---

title: "**Building a Movie Recommendation System**"

output:

  pdf_document: default

  Rmd_document: default

Project Overview

In this project, we will build a recommendation system in which we're able to use the dataset to predict what rating a particular user will give on a specific item. Therefore, items for which a high rating is predicted for a given user are then recommended to that user.

In order to accomplish the goal, we will compute the average of rating from the train-set, and consider both of movie-effect and user-specific factors to better fit our model in test-set. Finally comparing the RMSE between models to determine the better model.

Companies like Netflix, they use the similar recommendation systems to predict how many stars a user will give on a specific movie. One star suggests it's not a good movie, whereas five stars suggests it's an excellent movie.

Used Libraries

> library(tidyverse)

> library(ggplot2)

> library(caret)

> library(dplyr)

> library(matrixStats)

> library(dslabs)

> library(recommenderlab)

> library(data.table)

Dataset

We will use the 10M version of the MovieLens dataset - The dataset used is from Grouplens, and is publicly available at http://files.grouplens.org/datasets/movielens/ml-10m.zip.

We will use the following code to generate a train-set named as "edx" and test-set named as "validation" used in our project:

```{r}

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")
```

Data visualization, Data Exploration and Analysis Methods

Before building a model, we need to understand more about the data that we have on hand, so we first visualize some rows and columns:

```{r}
movielens[1:6,] %>%

       select(userId,movieId,rating,title)
```

	userId	movieId	rating	title
1	1	122	5	Boomerang (1992)
2	1	185	5	Net, The (1995)
3	1	231	5	Dumb & Dumber (1994)
4	1	292	5	Outbreak (1995)
5	1	316	5	Stargate (1994)

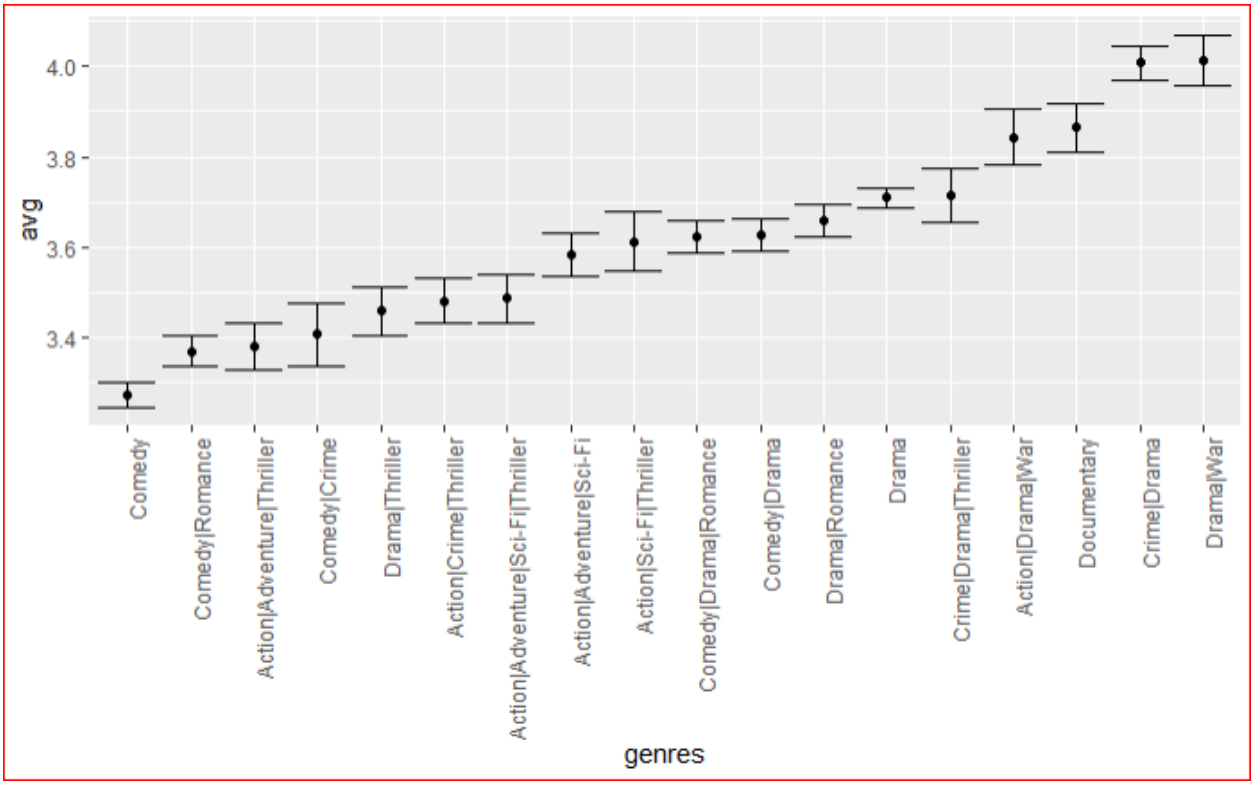
And to find out total how many users and movies within the dataset:

```{r}
movielens %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
  n_users   n_movies
  69878     10677
```

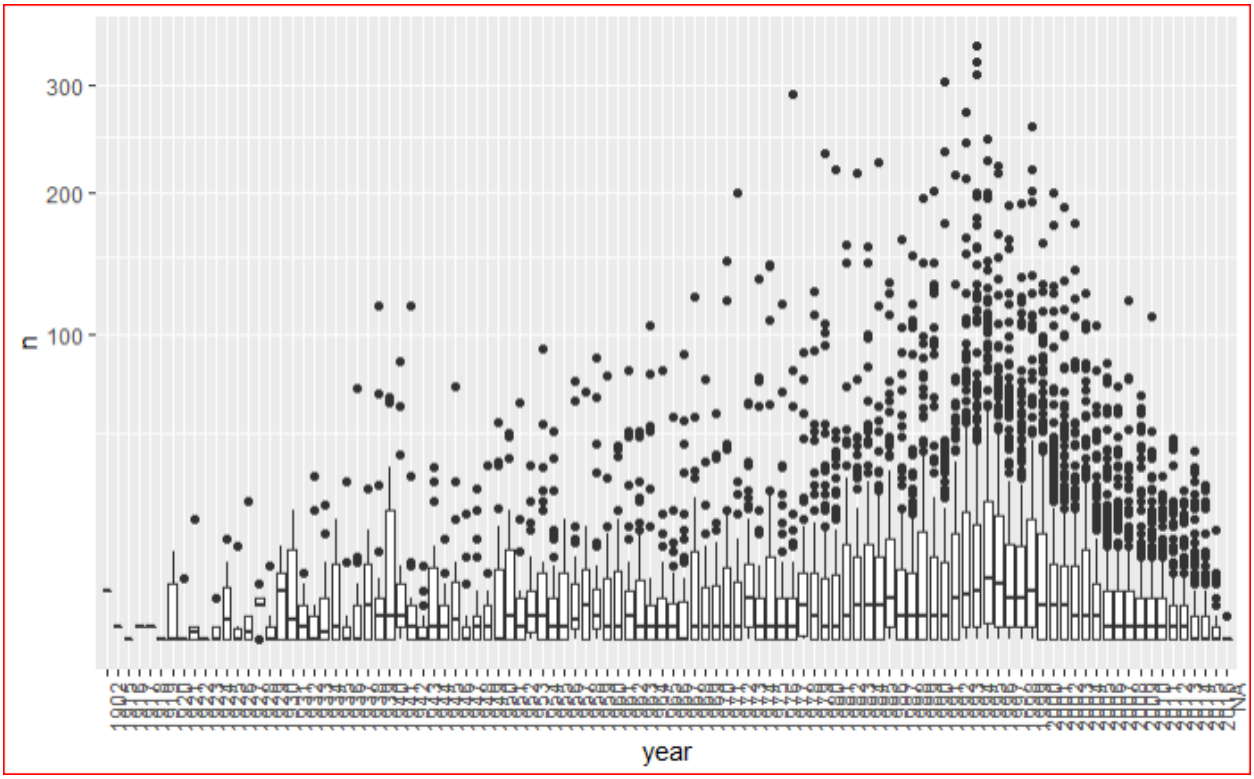
To further explore the data, we do some plotting. For example, to compute the average and standard error for each category of the movies, we then can use the analysis understanding more from the data:

```{r}
movielens %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



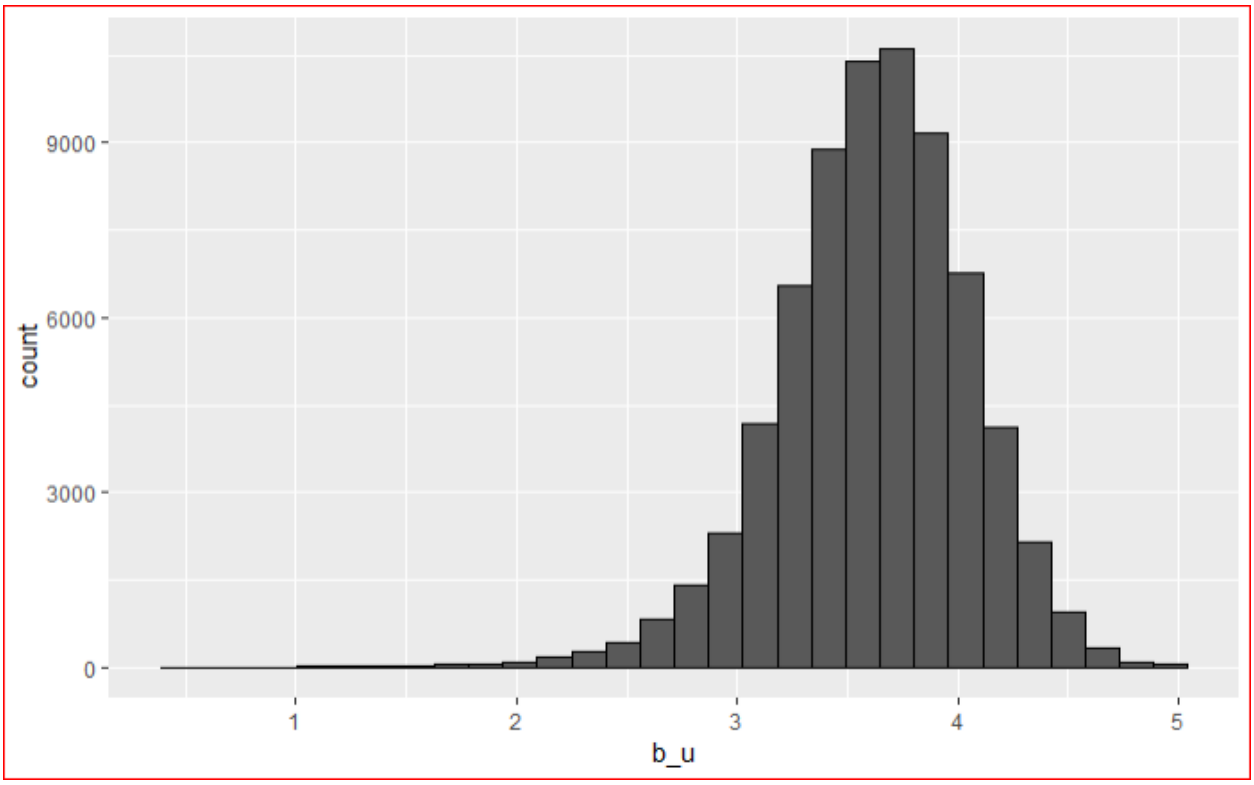
We may analyze the highest median number of ratings visualizing by year:

```{r}
movielens %>% group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(year))) %>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We can see more by computing the average rating for those users that have rated over 100 movies:

```{r}
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



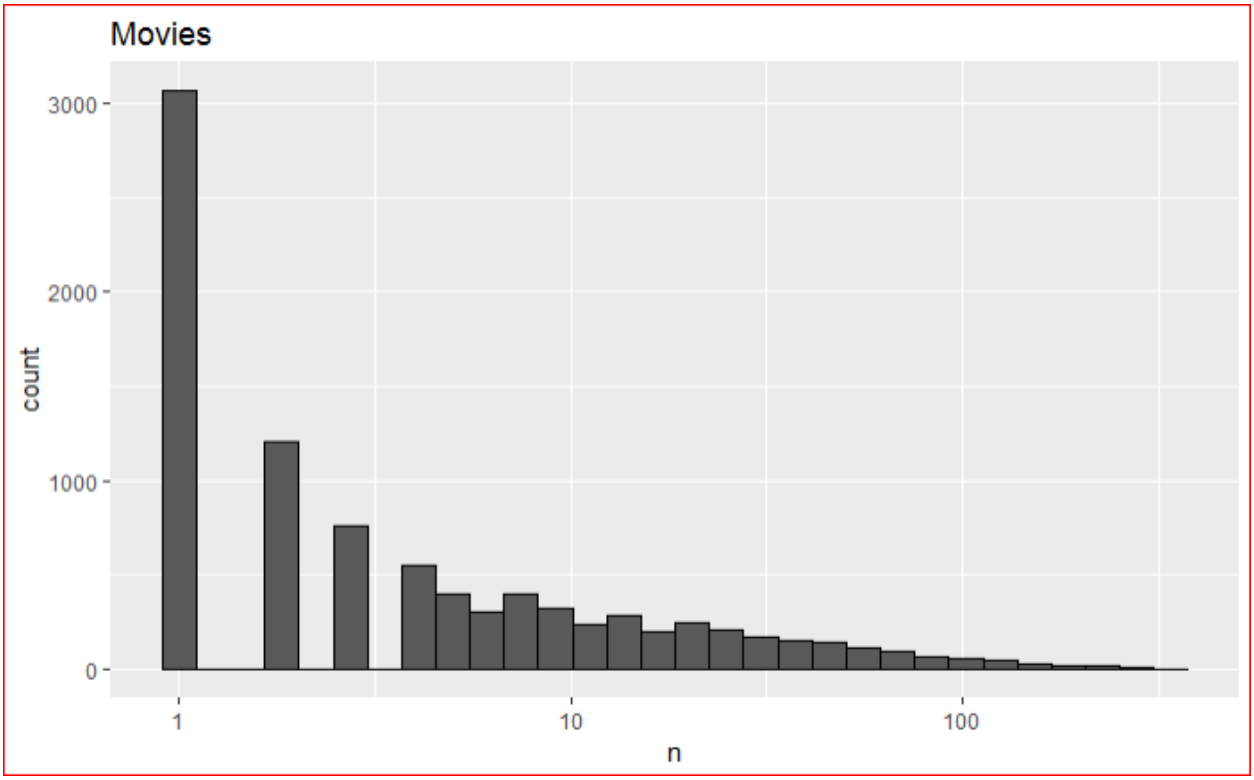
If we are predicting the rating for movie i by user u, in principle, all other ratings related to movie i and by user u may used as predictors, but different users rate a different number of movies and different movies. Furthermore, we may be able to use information from other movies that we have determined are similar to movie i or from users determined to be similar to user u. So, in essence, the entire matrix can be used as predictors for each cell.

We can see the movie-rating distribution:

```{r}
movielens %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
```

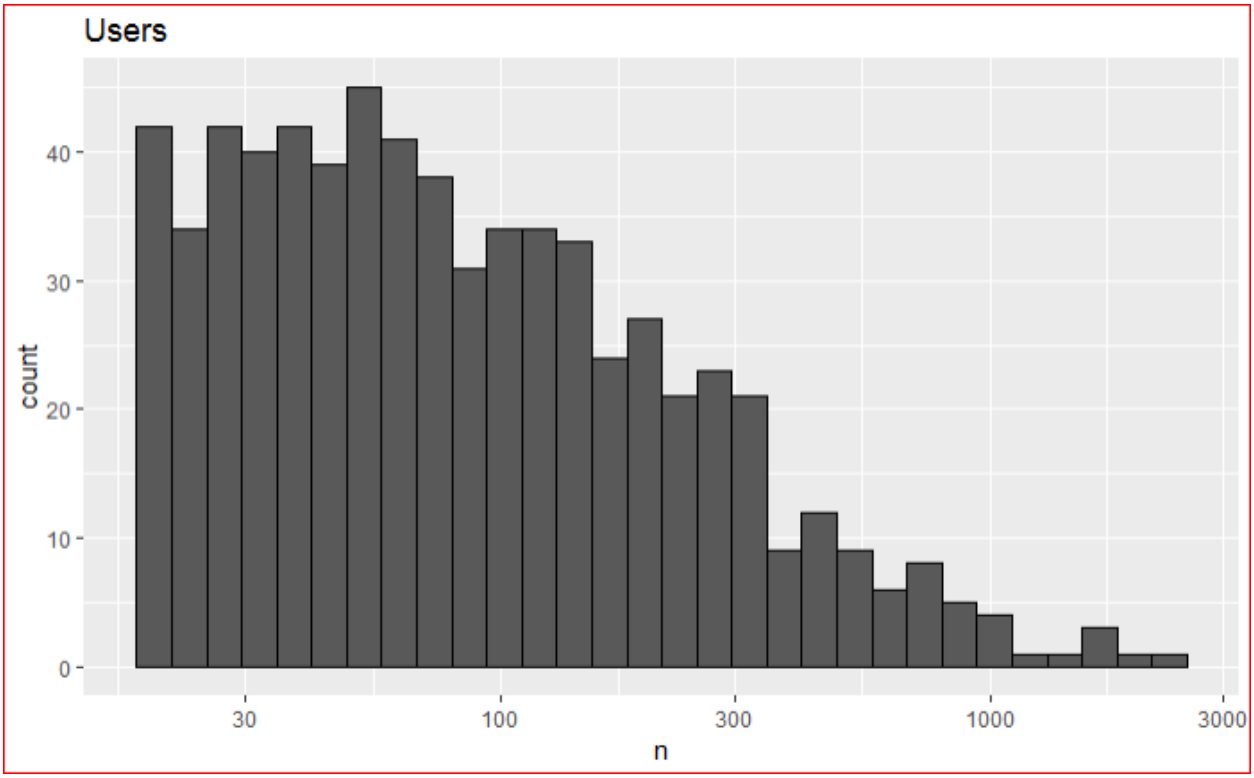
ggtitle("Movies")

```




Some users are more active than others at rating movies:

```{r}
movielens %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```
```



In this project, we consider Loss Function, and our models will be based on the residual mean squared error (RMSE) on a test-set. The RMSE is similarly to a standard deviation. If predicting error is larger than 1, it means the typical error is larger than one star, which is not good.

As we know, some movies are just generally rated higher than others. So We augment the first model by adding the term

b_i to represent average ranking for movie. We then build our first movie-effective model.

We can improve the model because we should include another factor as some users are more active than others at rating movies. This is a further improvement to our model.

In order to compare the models, we write the following function that computes the RMSE for vectors of ratings and their corresponding predictors based on the statistics formulation (Loss Function):

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

Fitting Models and Results

According to the foregoing analysis, discussion and methods, we can fit the first and final model:

```{r}
r_mu <- mean(edx$rating)

avgs_movie <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - r_mu))

predicted_ratingx <- r_mu + validation %>%
  left_join(avgs_movie, by = "movieId") %>%
  .$b_i
```

method	RMSE

|Effective Model | 0.9439087|

```{r}
avgs_user <- edx %>%
  left_join(avgs_movie, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - r_mu - b_i))

predicted_ratings <- validation %>%
  left_join(avgs_movie, by="movieId") %>%
  left_join(avgs_user, by="userId") %>%
  mutate(pred = r_mu + b_i + b_u) %>%
  .$pred
```

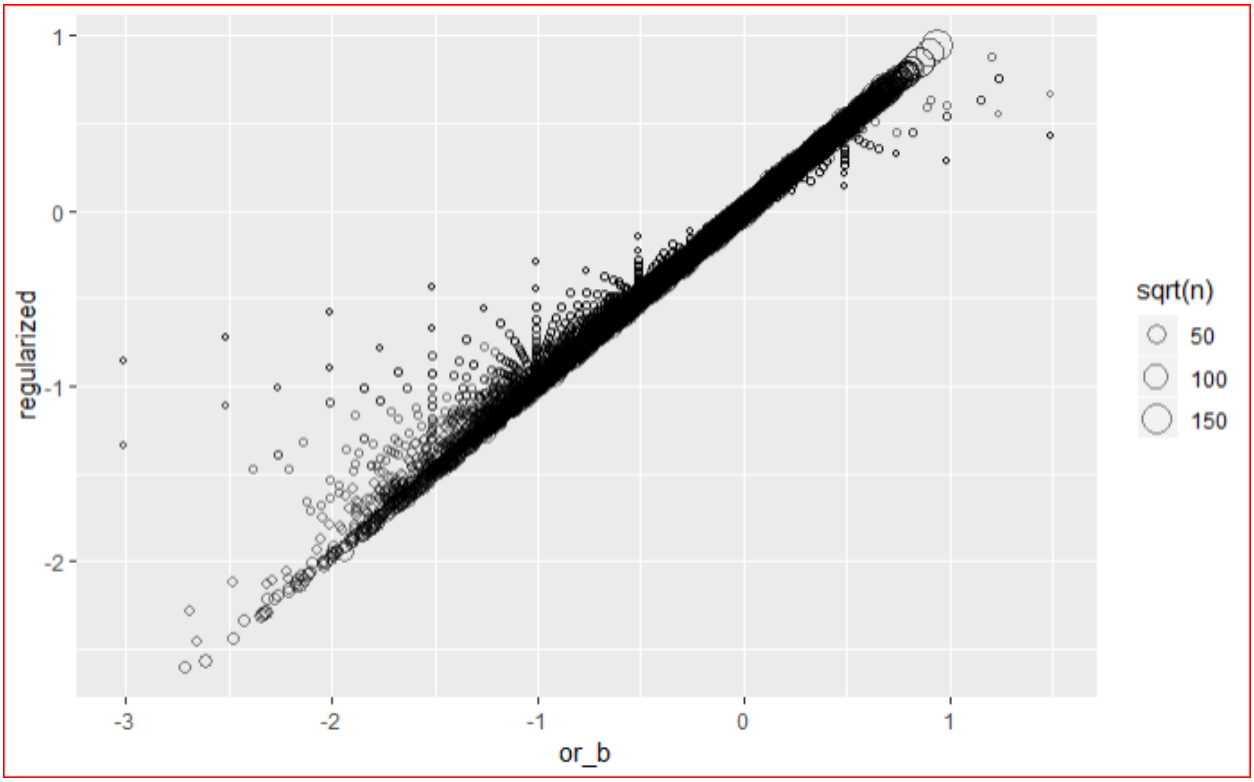
method	RMSE
Final Effective Model	0.8653488

However, we can do some more extra works. There are still some spaces to get improved. So we can use regularization method as to remove the bias of movie. Let's make a plot to view the regularized estimates vs. the least squares estimates:

```{r}
lambda <- 2.5
mu_1 <- mean(edx$rating)
```

```
movie_av_rg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_1)/(n()+lambda), n_i = n())


data_frame(or_b = avgs_movie$b_i,
           regularized = movie_av_rg$b_i,
           n = movie_av_rg$n_i) %>%
  ggplot(aes(or_b, regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5, color = "black")
```



In order to minimize the RMSE, now the question is - how can we pick the lambda value? Let's consider to remove the bias of both movie and user which means we need to account for movie

bias when calculating user bias. As a result, we're able to select the optimal lambda based on the metric that we're trying to minimize the RMSE:

```{r}
lambdas <- seq(0, 8, 0.25)
rmsex <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)

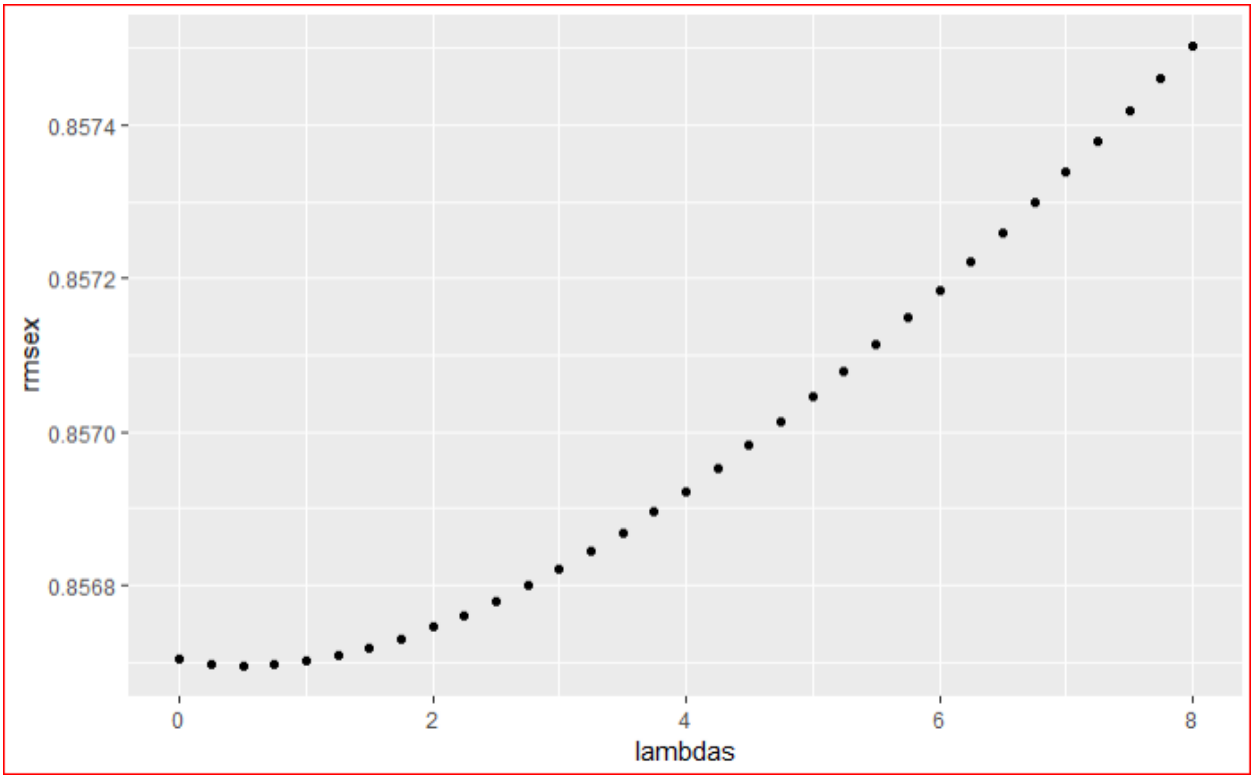
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratingx <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
return(RMSE(edx$rating,predicted_ratingx))
})
```

qplot(lambdas, rmsex)

```




```{r}

min_lda <- lambdas[which.min(rmsex)]

min_lda

```

Therefore, as we now have the optimal lambda, we can compare the models based on their RMSE:

```{r}

min_rmse <- sapply(min_lda, function(l){

  mu <- mean(edx$rating)

  b_i <- edx %>%

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

 predicted_ratingx <- edx %>%
   left_join(b_i, by = "movieId") %>%
   left_join(b_u, by = "userId") %>%
   mutate(pred = mu + b_i + b_u) %>%
   .$pred
 return(RMSE(edx$rating,predicted_ratingx))
})

rmse_results <- bind_rows(rmse_results,
                    data_frame(method="Regularized Model",
                               RMSE = min_rmse))

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

method	RMSE
Effective Model	0.9439087

|Final Effective Model | 0.8653488|

|Regularized Model | 0.8566952|

Conclusion

Based on the RMSE results, when we consider both user-specific and movie-effect, a further improvement to the final model not surprisingly meets our expectation:

|method | RMSE|

|:---|---:|

|Effective Model | 0.9439087|

|Final Effective Model | 0.8653488|