MovieLens Capstone Project

Mohamed Elsayed

August 3rd, 2021

I. Introduction

The MovieLens dataset is a database with over 10 million ratings for over 10,000 movies by more than 72,000 users as per the grouplens website. Based on these numbers, we would expect to have around 720 million ratings if every user rated every movie. But since the dataset has around 10 million ratings only this implies that the majority of movies were not rated which is a clear sign of a "Sparse Matrix". With the data available in this dataset I will build a machine learning algorithm to try to predict user ratings as accurately as possible. **RMSE** will be the method used to evaluate the performance of different models during development & therefore, several subsets of the `movielens` dataset will be used in this process as outlined below.

1. `edx` dataset which holds 90% of `MovieLens` data will be used for developing the model. Since the remaining 10% of the data will be treated as the TRUE ratings, they **Can't** be used to test the performance of models during development & therefore, `edx` will be split into:

- `train_set` which holds 80% of the data in `edx` to train the different models during development.
- `test_set` which holds 20% of the data in `edx` to test the performance of different models during development.

2. `validation` dataset which holds 10% of the `movielens` data will be used **only** for the final calculation of RMSE of to evaluate the best performing model.

The data includes a `userID` which is assigned to every unique user, `movieID` which is assigned to every unique movie, movie `ratings`, the `timestamp` of every rating, movie `title` & the movie `genre`.

Due to the large size of the dataset, using linear models ***lm*** to fit data & make predictions was not used. Instead, I will use Least Squares Estimates ***LSE*** to predict the ratings then use **RMSE** to evaluate those predictions as mentioned above. **My goal is to reach RMSE of < 0.80**.

II. Method

The methods in this project aim at developing a movie recommendation system based on user ratings. As mentioned above, *LSE* will be used to replace Linear Regression models as I develop the algorithm due to the large size of the data. In this section I will highlight below the steps used, including data preparation, data exploration and visualization & the insights gained, leading to building the recommendation model.

1. Data Preparation

Information on the MovieLens 10M dataset can be found here: https://grouplens.org/datasets/movielens/10m/, The dataset can be downloaded here: http://files.grouplens.org/datasets/movielens/ml-10m.zip

The following code is used to download the dataset and combine the information of the ratings and the movies into `movielens`. Then the `edx` & the `validation` datasets are created which will be used in developing & testing the algorithm.

```r
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)
library(data.table)
library(recommenderlab)
library(recosystem)
library(ggplot2)

# 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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
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") # if using R 3.5 or earlier, use `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)
```

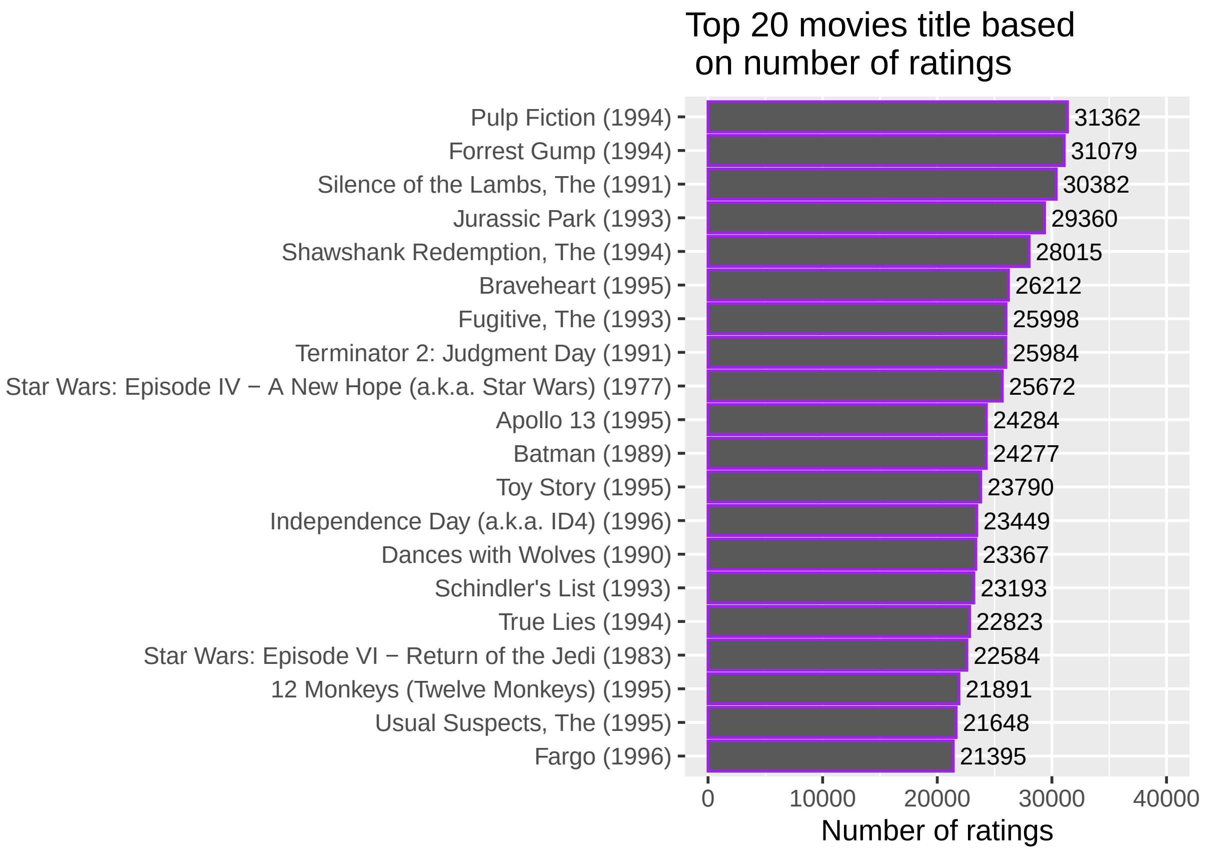
2. Data Exploration & Visualization

Now that the dataset is created, some data exploration will be needed to start building the approach with which the algorithm will be developed. Below I First explore how many users & movies are in the dataset, then I plot the top 20 rated movies, then I explore the number of ratings by movie & by user in separate plots.

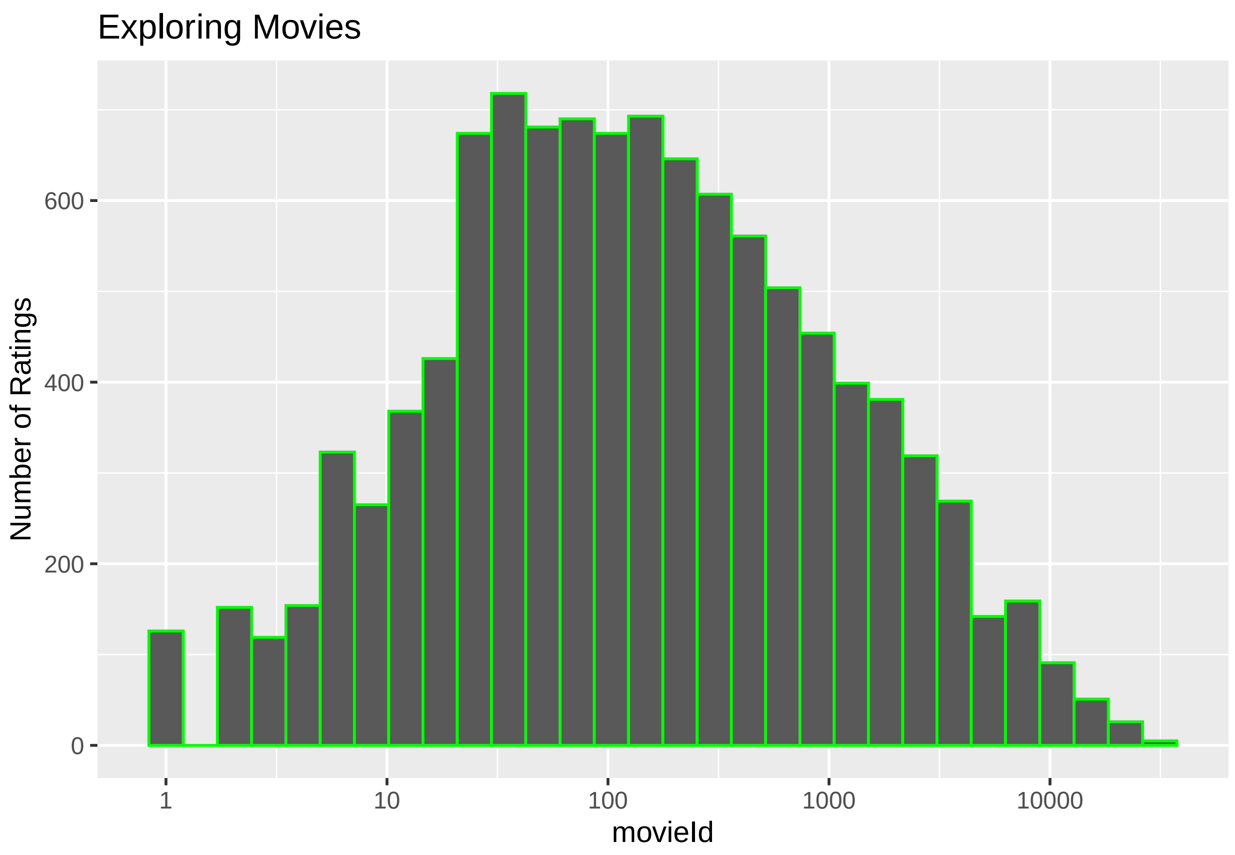
```
# Checking the number of unique users that provided ratings and how many unique movies were rated
edx %>% summarize(n_users = n_distinct(userId),n_movies = n_distinct(movieId))
```

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

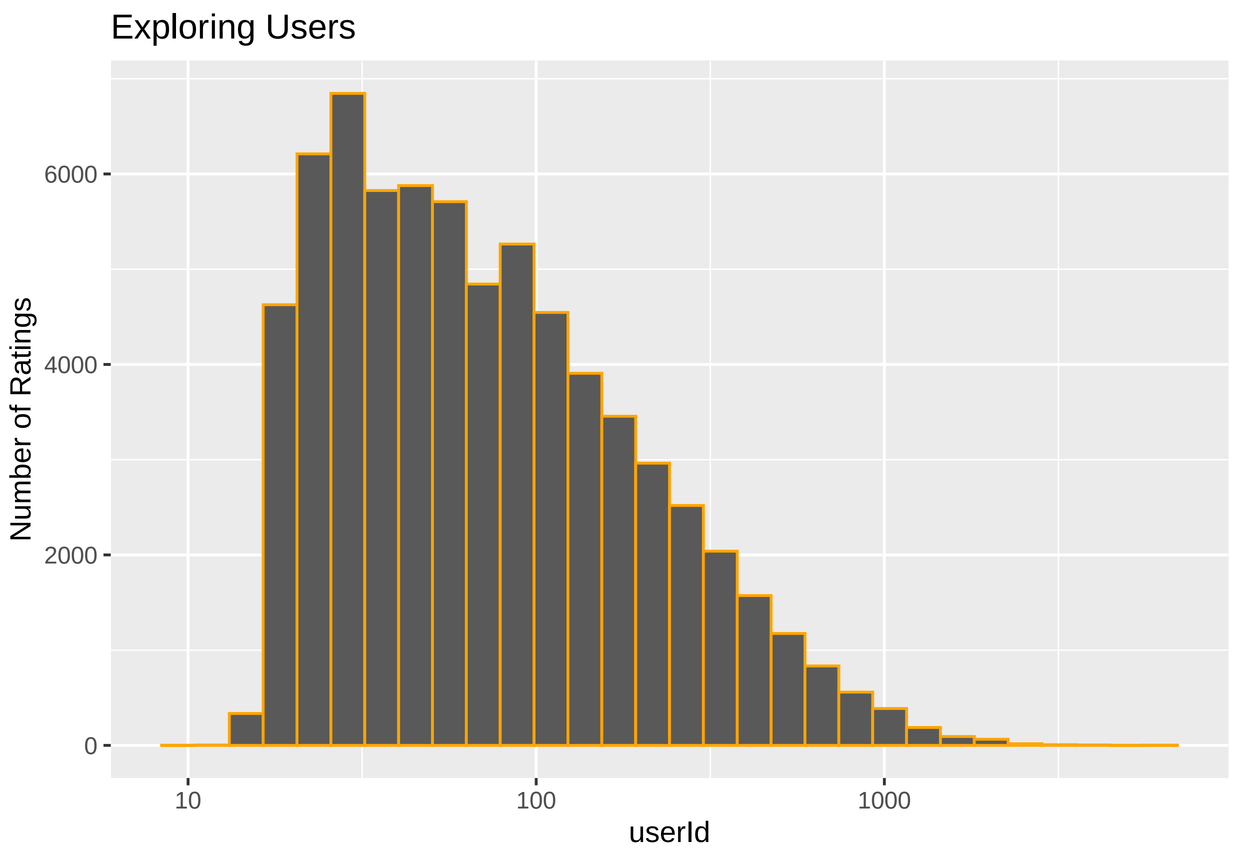
```
# Plotting top 20 movies
top_20 <- edx %>% group_by(title) %>% summarize(count=n()) %>% top_n(20,count) %>%
  arrange(desc(count))

top_20 %>%
ggplot(aes(x=reorder(title, count), y=count)) +
geom_bar(stat="identity",color= "purple") + coord_flip(y=c(0, 40000)) +
labs(x="", y="Number of ratings") +
geom_text(aes(label= count), hjust=-0.1, size=3) +
labs(title="Top 20 movies title based \n on number of ratings")
```



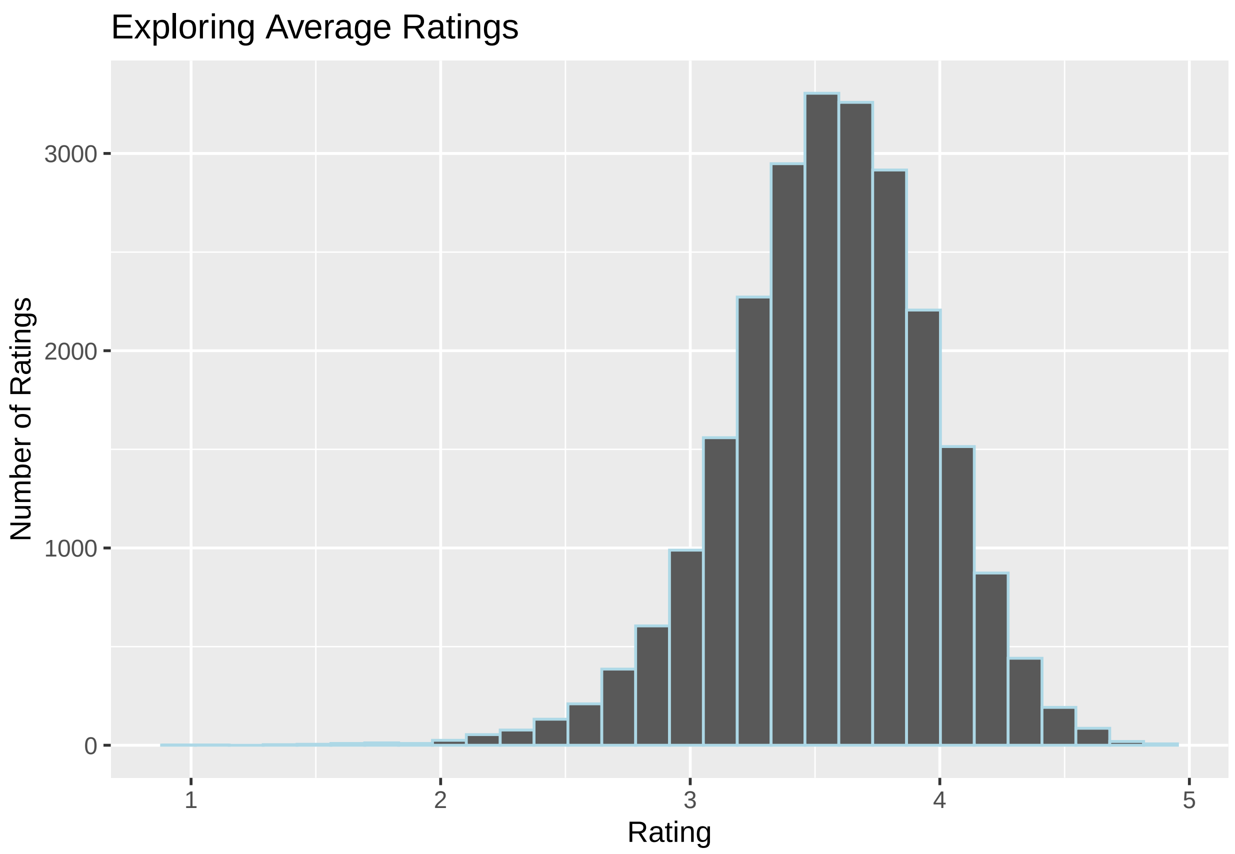
```r
# Exploring the number of ratings by movieId
edx %>%
    dplyr::count(movieId) %>%
    ggplot(aes(n)) +
    geom_histogram(bins = 30, color = "green") +
    scale_x_log10() +
    ggtitle("Exploring Movies") +
  labs(x= "movieId", y= "Number of Ratings")
```



```r
# Exploring the number of ratings by userId
edx %>%
    dplyr::count(userId) %>%
    ggplot(aes(n)) +
    geom_histogram(bins = 30, color = "orange") +
    scale_x_log10() +
    ggtitle("Exploring Users") +
  labs(x= "userId", y= "Number of Ratings")
```



```r
# Exploring the average ratings
 edx %>%
   group_by(userId) %>%
   filter(n()>=100) %>%
   summarize(b_u = mean(rating)) %>%
   ggplot(aes(b_u)) +
   geom_histogram(bins = 30, color = "lightblue") +
   ggtitle("Exploring Average Ratings") +
   labs(x= "Rating", y= "Number of Ratings")
```



3. Data Insights

The first thing to be noticed is that Not ALL movies are being rated. In fact as per the numbers above, there should be around 746 million reviews while there's only 10 million. Then there are some movies that are rated many more times than others & there are some users who usually rate much more movies than others. Similarly, there are users who usually give low or high rating regardless of the movie. That information is important for creating the strategy with which the algorithm will be developed. The Movie & User variability will be important effects that will be considered while developing the algorithm.

4. Modeling Approach

In order to judge which algorithm performs better we need to specify a way to quantify what does better or worse mean. In this project I will use **RMSE (Residual Mean Squared Error)**. We can interpret the RMSE similar to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good. Below is the formula used for RMSE.

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

Since the `validation` dataset can't be used until models are already built, a `test_set` is needed to evaluate model performance during development & compare different approaches to each other. In order to do this, I will split the `edx` dataset into `train_set` & `test_set` as per the code below. **Please note:** that In order to make sure that the movies & users in the `test_set` match the ones in the `train_set` I used the `semi_join()` function.

```r
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

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

5. Building models

I will start with a basic “Baseline” approach of predicting ratings then I will build on that basic model to include movie effects, user effects, regularization & matrix factorization until RMSE of < 0.80 is reached.

A. Baseline Model: The simplest model that we can start with & use as our baseline would be predicting the same rating for all movies & all users. I will use the average rating of all movies across all users as the first algorithm. Then I calculate the RMSE for that model.

```r
mu_baseline<- mean(train_set$rating)                                # Average Rating
baseline_RMSE<- RMSE(test_set$rating, mu_baseline)                  # Baseline RMSE
RMSE_results <- tibble(Model = "Baseline Model", RMSE = baseline_RMSE)    # Create a table with RMSE's
baseline_RMSE
```

Model	RMSE
Baseline	1.060698

B. Movie Effect Model: After exploring the number of ratings per movie plot above in the Data Exploration section, there seems to be a high degree of variability of the number of ratings for different movies. That’s expected as blockbuster movies are typically viewed more than small budget movies. In the Model below I will try to account for that effect.

```r
mu_least_squares <- mean(train_set$rating)

LSE_rating_movies<- train_set %>% group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_least_squares))

LSE_prediction_movies<- mu_least_squares + test_set %>%
  left_join(LSE_rating_movies, by='movieId') %>% .$b_i

LSE_RMSE_movies <- RMSE(LSE_prediction_movies, test_set$rating)

RMSE_results <- bind_rows(RMSE_results, tibble(Model="Movie Effect Model",
                                               RMSE = LSE_RMSE_movies ))
LSE_RMSE_movies
```

Model	RMSE
Move Effect	0.9439526

C. Adding User Effect to Movie Effect Model: Improving a bit further. Turning to "users". Since different users differ in how they rate movies as shown above in the Data Exploration section, I can account for the user effect "b_u". RMSE shows an improvement after accounting for that effect.

```
LSE_rating_users <- train_set %>% left_join(LSE_rating_movies, by='movieId') %>%
  group_by(userId) %>% summarize(b_u = mean(rating - mu_least_squares - b_i))

LSE_prediction_users <- test_set %>% left_join(LSE_rating_movies, by='movieId') %>%
  left_join(LSE_rating_users, by='userId') %>%
  mutate(pred = mu_least_squares + b_i + b_u) %>% .$pred

LSE_RMSE_users <- RMSE(LSE_prediction_users, test_set$rating)
RMSE_results <- bind_rows(RMSE_results, tibble(Model="Movie + User Effect Model",
                                               RMSE = LSE_RMSE_users ))

LSE_RMSE_users
```

Model	RMSE
Movie + User Effect	0.8661319

D. Regularization: Checking the table below gives a few more insights of the variability of ratings for both movies & users. Movies that were rated thousands of times shouldn't have the same weight in calculating the average as a movie that was only rated once!. And the same applies to user ratings as well.

Statistic	Avg	Min	Max
Movie Rarings	843	1	31362
User Ratings	129	10	6616

So far, the previous models haven't accounted for that effect in calculating averages. This is why I will introduce **Regularization** to account for that effect in the model below. The term **lambda** represents the "penalty" for penalizing large estimates that come from small sample sizes. I will use cross-validation to tune for the best lambda and calculate RMSE for that value.

```
lambdas <- seq(0, 10, 0.25)
best_lambda <- sapply(lambdas, function(l){
     mu <- mean(train_set$rating)
     b_i <- train_set %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))
     b_u <- train_set %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
     predicted_ratings <-
          test_set %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          mutate(pred = mu + b_i + b_u) %>%
          .$pred
     return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, best_lambda)
lambda <- lambdas[which.min(best_lambda)]
lambda

RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model="Regularized Movie + User Effect Model",
                                      RMSE = min(best_lambda)))
RMSE_results
```

Model	RMSE
Regularization	0.8654617

E. Matrix Factorization: Since not every user has rated every movie, there are many missing values in the matrix which results in a sparse matrix. The method used to solve that problem would be matrix factorization. The main point here is to try to reconstruct "r" (residuals) with the data available in our dataset. The model below uses matrix factorization to try to improve RMSE

```r
# Calculating Movie Effect to be used in Matrix Factorization
b_i <- edx %>% group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_least_squares)/(n()+lambda))

# Calculating Movie + User Effect to be used in Matrix Factorization
b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_least_squares)/(n()+lambda))

# Adding the Residuals to "edx"
edx_residual <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(residual = rating - mu_least_squares - b_i - b_u) %>%
  select(userId, movieId, residual)

# Preparing the datasets to be used with recommenderlab package
write.table(train_set , file = "trainset.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)
write.table(test_set , file = "testset.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

edx_mf <- as.matrix(edx_residual)
validation_mf <- validation %>% select(userId, movieId, rating)
validation_mf<- as.matrix(validation_mf)

write.table(validation_mf , file = "validation.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)
train_set_mf<- data_file("trainset.txt")
test_set_mf<- data_file("testset.txt")
validation_mf<- data_file("validation.txt")

# Building Recommender object & tuning it's parameters
r<- Reco()
tuning_mf <- r$tune(train_set_mf, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                        costp_l1 = 0, costq_l1 = 0,
                                        nthread = 1, niter = 10))

# Training the Recommender model
r$train(train_set_mf, opts = c(tuning_mf$min, nthread = 1, niter = 20))

# Making prediction on the test_set % calculating RMSE
pred_file <- tempfile()
r$predict(test_set_mf, out_file(pred_file))
predicted_residuals_mf <- scan(pred_file)
mf_RMSE <- RMSE(predicted_residuals_mf, test_set$rating)
RMSE_results <- bind_rows(RMSE_results, tibble(Model="Matrix Factorization Model",
                                              RMSE = mf_RMSE))
```

Model	RMSE
Matrix Factorization	0.7909813

III. Results

As building the model progressed, the RMSE continued to decrease until it reached the goal of being **< 0.8**. I have summarized all RMSE results of all the models built in this project in the below table. From the results shown, the best RMSE was achieved through **"Matrix Factorization"** but this RMSE was calculated based on predictions made from the `test_set` not the `validation` set.

Model	RMSE
Baseline Model	1.061
Movie Effect Model	0.944
Movie + User Effect Model	0.866
Regularized Movie + User Effect Model	0.865
Matrix Factorization Model	0.791

Therefore, in this section I will use the best performing model "Matrix Factorization" to make predictions on the `validation` set & display the results.

```r
# Preparing Validation Set
validation_rating <- read.table("validation.txt", header = FALSE, sep = " ")$V3

# Making predictions on the validation set
r$predict(validation_mf, out_file(pred_file))
predicted_residuals_mf_validation <- scan(pred_file)

# Calculating RMSE
mf_RMSE_validation <- RMSE(predicted_residuals_mf_validation, validation_rating)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model="Validation using Matrix Factorization Model",
                                 RMSE = mf_RMSE_validation))
mf_RMSE_validation
```

Model	RMSE
Validation using Matrix Factorization	0.790357

IV. Conclusion

Model	RMSE
Baseline Model	1.061
Movie Effect Model	0.944
Movie + User Effect Model	0.866
Regularized Movie + User Effect Model	0.865
Matrix Factorization Model	0.791
Validation using Matrix Factorization Model	0.790

From the summary of RMSE table above, it's clear that the Matrix Factorization model was the best performing model as it brought down RMSE from it's Baseline value of *1.060* to *0.790* which is a good improvement. Future work on this project could examine further aspects like correlations between movie genres & users of the same age groups to build models that utilize PCA & SVD to achieve even better prediction of ratings hence, be better at recommending movies that are going to be liked by users.