MovieLens Project Report

Nikhil Venkatachalam

July 19, 2023

Introduction

The goal of this project was to make a movie recommendation software utilizing machine learning algorithms. Specifically, the movie recommendation software must have aspired to a certain level of quality and accuracy. This level was measured by the RMSE (Root Mean Square Error), where a lower RMSE would be ideal. The RMSE goal set was 0.86490, so achieving less than this number would have been the target. This was accomplished by initially engaging in data exploration (i.e. visualizing the data and outputting its summary statistics), then moving on to actually training the data by first accounting for various biases in the rating of various movies, then using machine learning techniques to further decrease the RMSE and make the movie recommendation software optimal.

Initial Setup

This project utilized a dataset collected and provided by GroupLens, a research lab at the University of Minnesota specializing in recommender systems, among other things. As such, GroupLens has collected millions of movie reviews, offering these reviews in the form of datasets of various sizes. For this project, the 10M dataset was used, and was loaded as follows:

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

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

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

```
## Loading required package: caret
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"),
                                   simplify = TRUE), stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"),
                                  simplify = TRUE), stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>% mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with ‘by = join_by(userId, movieId, rating, timestamp, title, genres)‘
```

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Dividing the Algorithm

As can be observed, the 10M dataset was divided into two separate datasets,
those being edx and final_holdout_test. Then, the edx dataset was further split
into two datasets in order to properly build and test the machine learning algorithm. However, the final test was conducted using the final_holdout_test dataset.

```
# Further division of edx into training and testing sets
set.seed(1, sample.kind = "Rounding") # using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform ’Rounding’
## sampler used
```

```
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Matching userId and movieId in both train and test sets
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Adding rows back into train set
removed <- anti_join(temp, test_set)
```

```
## Joining with ‘by = join_by(userId, movieId, rating, timestamp, title, genres)‘
```

```
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Data Exploration

First, before engaging in any training of the data, I first had to take a closer look at the dataset that was provided. To do this, I outputted some summary statistics, and visualized the data so that I could better understand the data contained within the dataset.

Statistical Summary

```
# Statistical summary of the dataset edx
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  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
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The summary() function provided a statistical summary of the data contained within the dataset, which helped me look at the data with a new perspective.

Unique Users Versus Unique Movies

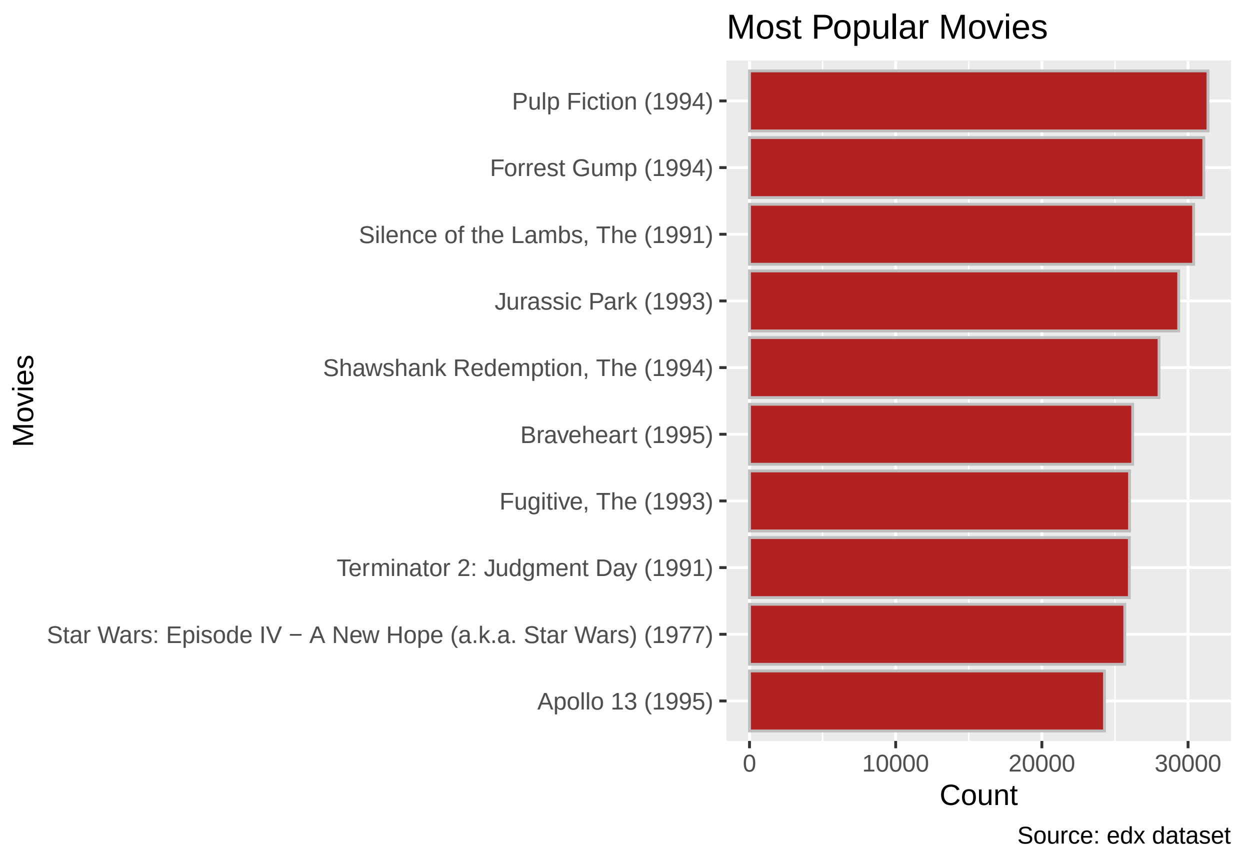
```
# Output number of users versus number of movies
summarize(edx, num_users = n_distinct(userId), num_movies = n_distinct(movieId))
```

```
##   num_users num_movies
## 1     69878      10677
```

As can be observed, there are 69878 unique users and 10677 unique movies within the edx dataset.

Top Movies Graph

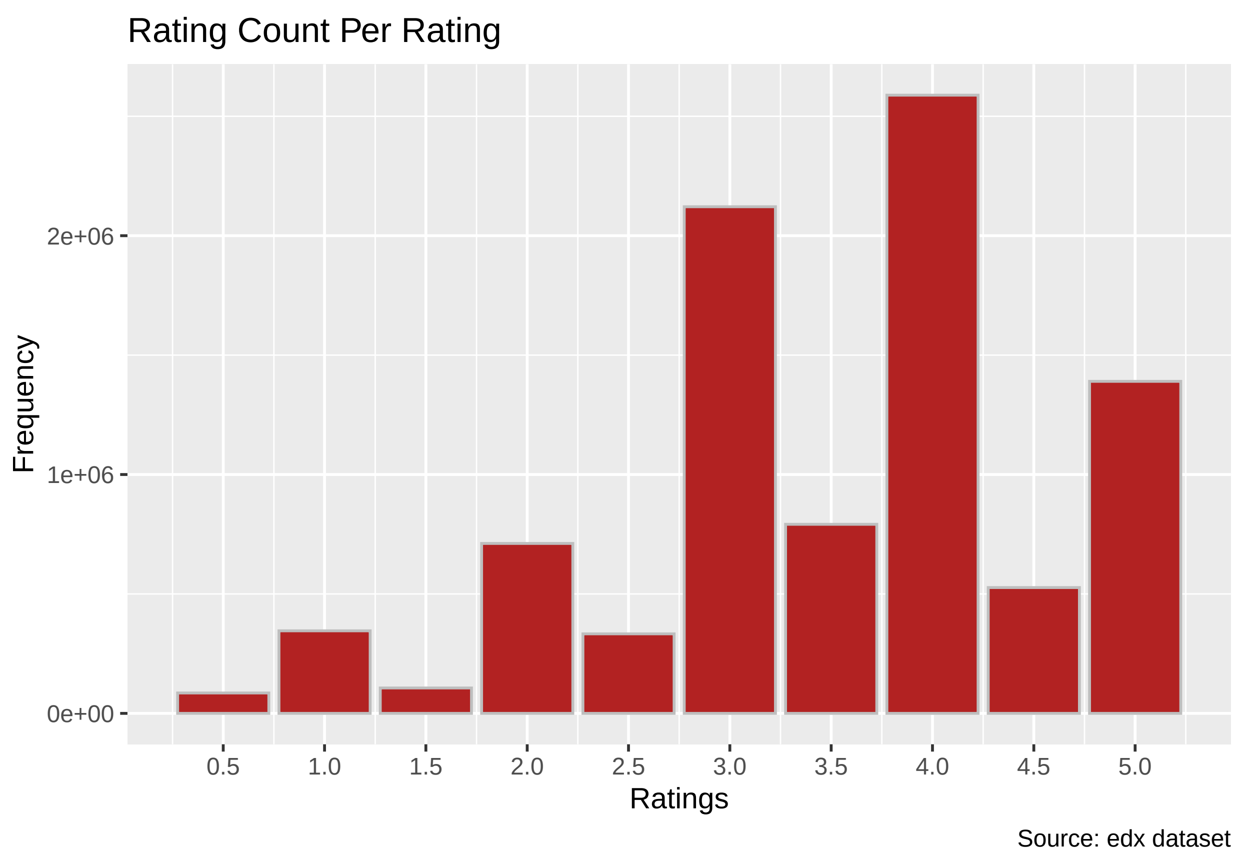
```
# Graph top movies
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  top_n(10, count) %>%
  arrange(-count) %>%
  ggplot(aes(count, reorder(title, count))) +
  geom_bar(color = "gray", fill = "firebrick", stat = "identity") +
  labs(x = "Count", y = "Movies", caption = "Source: edx dataset") +
  ggtitle("Most Popular Movies")
```



This graph is of the top 10 most popular movies within the dataset.

Number of Ratings Per Rating Graph

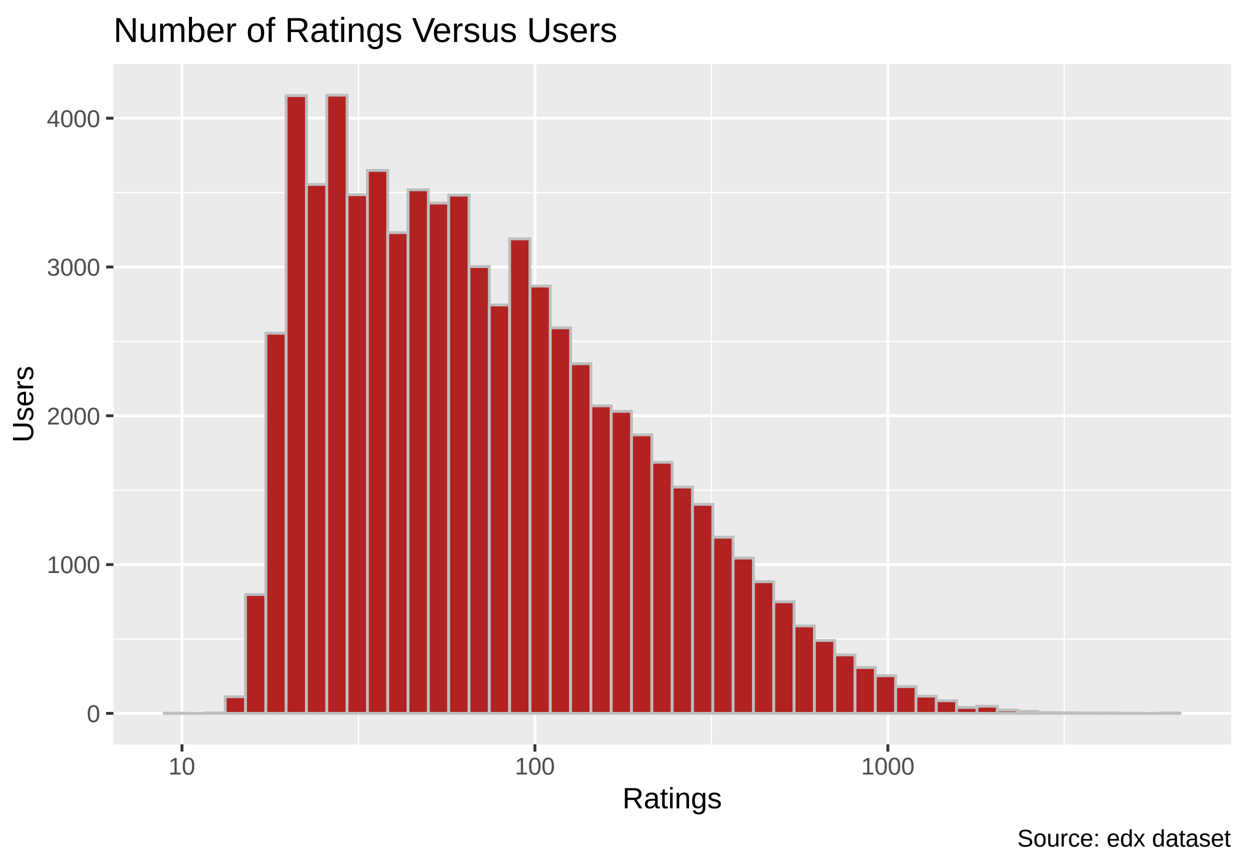
```
# Graph number of ratings per rating
edx %>%
  ggplot(aes(rating)) +
  geom_bar(color = "gray", fill = "firebrick") +
  labs(x = "Ratings", y = "Frequency", caption = "Source: edx dataset") +
  scale_x_continuous(breaks = seq(0, 5, by = 0.5)) +
  ggtitle("Rating Count Per Rating")
```



Within this graph, one can observe that no user has given 0 as a rating, and whole-star ratings are much more common than half-star ratings.

Number of Ratings Versus Users Graph

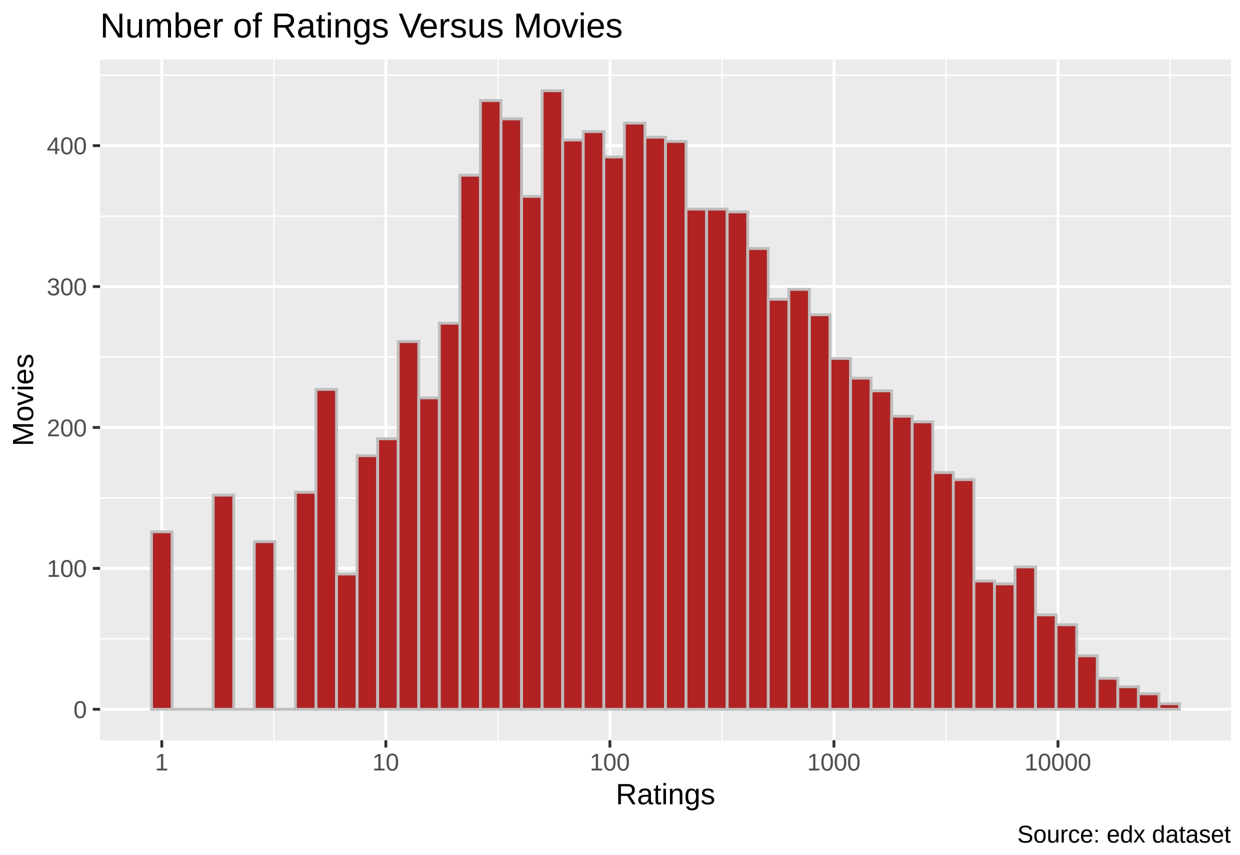
```r
# Graph number of ratings versus users
edx %>%
  group_by(userId) %>%
  summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color = "gray", fill = "firebrick", bins = 50) +
  labs(x = "Ratings", y = "Users", caption = "Source: edx dataset") +
  ggtitle("Number of Ratings Versus Users") +
  scale_x_log10()
```



Looking at this graph, one can observe that the graph is skewed to the right. This indicates that most users are not as active, but there are a few users that are much more active than the others.

Number of Ratings Versus Movies Graph

```r
# Graph number of ratings versus movies
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color = "gray", fill = "firebrick", bins = 50) +
  labs(x = "Ratings", y = "Movies", caption = "Source: edx dataset") +
  ggtitle("Number of Ratings Versus Movies") +
  scale_x_log10()
```



Compared to the Rating Versus Users graph, this graph is much more normally distributed. Still, one can observe that some movies get rated much more often than others.

Data Analysis

With the data exploration finished, it was time to start on the actual data analysis and training. During this time, I trained 7 models in total, each one bringing the overall quality of the model closer to the ideal RMSE defined at the start of the project.

RMSE and Mean

To start the data analysis, I made a function that would calculate the RMSE so that I would not have to calculate it manually every time I created a new model.

```r
# Function to calculate RMSE
rmse <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

This is the function mentioned above. It takes in the actual movie ratings and the movie ratings predicted by the model, and outputs the RMSE of the model.

```r
# Mean of all ratings
mean_rating <- mean(train_set$rating)
mean_rating
```

```
## [1] 3.512456
```

As a baseline, I calculated the mean of all the ratings in the training set. This was then used to train the first model.

```r
# RMSE calculated with just the mean
mean_rmse <- rmse(test_set$rating, mean_rating)
mean_rmse
```

```
## [1] 1.060054
```

This first model was trained using simply the mean of the training set. Because the RMSE is above 1, this is not a very good model, and needs to be improved upon in later models.

Adding Bias to Calculations

Once I established a starting point for my future models to improve on, I could implement various changes and fixes that would decrease the RMSE further in order to bring myself closer to the goal set previously. Therefore, I decided to account for the various biases that can exist in the ratings of movies.

Adding Movie Bias

As can be seen through experience, some types of movies are more popular than others. Therefore, I added this bias into the next model trained.

```r
# Add movie bias to calculation
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mean_rating))
predicted_ratings <- mean_rating + test_set %>%
  left_join(bi, by = "movieId") %>%
  pull(b_i)
```

```r
# RMSE calculated with mean and movie bias
movie_bias_rmse <- rmse(predicted_ratings, test_set$rating)
movie_bias_rmse
```

```
## [1] 0.9429615
```

As can be observed, adding movie bias to the model decreased the RMSE below 1. This was built upon in future models.

Adding User Bias

Next, the users (i.e. the people who rate the movies) are biased in their own way, as certain users will rate movies a certain way. For instance, some users may tend to be overly positive or overly negative in their ratings. Thus, I took this into account with the next model.

```r
# Add user bias to calculation
bu <- train_set %>%
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mean_rating - b_i))
predicted_ratings <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(pred = mean_rating + b_i + b_u) %>%
  pull(pred)
```

```r
# RMSE calculated with mean, movie, and user bias
user_bias_rmse <- rmse(predicted_ratings, test_set$rating)
user_bias_rmse
```

```
## [1] 0.8646843
```

Adding user bias to this next model reduced the RMSE further. In fact, this RMSE is below the target set at the beginning of the project, but it is not low enough to be a significant change. Therefore, further models improved on this number.

Adding Time Bias

Movies are also rated differently depending on the time they came out. Movies may not generate as much hype if they come out during a non-peak time, so they will be rated lower. Thus, I accounted for this with the next model by adding time bias into the calculations.

```r
# Add time bias to calculation
bt <- train_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mean_rating - b_i - b_u))
predicted_ratings <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(bt, by = "date") %>%
  mutate(pred = mean_rating + b_i + b_u + b_t) %>%
  pull(pred)
```

```r
# RMSE calculated with mean, movie, user, and time bias
time_bias_rmse <- rmse(predicted_ratings, test_set$rating)
time_bias_rmse
```

```
## [1] 0.8645933
```

Adding time bias to the model made it perform better, but still not by a significant amount. In fact, this improvement was the smallest one yet. Therefore, to decrease the RMSE further, I had to employ a new strategy.

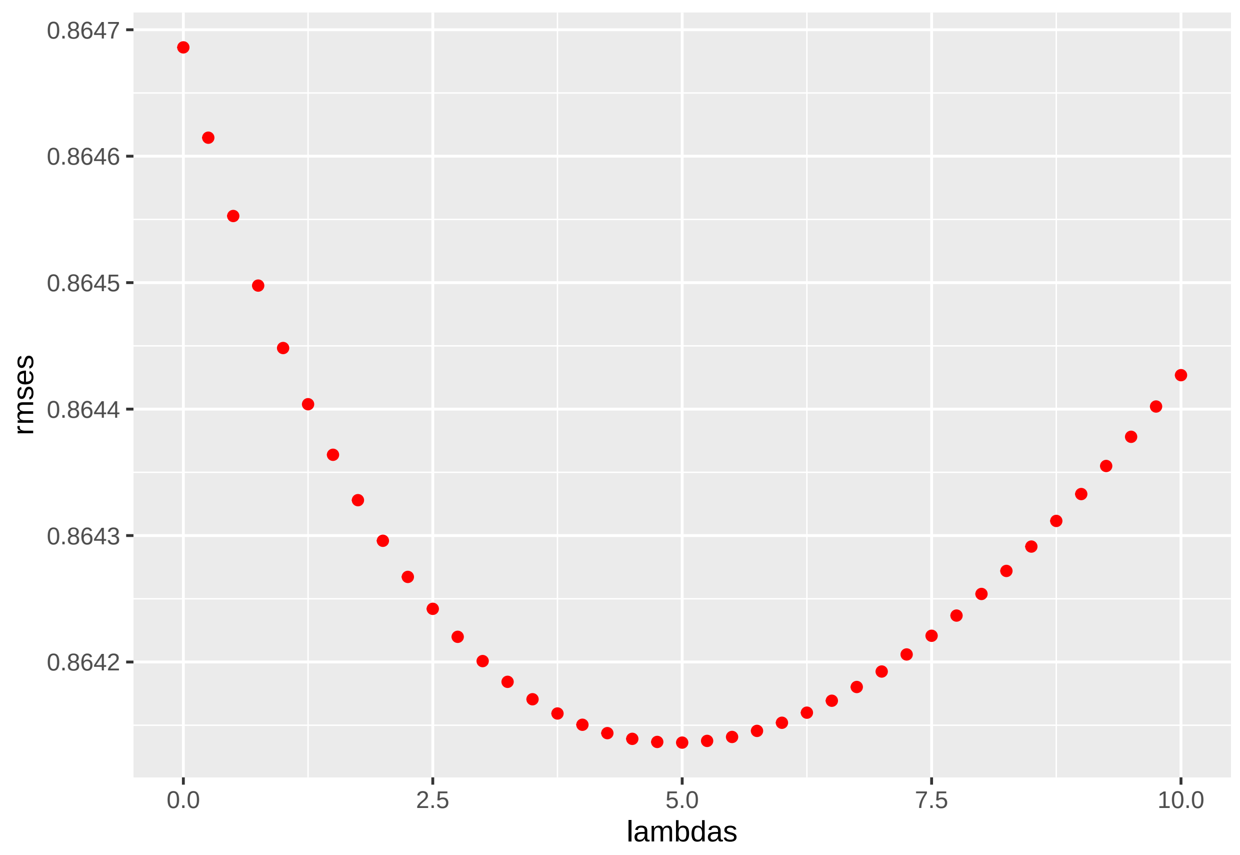
Data Regularization

Some of the data in any dataset can be “noisy”, where large estimates come from small sample sizes. In the case of this dataset, it can mean that there are ratings of obscure movies by only a few users within the dataset. Therefore, I used regularization to penalize these large estimates from small sample sizes. In addition to this, I used cross-validation to find the optimal penalty to use (denoted by lambda) and applied it to the next model generated.

```r
# Applying data regularization
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(x){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mean_rating)/(n() + x)) # adding movie bias
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i -
                          mean_rating)/(n() + x)) # adding user bias
  b_t <- train_set %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(date) %>%
    summarize(b_t = mean(rating - b_i - b_u -
                           mean_rating)/(n() + x)) # adding time bias
  predicted_ratings <- test_set %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "date") %>%
    mutate(pred = mean_rating + b_i + b_u + b_t) %>%
    pull(pred)
  return(rmse(predicted_ratings, test_set$rating))
})
```

```r
# Plotting lambdas versus RMSEs
qplot(lambdas, rmses, color = I("red"))
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



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

```
## [1] 5
```

After calculating and plotting a range of lambdas versus RMSEs, I found the optimal lambda to use on the model. This optimal lambda provides the lowest error (i.e. it is the lowest point on the graph).

```
# Selecting the lambda with the lowest RMSE
regularized_rmse <- min(rmses)
regularized_rmse
```

```
## [1] 0.8641363
```

While the RMSE did improve in a greater manner compared to the previous model, it is still quite a small improvement compared to what came before. Additionally, the RMSE is still a rounding error within the target RMSE. To make the RMSE sufficiently low, I had to switch tactics once again.

Matrix Factorization (using the recosystem package)

To make the RMSE lower, I turned to matrix factorization, which is a process by which the data is processed as a large and sparse matrix, then decomposed into two smaller dimensional matrices with less sparsity and latent features. This was accomplished by using the recosystem package. The data was converted into the recosystem format, tuned to select the best tuning parameters along a set of candidate values, trained, and finally tested by computing the predicted values.

```r
# Applying matrix factorization using the recosystem package
if(!require(recosystem)) install.packages(
  "recosystem", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: recosystem
```

```r
library(recosystem)
set.seed(1, sample.kind = "Rounding") # using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
reco_train <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))
reco_test <- with(test_set, data_memory(user_index = userId,
                                        item_index = movieId, rating = rating))
reco <- Reco()

reco_para <- reco$tune(reco_train, opts = list(dim = c(20, 30),
                                               costp_l2 = c(0.01, 0.1),
                                               costq_l2 = c(0.01, 0.1),
                                               lrate = c(0.01, 0.1),
                                               nthread = 4, niter = 10))

reco$train(reco_train, opts = c(reco_para$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9827   1.1032e+07
##    1       0.8757   8.9947e+06
##    2       0.8429   8.3407e+06
##    3       0.8206   7.9637e+06
##    4       0.8043   7.6916e+06
##    5       0.7920   7.5008e+06
##    6       0.7820   7.3567e+06
##    7       0.7735   7.2392e+06
##    8       0.7663   7.1454e+06
##    9       0.7600   7.0673e+06
##   10       0.7546   7.0039e+06
##   11       0.7495   6.9474e+06
##   12       0.7449   6.8963e+06
##   13       0.7409   6.8531e+06
##   14       0.7372   6.8146e+06
##   15       0.7338   6.7795e+06
##   16       0.7306   6.7477e+06
##   17       0.7277   6.7217e+06
##   18       0.7250   6.6962e+06
##   19       0.7225   6.6727e+06
##   20        0.7202   6.6519e+06
##   21        0.7181   6.6328e+06
##   22        0.7160   6.6155e+06
##   23        0.7141   6.5996e+06
##   24        0.7124   6.5829e+06
##   25        0.7107   6.5699e+06
##   26        0.7092   6.5564e+06
##   27        0.7077   6.5449e+06
##   28        0.7063   6.5324e+06
##   29        0.7050   6.5219e+06
```

```r
reco_first <- reco$predict(reco_test, out_memory())
```

```r
# RMSE calculated with matrix factorization
factorization_rmse <- RMSE(reco_first, test_set$rating)
factorization_rmse
```

```
## [1] 0.7845246
```

This RMSE is the best one yet, and the improvement is significant enough that it can be seen as sufficiently lower than the target RMSE. Therefore, this is the process by which I conducted the final holdout test.

Final Holdout Test

Because I determined that this method would be the ideal method to conduct the final holdout test with, I trained the model using the edx set and tested it using the final_holdout_test set.

```r
# Using matrix factorization on final holdout test
set.seed(1, sample.kind = "Rounding") # using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
reco_edx <- with(edx, data_memory(user_index = userId, item_index = movieId,
                                  rating = rating))
reco_final_holdout <- with(final_holdout_test, data_memory(user_index = userId,
                                                           item_index = movieId,
                                                           rating = rating))
reco <- Reco()

reco_para <- reco$tune(reco_edx, opts = list(dim = c(20, 30),
                                             costp_l2 = c(0.01, 0.1),
                                             costq_l2 = c(0.01, 0.1),
                                             lrate = c(0.01, 0.1),
                                             nthread = 4, niter = 10))

reco$train(reco_edx, opts = c(reco_para$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9730   1.2017e+07
##    1        0.8739   9.9025e+06
##    2        0.8401   9.1935e+06
##    3        0.8176   8.7588e+06
##    4        0.8016   8.4794e+06
##    5        0.7895   8.2754e+06
##    6        0.7798   8.1249e+06
##    7        0.7718   8.0024e+06
##    8        0.7649   7.9083e+06
##    9        0.7592   7.8268e+06
##   10        0.7541   7.7623e+06
##   11        0.7494   7.7016e+06
##   12        0.7452   7.6493e+06
##   13        0.7415   7.6103e+06
##   14        0.7381   7.5707e+06
##   15        0.7350   7.5381e+06
##   16        0.7322   7.5065e+06
##   17        0.7296   7.4776e+06
##   18        0.7272   7.4516e+06
##   19        0.7249   7.4304e+06
##   20        0.7229   7.4100e+06
##   21        0.7209   7.3905e+06
##   22        0.7191   7.3729e+06
##   23        0.7175   7.3593e+06
##   24        0.7159   7.3423e+06
##   25        0.7144   7.3299e+06
##   26        0.7130   7.3158e+06
##   27        0.7117   7.3046e+06
##   28        0.7104   7.2930e+06
##   29        0.7092   7.2832e+06
```

```
reco_final <- reco$predict(reco_final_holdout, out_memory())
```

```
# Generating final RMSE
final_rmse <- RMSE(reco_final, final_holdout_test$rating)
final_rmse
```

```
## [1] 0.7813772
```

This is the final RMSE, generated using the final_holdout_test dataset. No more testing occured after this.

Final Results

The final results of the data analysis can be seen in the table generated below:

```
# Table made using the reactable package
if(!require(reactable)) install.packages("reactable",
                                          repos = "http://cran.us.r-project.org")
```

```
## Loading required package: reactable
```

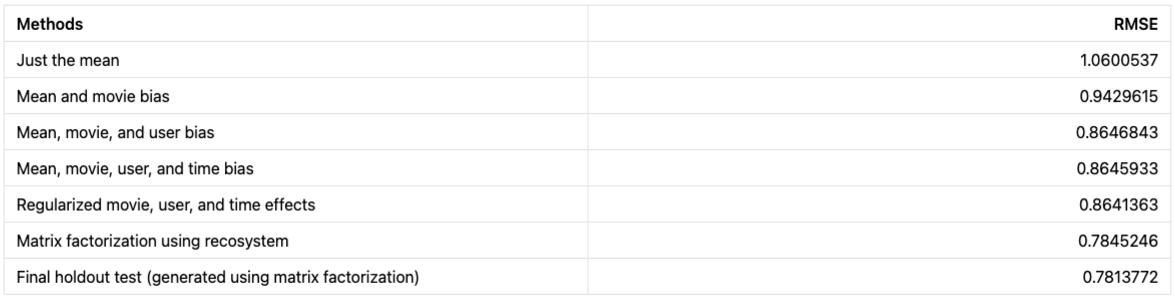
```r
library(reactable)
if(!require(webshot2)) install.packages("webshot2",
                                         repos = "http://cran.us.r-project.org")
```

```
## Loading required package: webshot2
```

```r
library(webshot2)
if(!require(htmlwidgets)) install.packages("htmlwidgets",
                                            repos = "http://cran.us.r-project.org")
```

```
## Loading required package: htmlwidgets
```

```r
library(htmlwidgets)
Methods <- c("Just the mean", "Mean and movie bias",
             "Mean, movie, and user bias", "Mean, movie, user, and time bias",
             "Regularized movie, user, and time effects",
             "Matrix factorization using recosystem",
             "Final holdout test
             (generated using matrix factorization)") # first column
RMSE <- c(round(mean_rmse, 7), round(movie_bias_rmse, 7),
          round(user_bias_rmse, 7), round(time_bias_rmse, 7),
          round(regularized_rmse, 7), round(factorization_rmse, 7),
          round(final_rmse, 7)) # second column
final_results <- data.frame(Methods, RMSE)
table <- reactable(final_results,
  highlight = TRUE,
  bordered = TRUE,
  theme = reactableTheme(
    borderColor = "#dfe2e5",
    highlightColor = "#f0f5f9",
    cellPadding = "8px 12px",
    style = list(fontFamily = "-apple-system, BlinkMacSystemFont,
                 Segoe UI, Helvetica, Arial, sans-serif"),
    )
  )
saveWidget(widget = table, file = "table_html.html", selfcontained = TRUE)
webshot(url = "table_html.html", file = "final_table.png", delay = 0.1,
        vwidth = 1245)
```



Methods	RMSE
Just the mean	1.0600537
Mean and movie bias	0.9429615
Mean, movie, and user bias	0.8646843
Mean, movie, user, and time bias	0.8645933
Regularized movie, user, and time effects	0.8641363
Matrix factorization using recosystem	0.7845246
Final holdout test (generated using matrix factorization)	0.7813772

As can be observed, the final RMSE generated was significantly lower than the target RMSE of 0.86490. I tested several models, but achieved the best results using matrix factorization with the recosystem package. Therefore, matrix factorization, at least concerning this model, was the best option which provided an optimal movie recommendation system by the metrics tested by this project, that being the RMSE.

Conclusion

To conclude this project, I set out to make a movie recommendation software using machine learning algorithms. This was accomplished by using data exploration to better understand the data I was working with, and then data analysis in order to eliminate various types of biases in movie rating, regularize the data, and finally utilize matrix factorization in order to make the best movie recommendation system that I could. Keeping in mind that the RMSE goal set was 0.86490, I achieved an RMSE lower than this by a not insignificant amount, which I find to be a success.

However, some limitations of the project include factoring in time bias. This increased the runtime of the program by a significant amount, for not much gain in return, in terms of how much the RMSE was lowered. In general, the runtime of the code could be reduced in a number of places, but this was the biggest detractor in that regard. In addition to this, to state the obvious, this project is simply the first step in making software that can actually predict the ratings of movies before they even come out.

In the future, this project can be used as a shell or stepping stone for other machine learning algorithms to predict other forms of ratings. For instance, it can be used to predict the ratings of books or video games. However, it could go even further than that, and be used for other predictions for any topic imaginable. This can all be accomplished by using the methods employed in this project as a base.