3540930
 3 Action        2560545
 4 Thriller      2325899
 5 Adventure     1908892
 6 Romance       1712100
 7 Sci-Fi        1341183
 8 Crime         1327715
 9 Fantasy        925637
10 Children       737994
```

#As mentioned above, ratings can range from 0.5to 5 and ratings are in increments of 0.5 as seen here.

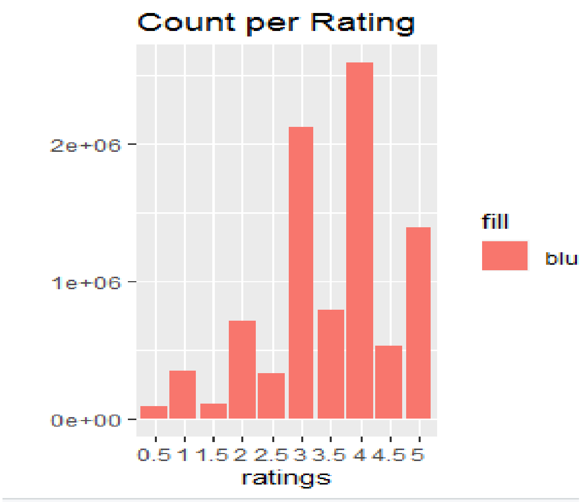
ratings <- as.vector(edx$rating)
unique(ratings)

```
> ratings <- as.vector(edx$rating)
> unique(ratings)
 [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

#Chart showing how many cunt there are for each possible rating value.

ratings <- ratings[ratings != 0]
ratings <- factor(ratings)
qplot(ratings, fill= "blue") +
ggtitle("Count per Rating")

#From the graph we wil notice that whole ratings are more common compared to "half" ratings, furthermore a rating of 4 is more common than any other whole rating

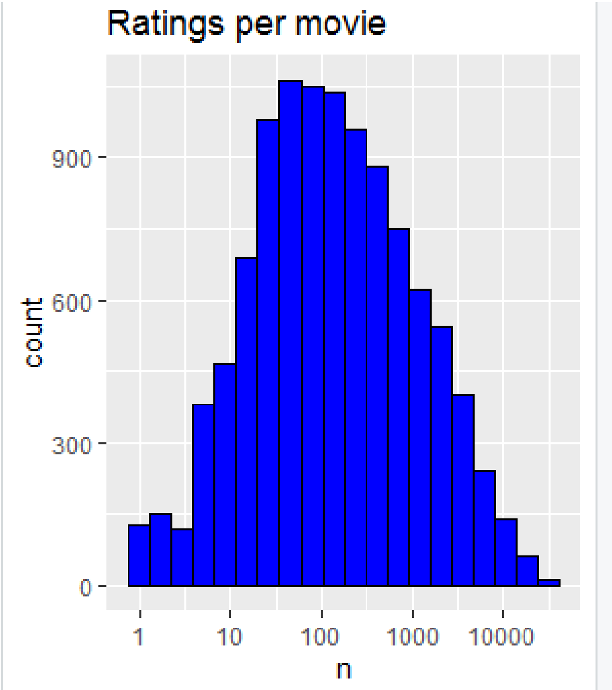


3. Analysis Strategy

#We must keep in mind that movie ratings will vary across people, movies, and genres (among other things). User preferences may result in a bad rating on a "good" movie or a good rating on a "bad" movie. On the other hand, more known movies will tend to average higher ratings than less known movies. We can also see fluctuating ratings based on movie genre. We will try to incorporate these biases into our model in order to have a reasonable prediction.

#We see here the distribution of ratings per movie

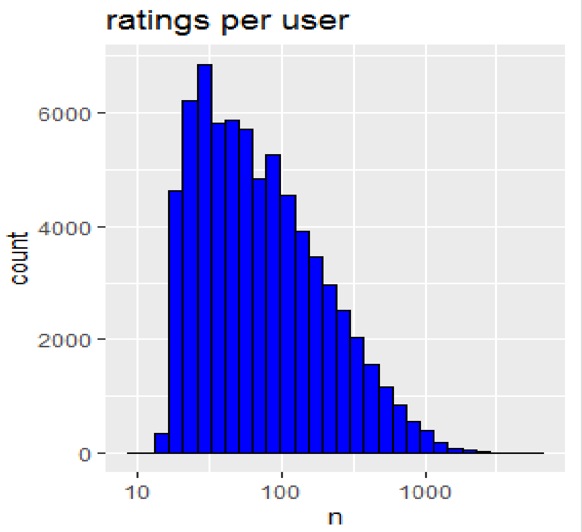
edx %>% count(movieId) %>% ggplot(aes(n)) + geom_histogram(bins = 20, color = "black", fill = "blue") + scale_x_log10() + ggtitle("Ratings per movie")



From this graph, it is noticeable how there are certain movies that receive minimal ratings, the reason for this could be generalized as a sort of "movie" effect where some movies may not be as good or as appealing to the masses and therefore tend to not receive as much ratings as other popular movies.

#Here, we see ratings per user

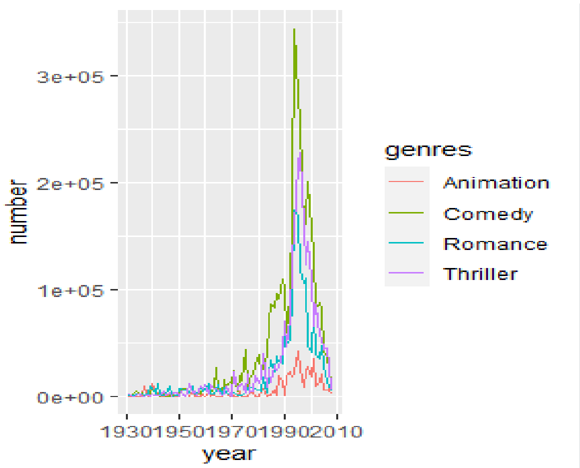
edx %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(bins = 30, color = "black", fill = "blue") + scale_x_log10() + ggtitle("ratings per user")



Looking at the graph, we can see how there are numerous users who rated very few movies. This could be caused by a number of different factors or it may simply be a case of some people who do not like to rate things (some would even rate randomly just to have given a rating).

#Here, we visualize how genre popularity changed throughout the years by looking at the trend for selected genres

```
popular_genres <- sep_edx %>%
 na.omit() %>% select(movieId, year, genres) %>% mutate(genres = as.factor(genres))
%>%   group_by(year, genres) %>% summarise(number = n()) %>%
complete(year = full_seq(year, 1), genres, fill = list(number = 0))

popular_genres %>%
filter(year > 1930) %>% filter(genres %in% c("Comedy", "Thriller", "Animation",
"Romance")) %>% ggplot(aes(x = year, y = number)) + geom_line(aes(color=genres)) +
scale_fill_brewer(palette = "Paired")
```



4. Preparing the Model

#The goal is to compare RSME for different predicting models, we will keep track of our RSMEs with this code

rmse_tracker <- data_frame()

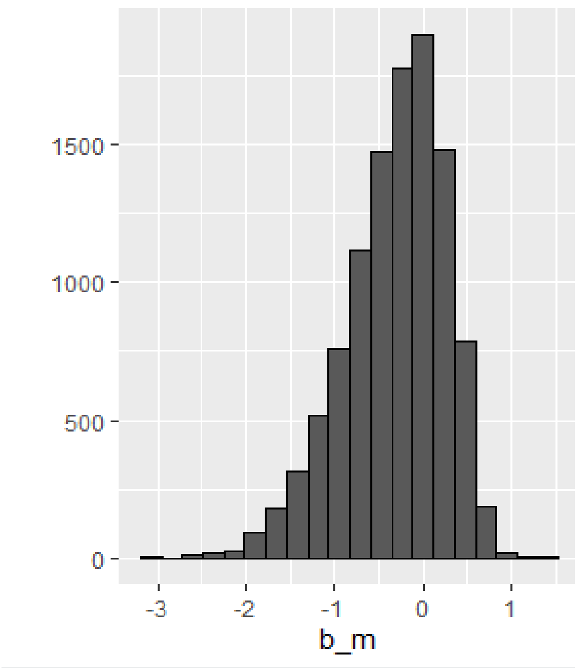
4a. #the simplest model we can use for prediction is with the mean rating, meaning were are to use the mean as our predicted rating for al movies.

mu <- mean(edx$rating)

```
mu <- mean(edx$rating)
mu
1] 3.512465
```

4b. #Our next model will take into account the movie bias, that is different movies (e.g blockbuster vs indie) May results in stark differences in rating.

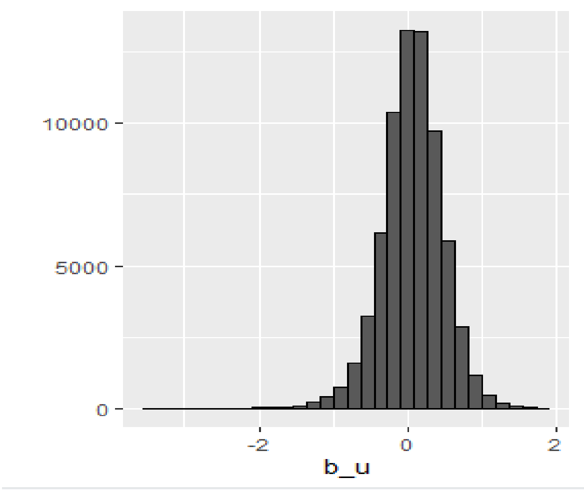
beta_m <- edx %>%
group_by(movieId) %>% summarize(b_m = mean(rating - mu))
beta_m %>% qplot(b_m, geom ="histogram", bins = 20, data = ., color = I("black"))



We can notice how there are movies that were rated very little times; something to look out for in the analysis.

4c. #Next, we stated that different viewers have different tendencies in rating movies. Thus, like in our movie case, we are going to compute for our user bias by running this code below

beta_u<- edx %>% left_join(beta_m, by='movieId') %>% group_by(userId) %>% summarize(b_u = mean(rating - mu - b_m))
beta_u %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))

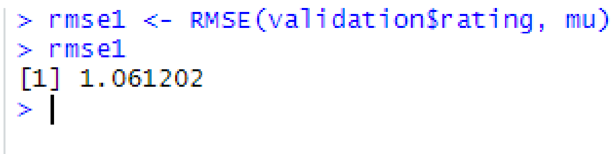


We see that not every user rates as much as the other, another thing to note out for later in our analysis.

5. The model
#The goal is for our model to produce a low(er) rsme. We will try to examine different resulting RSMEs when accounting for the different biases that we've talked about

#this will be our code for the base model.

rmse1 <- RMSE(validation$rating, mu)



#to check our current tracker

rmse_tracker <- data_frame(method = "Mean", RMSE = rmse1)
rmse_tracker

```
> rmse_tracker
# A tibble: 1 x 2
  method   RMSE
  <chr>   <dbl>
1 Mean     1.06
```

#with movie efect

rmse2 <- validation %>% left_join(beta_m, by='movieId') %>% mutate(pred = mu + b_m)
model1 <- RMSE(validation_a$rating,rmse2$pred)
rmse_tracker <- bind_rows(rmse_tracker, data_frame(method="Mean + Beta_m", RMSE = model1))
rmse_tracker

```
> rmse_tracker <- bind_rows(rmse_tracker, data_frame(method="Mean + Beta_m", RMSE = model1 ))
> rmse_tracker
# A tibble: 2 x 2
  method          RMSE
  <chr>          <dbl>
1 Mean           1.06
2 Mean + Beta_m 0.944
> 
```

#here we combine the movie and user effect on our model. Notice how we came up with a lower rmse after incorporating one factor in our model.

rmse3 <- validation %>% left_join(beta_m, by='movieId') %>% left_join(beta_u, by='userId') %>% mutate(pred = mu + b_m + b_u)

 # update rmse results

model2 <- RMSE(validation_a$rating,rmse3$pred)
rmse_tracker <- bind_rows(rmse_tracker,data_frame(method="Mean + b_m + b_u", RMSE = model2))
rmse_tracker

```
> rmse_tracker <- bind_rows(rmse_tracker,data_frame(method="Mean + b_m + b_u",  RMSE = model2))
> rmse_tracker
# A tibble: 3 x 2
  method             RMSE
  <chr>             <dbl>
1 Mean              1.06
2 Mean + Beta_m     0.944
3 Mean + b_m + b_u 0.865
```

Again, notice how we get a better (lower) rmse value after another factor was incorporated.

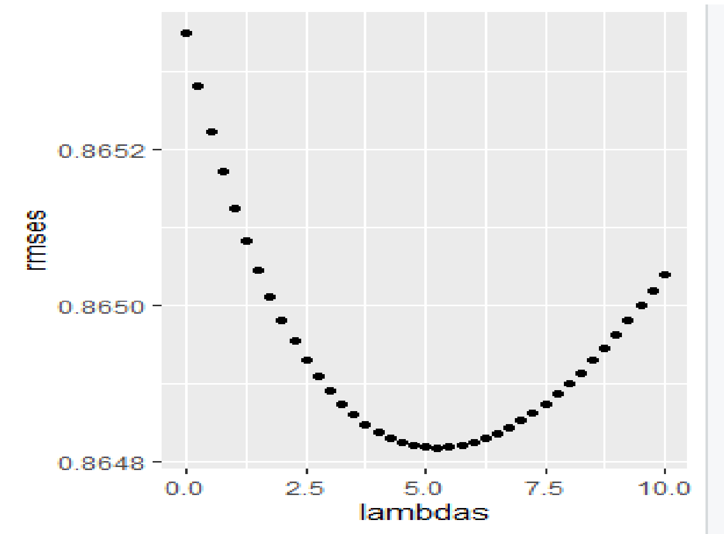
6. Regularisation

#We saw a while ago that there are many "outliers" in our data. These may be users who rated rarely, or movies that were rarely given a rating. TO make a better prediction, we must be able to incorporate there into our model by giving them a reduced impact on our model so to say. We make use of a tuning parameter in this case, lambda. 'We'll use cross validation to find our perfect lambda. For each lambda, we will compute the corresponding b_m and b_u.

```
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){

mu <- mean(edx$rating)

b_m <- edx %>% group_by(movieId) %>%summarize(b_m = sum(rating - mu)/(n()+l))

b_u <- edx %>% left_join(b_m, by="movieId") %>% group_by(userId) %>%
summarize(b_u = sum(rating - b_m - mu)/(n()+l))

predicting <- validation %>% left_join(b_m, by = "movieId") %>% left_join(b_u, by =
"userId") %>% mutate(pred = mu + b_m + b_u) %>% .$pred

return(RMSE(validation_a$rating,predicting))
})
```

#To show an rmse-lambda plot that will help us visualize what value of lambda will be optimal

```
qplot(lambdas, rmses)
```



#To check which value of lambda will give us the minimum rmse

lambda <- lambdas[which.min(rmses)]
lambda

```
> lambda <- lambdas[which.min(rmses)]
> lambda
[1] 5.5
>
```

```
> rmses
 [1] 0.8655329 0.8654680 0.8654111 0.8653602 0.8653141 0.8652723 0.8652344 0.8652000 0.8651688 0.8651406
[11] 0.8651153 0.8650927 0.8650725 0.8650548 0.8650394 0.8650261 0.8650149 0.8650056 0.8649982 0.8649926
[21] 0.8649887 0.8649864 0.8649857 0.8649865 0.8649888 0.8649924 0.8649974 0.8650036 0.8650111 0.8650197
[31] 0.8650294 0.8650403 0.8650522 0.8650651 0.8650789 0.8650937 0.8651094 0.8651260 0.8651434 0.8651616
[41] 0.8651806
```

#I ran the code multiple times because I was getting mixed lambdas of 5.25 and 5.5. I decided to just run it again with different increments (0.25 and 0.5) just to see which would result in the "correct"lambda.

#Next, we look for regularised values for b_m and b_u for our given lambda value.

regular_beta_m<- edx %>% group_by(movieId) %>% summarize(b_m = sum(rating - mu)/(n()+lambda), n_i = n())

 # Compute regularized estimates of b_u using our lambda

regular_beta_u <- edx %>% left_join(regular_beta_m, by='movieId') %>% group_by(userId) %>% summarize(b_u = sum(rating - mu - b_m)/(n()+lambda), n_u = n())

 # Predict ratings

regular_predicting <- validation %>% left_join(regular_beta_m, by='movieId') %>% left_join(regular_beta_u, by='userId') %>% mutate(pred = mu + b_m + b_u) %>% .$pred

 # Update results

model3 <- RMSE(validation_a$rating,regular_predicting)
rmse_tracker <- bind_rows(rmse_tracker, data_frame(method="Beta_m and Beta_u (regularized)", RMSE = model3))
rmse_tracker

```
> rmse_tracker
# A tibble: 4 x 2
  method                          RMSE
  <chr>                          <dbl>
1 Mean                           1.06
2 Mean + Beta_m                  0.944
3 Mean + b_m + b_u               0.865
4 Beta_m and Beta_u (regularized) 0.865
> |
```

#Regularisation with the other effects

#This code below will help us choose the optimal lambda that will result in the lowest rmse we can obtain in our model. However, running the code leads to my laptop freezing which then would force me to restart and reload everything. The solution I thought of was to choose a few "smart" random choices of lambda instead of running this code as to prevent freezing.

```
lambdas <- seq(0, 30, 1)

rmses <- sapply(lambdas, function(l){

mu <- mean(edx$rating)

b_m <- sep_edx %>% group_by(movieId) %>% summarize(b_m = sum(rating -
mu)/(n()+l))

b_u <- sep_edx %>% left_join(b_m, by="movieId") %>%group_by(userId) %>%
summarize(b_u = sum(rating - b_m - mu)/(n()+l))

b_y <- sep_edx %>% left_join(b_m, by='movieId') %>% left_join(b_u, by='userId')
%>% group_by(year) %>% summarize(b_y = sum(rating - mu - b_m -
b_u)/(n()+lambda), n_y = n())

b_g <- sep_edx %>% left_join(b_m, by='movieId') %>%
left_join(b_u, by='userId') %>% left_join(b_y, by = 'year') %>%
group_by(genres) %>% summarize(b_g = sum(rating - mu - b_m - b_u -
b_y)/(n()+lambda), n_g = n())

predicting <- sep_validation %>% left_join(b_m, by='movieId') %>%
left_join(b_u, by='userId') %>% left_join(b_y, by = 'year') %>%
left_join(b_g, by = 'genres') %>% mutate(pred = mu + b_m + b_u + b_y + b_g) %>%
.$pred

return(RMSE(sep_validation_a$rating,predicting))
})

qplot(lambdas, rmses)
```

```
regular_beta_m2 <- sep_edx %>% group_by(movieId) %>%
summarize(b_m = sum(rating - mu)/(n()+lambda_opt), n_i = n())

regular_beta_u2 <- sep_edx %>% left_join(regular_beta_m2, by='movieId') %>%
group_by(userId) %>% summarize(b_u = sum(rating - mu - b_m)/(n()+lambda_opt), n_u
= n())

regular_beta_y <- sep_edx %>% left_join(regular_beta_m2, by='movieId') %>%
left_join(regular_beta_u2, by='userId') %>% group_by(year) %>%
summarize(b_y = sum(rating - mu - b_m - b_u)/(n()+lambda_opt), n_y = n())


predicting <- sep_validation %>% left_join(regular_beta_m2, by='movieId') %>%
left_join(regular_beta_u2, by='userId') %>% left_join(regular_beta_y, by = 'year') %>%
left_join(regular_beta_g, by = 'genres') %>% mutate(pred = mu + b_m + b_u + b_y)
%>% .$pred



model4 <- RMSE(sep_validation_a$rating,predicting)
rmse_tracker <- bind_rows(rmse_tracker, data_frame(method="Beta_m, beta_u, beta_y
(regularized)",  RMSE = model4 ))
rmse_tracker
```

7. Results

This first result shows us that regularizing and adding a beta for year will decrease(improve) our rmse.

```
> rmse_tracker
# A tibble: 5 x 2
  method                              RMSE
  <chr>                              <dbl>
1 Mean                               1.06
2 Mean + Beta_m                      0.944
3 Mean + b_m + b_u                   0.866
4 Beta_m and Beta_u (regularized)    0.865
5 Beta_m, beta_u, beta_y(regularized) 0.863
```

We could also add more factors. In this case, we added a beta for genre however we can see that rmse did not change or if it did it changed very little.

```
> rmse_tracker
# A tibble: 5 x 2
  method                                      RMSE
  <chr>                                      <dbl>
1 Mean                                       1.06 
2 Mean + Beta_m                              0.944
3 Mean + b_m + b_u                           0.866
4 Beta_m and Beta_u (regularized)            0.865
5 Beta_m, beta_u, beta_y, beta_g (regularized) 0.863
> 
```

Here we added another row, this is the same as the previous one except for the fact that our lambda here is 12 instead of the 14 value chosen a while ago.

```
> rmse_tracker
# A tibble: 6 x 2
  method                                      RMSE
  <chr>                                      <dbl>
1 Mean                                       1.06 
2 Mean + Beta_m                              0.944
3 Mean + b_m + b_u                           0.866
4 Beta_m and Beta_u (regularized)            0.865
5 Beta_m, beta_u, beta_y, beta_g (regularized) 0.863
6 Beta_m, beta_u, beta_y, beta_g (regularized) 0.863
> 
```

The reason for choosing as stated earlier is that the code takes forever to run, thus it was a safer option to choose a "smart" lamba choice. The decision process was basically to choose somewhere in the middle since there is a good chance that the lambda_opt would land there.

8. Conclusion

#After using the model (incorporating movie, user, year) we were able to lower the original rmse we got from using only the mean (from 1.06 to 0.863). One limitation of my approach was that we are not guaranteed that the final rmse (number 5 and number 6) are the absolute minimum since I was not able to simulate the selection of optimal lambdas (lambda_opt). However, we were able to reduce the rmse significantly to the point where we can say that our predictions will be of value.

References
1. https://github.com/cmrad/Updated-MovieLens-Rating-Prediction