MovieLens Recommendation System

Justin Farnsworth

6/2/2020

Summary

Every day, the demand for data and artifical intelligence is growing. Automation is becoming almost a necessity in today's markets. Predicting outcomes accuracy is also desired, however the results aren't always perfect. Despite this, companies collect data which is then used to implement such systems. In the context of streaming services such as Netflix and Hulu, user ratings for movies are used to build movie reccomendation systems.

In this report, the goal was to **implement a movie reccomendation system** using the MovieLens dataset, which contains about 10 million user ratings. The ratings are represented using a 5-star system, from 1 being the worst rating to 5 being the best. In the dataset, we are also given the movie IDs, the user IDs, the movie title (with the release year attached to them), the genre(s), and the time in which was rating was given. In our analysis, we were able to discover key trends in out dataset's features, such as the variability of ratings across users and genres.

Using these patterns, we implemented numberous models to predict the movie ratings. We experimented with the various features to see how much of an effect they had on the RMSE. To prevent overfitting, we split the dataset into a training dataset (edx) and a test set (validation). All records in the validation set are also in the edx dataset to ensure we can properly predict their ratings. The edx dataset consisted of approximately 90% of MovieLens dataset, or 9 million records. The validation set consisted of the of the other 10%, or nearly 1 million records.

We also used regularization to help improve our prediction. Using this method, we were able to achieve a **residual mean squared error (RMSE) of 0.8644229** using a regularized model of the movie, user, release year, and genre effects. However, to implement this model, the analysis resulted in roughly 25GB of RAM. Therefore, devices that do not have sufficient memory will crash. Nevertheless, the results are shared in this document.

Each section has their methods and models explained, followed by their respective results.

The dataset can be accessed here: https://grouplens.org/datasets/movielens/10m/

Analysis

Before conducting the analsis, the dataset was reformatted so that the `timestamp` displayed the date and time of the rating. Also, the release year was extracted from the `title` and placed into its own column named `releaseYear`. The `releaseYear` is one of the features used to implement the recommendation system, including `movieId`, `userId`, and `genres`.

When exploring the dataset, we focused on the movies, users, ratings, genres, and release years categorically. The other columns were not analyzed directly for this project.

For the models, we started with predicting just the average. The goal was to reduce the RMSE generated from this by adding more features. The four features used were `movieId`, `userId`, `releaseYear`, and `genres`. Each model using these features were regularized and tuned to help improve the RMSE even further by obtaining the λ with the lowest RMSE.

Exploring the Dataset - Overview

Since the validation set's records are also in the edx dataset, we can just analyze the edx set alone.

The rows and columns of the dataset respectively are:

```r
# Dimensions of the dataset
dim(edx)
```

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

Here are the first 10 rows of the dataset:

```r
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
```

While the data appears to be tidy, we can see that some information can be reformatted or extracted into their own columns. The `timestamp` represents the number of seconds that have passed since the Unix Epoch (January 1, 1970, 12:00:00AM). To understand when the ratings have been recorded, this should be converted to datetime. We also see that the movie `title` and release year are in the same column. It would be ideal to have the `title` and the release year in their own columns.

The datatypes for each column are:

```
##   column_names data_type
## 1       userId   integer
## 2      movieId   numeric
## 3       rating   numeric
## 4    timestamp   integer
## 5        title character
## 6       genres character
```

Cleaning Up the Dataset

First, we want to check for any null values.

```
# Check for any null values
any(is.na(edx))
```

```
## [1] FALSE
```

```
any(is.na(validation))
```

```
## [1] FALSE
```

Since there are no null cells, we can clean up the dataset. We will convert the `timestamp` into a datetime (we will rename the column to `dateTimeOfRating`) and split the `title` column into two columns: `title` and `releaseYear`.

NOTE: We will have to do this for the validation dataset as well.

After cleanup, the first 10 rows look like this:

```
##    userId movieId rating                                     genres
## 1       1     122      5                             Comedy|Romance
## 2       1     185      5                      Action|Crime|Thriller
## 3       1     292      5               Action|Drama|Sci-Fi|Thriller
## 4       1     316      5                    Action|Adventure|Sci-Fi
## 5       1     329      5              Action|Adventure|Drama|Sci-Fi
## 6       1     355      5                    Children|Comedy|Fantasy
## 7       1     356      5                   Comedy|Drama|Romance|War
## 8       1     362      5                 Adventure|Children|Romance
## 9       1     364      5 Adventure|Animation|Children|Drama|Musical
## 10      1     370      5                              Action|Comedy
##                               title releaseYear    dateTimeOfRating
## 1                         Boomerang        1992 1996-08-02 07:24:06
## 2                          Net, The        1995 1996-08-02 06:58:45
## 3                          Outbreak        1995 1996-08-02 06:57:01
## 4                          Stargate        1994 1996-08-02 06:56:32
## 5              Star Trek: Generations        1994 1996-08-02 06:56:32
## 6                  Flintstones, The        1994 1996-08-02 07:14:34
## 7                      Forrest Gump        1994 1996-08-02 07:00:53
## 8                  Jungle Book, The        1994 1996-08-02 07:21:25
## 9                    Lion King, The        1994 1996-08-02 07:01:47
## 10 Naked Gun 33 1/3: The Final Insult        1994 1996-08-02 07:16:36
```

Exploring the Dataset - Movies

Based on the number of movie IDs, there are 10677 movies in the dataset.

Here are the 10 movies with the most ratings:

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                 count
```

```
##      <dbl> <chr>                                                    <int>
##  1     296 Pulp Fiction                                             31362
##  2     356 Forrest Gump                                             31079
##  3     593 Silence of the Lambs, The                                30382
##  4     480 Jurassic Park                                            29360
##  5     318 Shawshank Redemption, The                                28015
##  6     110 Braveheart                                               26212
##  7     457 Fugitive, The                                            25998
##  8     589 Terminator 2: Judgment Day                               25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) 25672
## 10     150 Apollo 13                                                24284
## # ... with 10,667 more rows
```

We see that `Pulp Fiction` has the most ratings. However, this doesn't give us any information on what those ratings are, or how good the movie is compared to others.

Here are the top 10 movies based on average ratings (minimum of 1,000 ratings):

```
## # A tibble: 1,902 x 4
## # Groups:   movieId [1,902]
##    movieId title                                  avg_rating count
##      <dbl> <chr>                                       <dbl> <int>
##  1     318 Shawshank Redemption, The                    4.46 28015
##  2     858 Godfather, The                               4.42 17747
##  3      50 Usual Suspects, The                          4.37 21648
##  4     527 Schindler's List                             4.36 23193
##  5     912 Casablanca                                   4.32 11232
##  6     904 Rear Window                                  4.32  7935
##  7     922 Sunset Blvd. (a.k.a. Sunset Boulevard)       4.32  2922
##  8    1212 Third Man, The                               4.31  2967
##  9    3435 Double Indemnity                             4.31  2154
## 10    1178 Paths of Glory                               4.31  1571
## # ... with 1,892 more rows
```

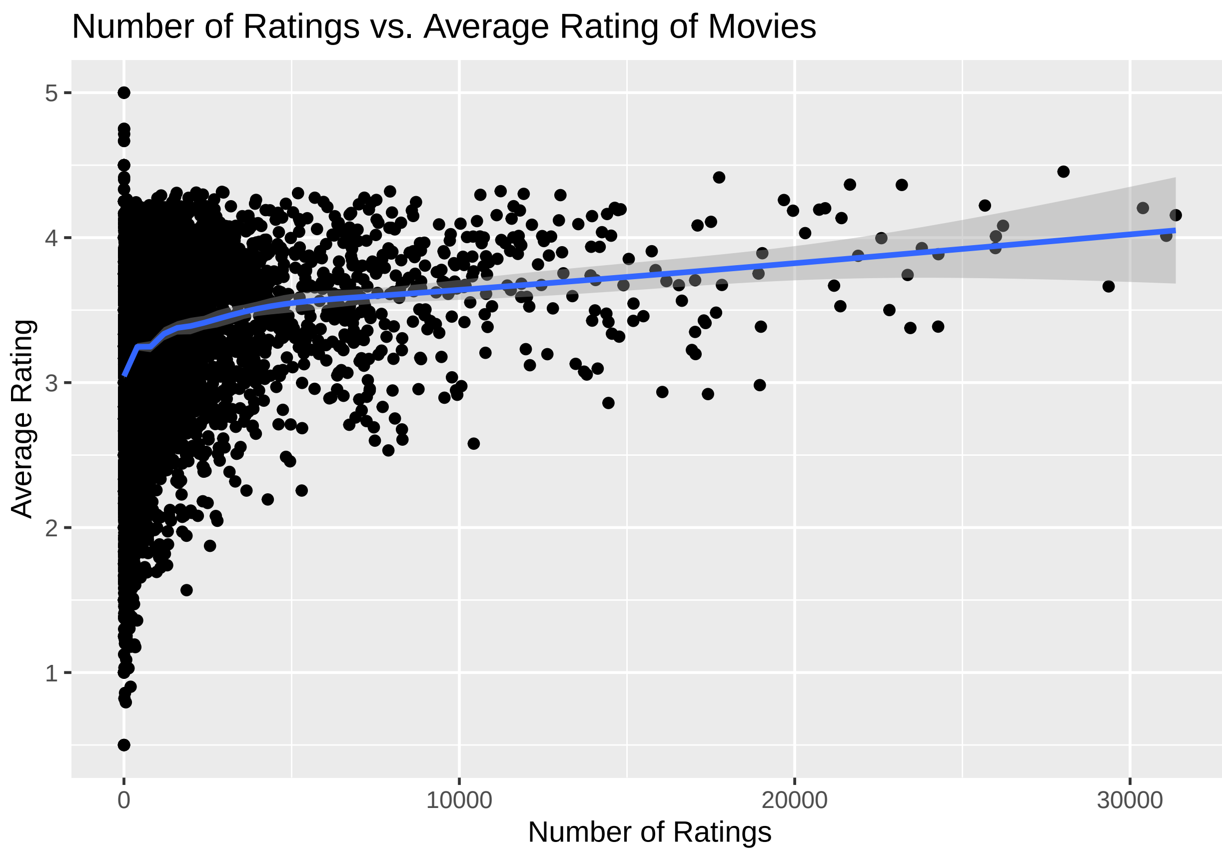
Notice that while `Pulp Fiction` had more ratings, they are not in the top 10 by averages. In fact, only one of the movies from the previous chart are in the top 10 highest averages. The only movie that made the top 10 in both tables was `The Shawshank Redemption`, which topped the charts by averages.

Observe the bottom 10 movies with a minimum of 1,000 ratings:

```
## # A tibble: 1,902 x 4
## # Groups:   movieId [1,902]
##    movieId title                                     avg_rating count
##      <dbl> <chr>                                          <dbl> <int>
##  1    3593 Battlefield Earth                               1.57  1869
##  2    2383 Police Academy 6: City Under Siege              1.72  1092
##  3    1760 Spice World                                     1.74  1288
##  4    2382 Police Academy 5: Assignment: Miami Beach       1.78  1111
##  5    1389 Jaws 3-D                                        1.79  1052
##  6    1707 Home Alone 3                                    1.82  1221
##  7    1556 Speed 2: Cruise Control                         1.87  2566
##  8     181 Mighty Morphin Power Rangers: The Movie         1.88  1316
##  9    5452 Look Who's Talking Now                          1.88  1059
## 10    1381 Grease 2                                        1.94  1866
## # ... with 1,892 more rows
```

Based on the top 10 and bottom 10 movies, the top 10 movies tend to have much more ratings than those in the bottom top 10. However, let's plot the number of ratings and the average rating for each movie:

```
# Calculate the correlation between the average ratings and the number of ratings
movie_avg_ratings_and_counts <- edx %>%
  group_by(movieId) %>%
  summarize(avg_rating = mean(rating), count = n()) %>%
  select(avg_rating, count)

movie_avg_ratings_and_counts %>%
  ggplot(aes(count, avg_rating)) +
  geom_point() +
  geom_smooth(method = "gam", formula = y ~ s(x, bs = "cs")) +
  ggtitle("Number of Ratings vs. Average Rating of Movies") +
  xlab("Number of Ratings") +
  ylab("Average Rating")
```



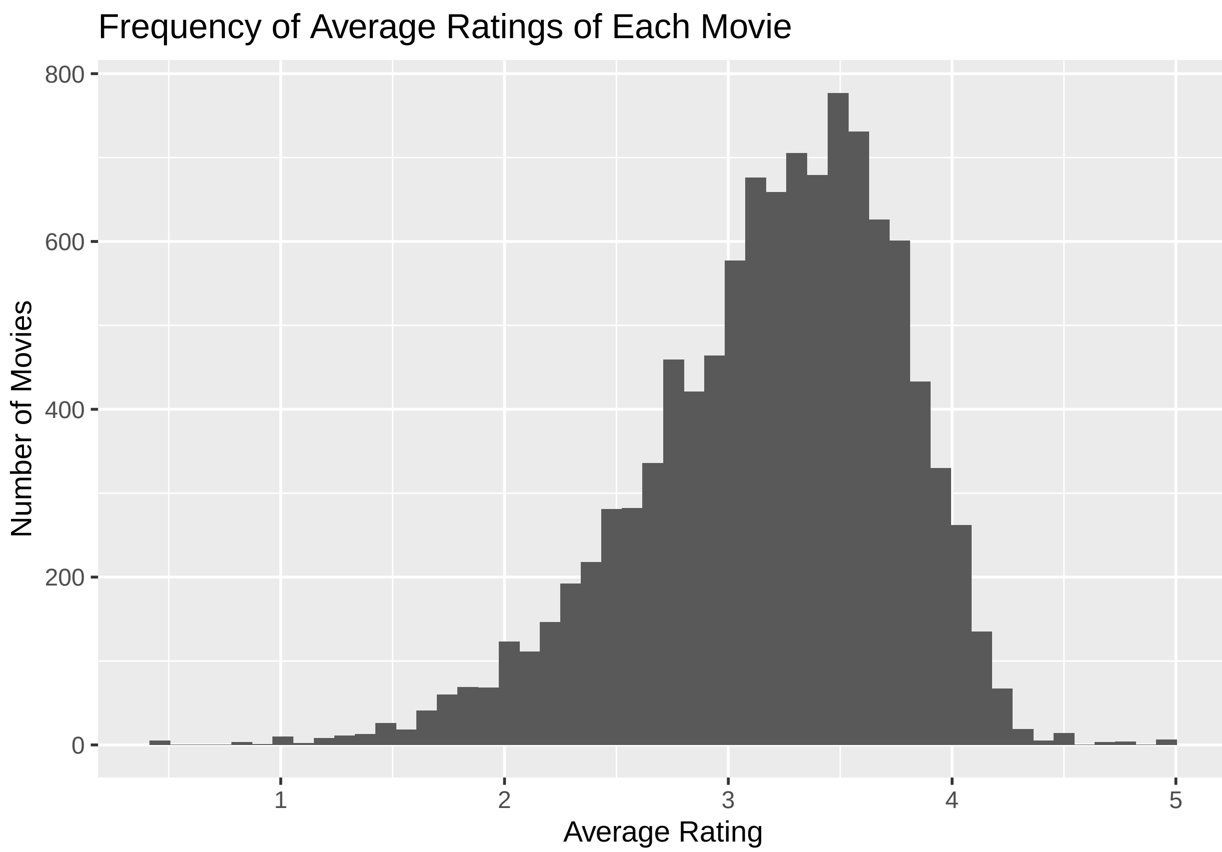
Surprisingly, there seems to be a trend across the entire dataset. Movies with more ratings tend to have higher ratings on average. However, based on the correlation coefficient, the correlation is weakly correlated.

```
cc <- movie_avg_ratings_and_counts %>%
  summarize(r = cor(count, avg_rating)) %>%
  .$r
```

The correlation coefficient between the number of ratings and the average rating is **0.2114161**.

Here is the distribution of the average ratings for each movie:

```r
# Plot the average ratings for each movie
edx %>%
  group_by(movieId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(bins = 50) +
  ggtitle("Frequency of Average Ratings of Each Movie") +
  xlab("Average Rating") +
  ylab("Number of Movies")
```



The distribution appears to be skewed to the left. We cannot say that the distribution is approximately normal.

Now let's see the top median movie ratings:

```
## # A tibble: 1,902 x 4
## # Groups:   movieId [1,902]
##    movieId title                                                  median count
##      <dbl> <chr>                                                   <dbl> <int>
##  1     318 Shawshank Redemption, The                                 5   28015
##  2     858 Godfather, The                                            5   17747
##  3      50 Usual Suspects, The                                       4.5 21648
##  4     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)     4.5 25672
##  5     296 Pulp Fiction                                              4.5 31362
##  6     527 Schindler's List                                          4.5 23193
##  7     720 Wallace & Gromit: The Best of Aardman Animation           4.5  3908
```

```
## 8      745 Wallace & Gromit: A Close Shave                                4.5  5690
## 9      750 Dr. Strangelove or: How I Learned to Stop Worrying and ~    4.5 10627
## 10     904 Rear Window                                                    4.5  7935
## # ... with 1,892 more rows
```

We see `The Shawshank Redemption` topping the charts for median averages. We also see movies like `Schindler's List`, `The Godfather`, and `Rear Window` from the top 10 averages as well. Surprisingly, we see `Pulp Fiction` reach the top 5 median ratings, despite the fact that it wasn't in the top 10 average ratings.

Here are the bottom 10 median movie ratings:

```
## # A tibble: 1,902 x 4
## # Groups:   movieId [1,902]
##    movieId title                                      median count
##      <dbl> <chr>                                       <dbl> <int>
##  1    3593 Battlefield Earth                             1    1869
##  2    1707 Home Alone 3                                  1.5  1221
##  3    1760 Spice World                                   1.5  1288
##  4    2382 Police Academy 5: Assignment: Miami Beach     1.5  1111
##  5    2383 Police Academy 6: City Under Siege            1.5  1092
##  6      66 Lawnmower Man 2: Beyond Cyberspace            2    1372
##  7     169 Free Willy 2: The Adventure Home              2    1630
##  8     181 Mighty Morphin Power Rangers: The Movie       2    1316
##  9     193 Showgirls                                     2    3654
## 10     502 Next Karate Kid, The                          2    1719
## # ... with 1,892 more rows
```

Here, we see `Battlefield Earth`, `Home Alone 3`, `Spice World`, and several other movies appear in the lowest median and mean ratings, with `Battlefield Earth` having the worst mean and median ratings.

Exploring the Dataset - Users

Based on the number of user IDs, there are 69878 users in the dataset.

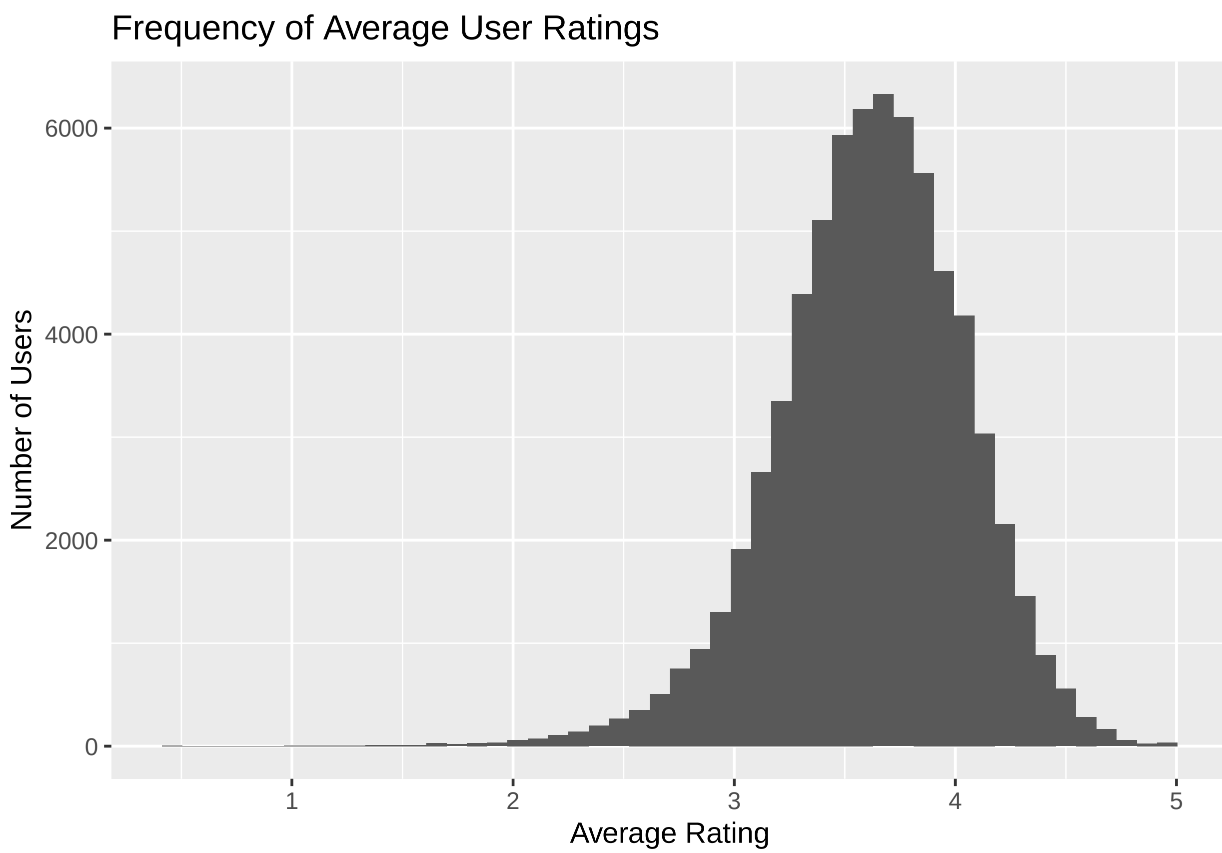
Here are the top 10 user IDs that have sumbitted the most ratings:

```
## # A tibble: 10 x 2
##    userId count
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
```

While all of the users in the top 10 have rated over 3,000 movies, two of them have rated over 6,000 movies! That's more than half the movies in the dataset!

Here is the distribution of the average user ratings. Notice how they're approximately normal:

```r
# Plot the average rating for each user
avg_of_user_avgs <- edx %>%
  group_by(userId) %>%
  summarize(avg_rating = mean(rating))

avg_of_user_avgs %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(bins = 50) +
  ggtitle("Frequency of Average User Ratings") +
  xlab("Average Rating") +
  ylab("Number of Users")
```



We can see that the majority of the average user ratings are somewhere between 3 and 4 stars. There are some users who tend to really like all the movies they watched and there are some who tend to really dislike all the movies they watched. However, the mean and standard deviation of the average user ratings are the following:

```
## # A tibble: 1 x 2
##    mean st_dev
##   <dbl>  <dbl>
## 1  3.61  0.431
```

Exploring the Dataset - Release Years

The range of release years in the dataset span from `min(edx$releaseYear)` to `max(edx$releaseYear)`. Certainly this is a wide range of movies. However, here are the years that had the most movie releases:

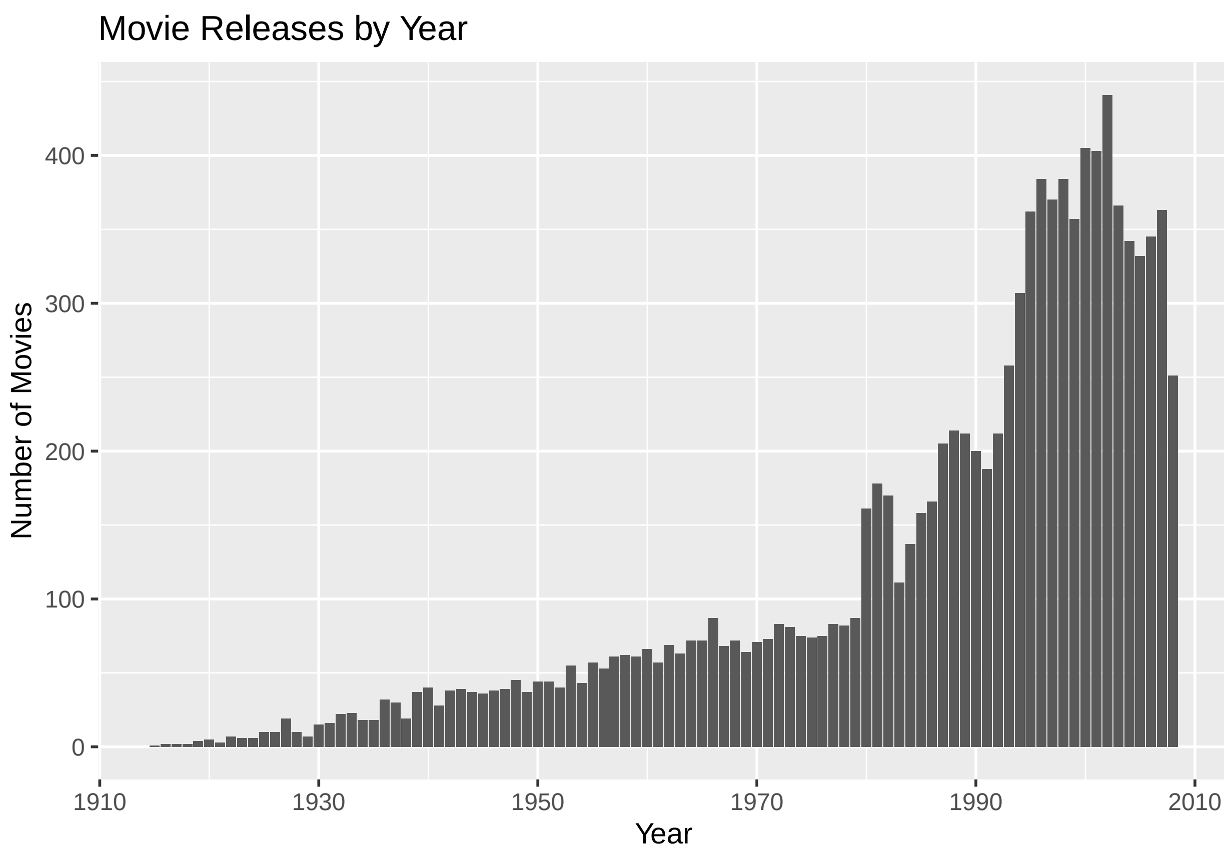
```
# Show the year that had the most movie releases
number_of_releases_by_year <- edx %>%
  select(movieId, releaseYear) %>%
  unique() %>%
  group_by(releaseYear) %>%
  summarize(count = n())

number_of_releases_by_year %>%
  arrange(desc(count)) %>%
  top_n(10)
```

```
## Selecting by count
```

```
## # A tibble: 10 x 2
##    releaseYear count
##          <int> <int>
##  1        2002   441
##  2        2000   405
##  3        2001   403
##  4        1996   384
##  5        1998   384
##  6        1997   370
##  7        2003   366
##  8        2007   363
##  9        1995   362
## 10        1999   357
```

Based on the top 10, more movies have been released in more recent years than in previous years. To visualize this, the following plot shows the number of movie releases for each year:

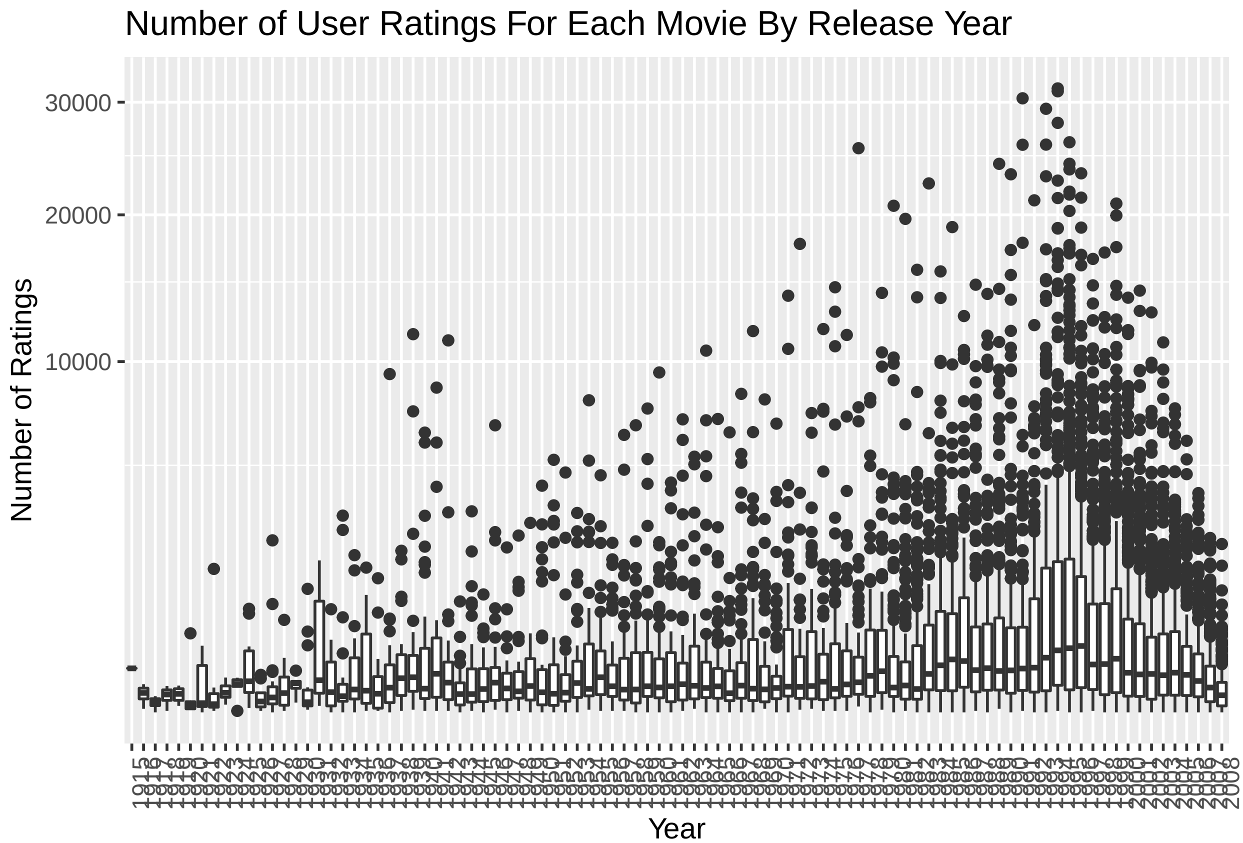
```
# Plot the number of releases by year
number_of_releases_by_year %>%
  ggplot(aes(releaseYear, count)) +
  geom_bar(stat = "identity") +
  ggtitle("Movie Releases by Year") +
  xlab("Year") +
  ylab("Number of Movies")
```



We can see that the number of movies released increased substantially since the 1980s. The dataset suggests that from the early 1980s to the early 2000s, the number of movies have approximately doubled.

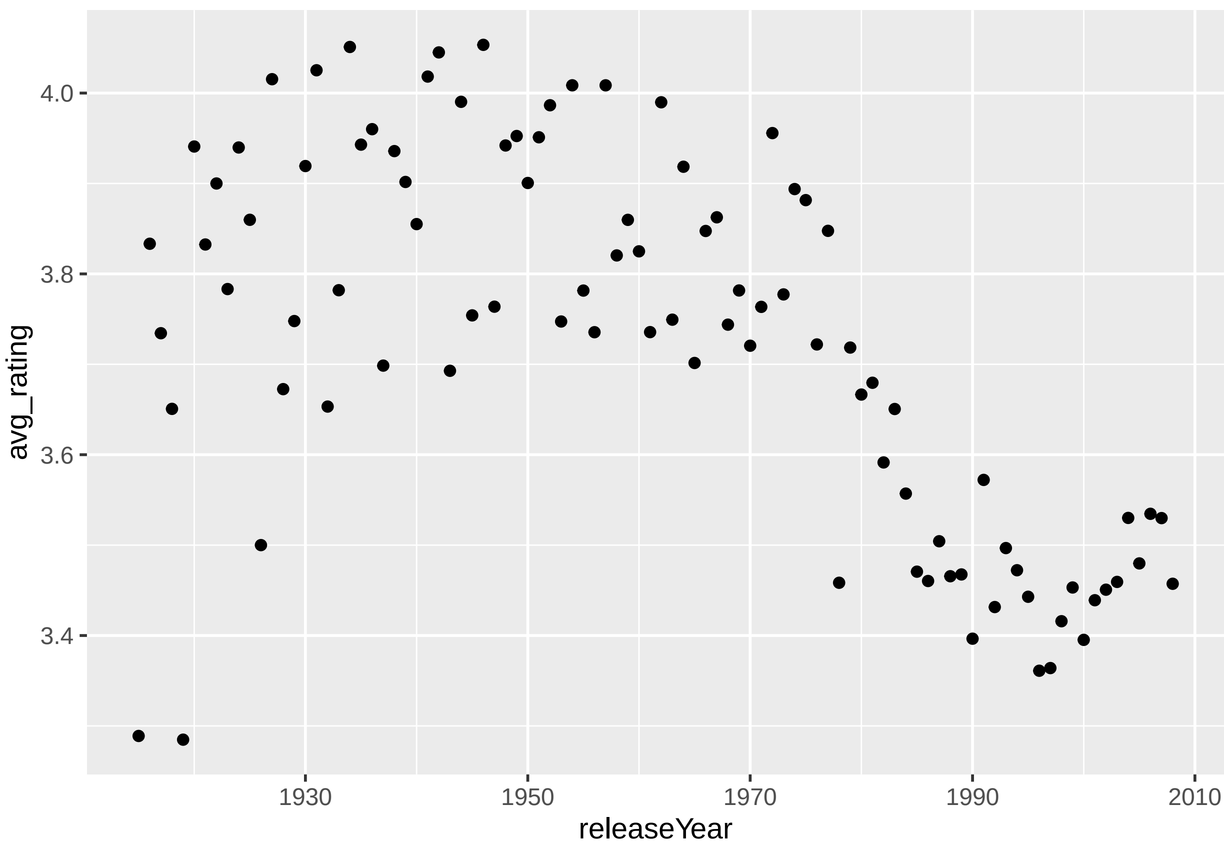
The following plot shows the number of user ratings for each movie, based on their release year. We can see that newer movies tend to get more ratings:

```
# Plot the boxplots of the total ratings for each movie by year
edx %>%
  group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(releaseYear))) %>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Number of User Ratings For Each Movie By Release Year") +
  xlab("Year") +
  ylab("Number of Ratings")
```



In the following plot, we calculate the average rating of movies within their respective release years. Interestingly, the average rating of movies before 1980 tend to be higher. However, the previous plot shows they don't have as many ratings as newer movies.

```
# Plot the average rating by release year
edx %>%
  select(releaseYear, rating) %>%
  group_by(releaseYear) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(releaseYear, avg_rating)) +
  geom_point()
```



Exploring the Dataset - Genres

While movies can have multiple genres, their combinations are made up of the following genres:

```
# Show the list of genres
list_of_genres <- edx %>%
  select(genres) %>%
  distinct() %>%
  separate_rows(genres, sep = "\\|") %>%
  distinct() %>%
  arrange(genres) %>%
  .$genres

list_of_genres
```

```
##  [1] "(no genres listed)" "Action"             "Adventure"
##  [4] "Animation"          "Children"           "Comedy"
##  [7] "Crime"              "Documentary"        "Drama"
## [10] "Fantasy"            "Film-Noir"          "Horror"
## [13] "IMAX"               "Musical"            "Mystery"
## [16] "Romance"            "Sci-Fi"             "Thriller"
## [19] "War"                "Western"
```

While the dataset suggests there are 20 genres, we can see that one of the genres is `(no genres listed)`. We are curious to know which movie(s) do not have a genre.

The following movie(s) without a genre are:

```
##   movieId         title releaseYear
## 1    8606 Pull My Daisy        1958
```

Interestingly, `Pull My Daisy` is a short film that is classified as Comedy by Rotten Tomatoes. See https://www.rottentomatoes.com/m/pull_my_daisy for more information about the movie. (Note that the movie's release year is listed as 1959 while the dataset says it was released in 1958.)

Here are the number of movies that are in each genre:

```
# Count the number of movies for each genre
# NOTE: movies can have more than one genre
movies <- edx %>%
  select(movieId, title, releaseYear, genres) %>%
  unique()

data.frame(genre = list_of_genres,
           count = map_dbl(list_of_genres, function(genre){
             sum(str_detect(movies$genres, genre))
           })
) %>% arrange(desc(count))
```

```
##                 genre count
## 1               Drama  5336
## 2              Comedy  3703
## 3            Thriller  1705
## 4             Romance  1685
## 5              Action  1473
## 6               Crime  1117
## 7           Adventure  1025
## 8              Horror  1013
## 9              Sci-Fi   754
## 10            Fantasy   543
## 11           Children   528
## 12                War   510
## 13            Mystery   509
## 14        Documentary   481
## 15            Musical   436
## 16          Animation   286
## 17            Western   275
## 18          Film-Noir   148
## 19               IMAX    29
## 20 (no genres listed)     1
```

We see that the most common genre is `Drama`, while `Comedy` is the next prevalent genre.

Here are the number of ratings for each genre. Note that since movies can have multiple genres, the movies may be counted more than once.

```
# Count the number of ratings for each genre
data.frame(genre = list_of_genres,
           count = map_dbl(list_of_genres, function(genre){
           sum(str_detect(edx$genres, genre))
})) %>% arrange(desc(count))
```

```
##                  genre   count
## 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
## 11              Horror  691485
## 12             Mystery  568332
## 13                 War  511147
## 14           Animation  467168
## 15             Musical  433080
## 16             Western  189394
## 17           Film-Noir  118541
## 18         Documentary   93066
## 19                IMAX    8181
## 20 (no genres listed)       7
```

It appears that the 3 most rated genres are `Drama`, `Comedy`, and `Action`. It's no surprise that `Drama` and `Comedy` remain in the top 2 since they're the most common genres in the dataset. However, the prevalence of movies with these genres and number of ratings for those genres do not indicate whether people view the movies of those genres favorably. To answer this question, we will calculate the average rating and standard error for each genre to see what ratings the movies with those genres are receiving collectively.

Here are the average ratings by genre: (note that we removed `(no genres listed)`)

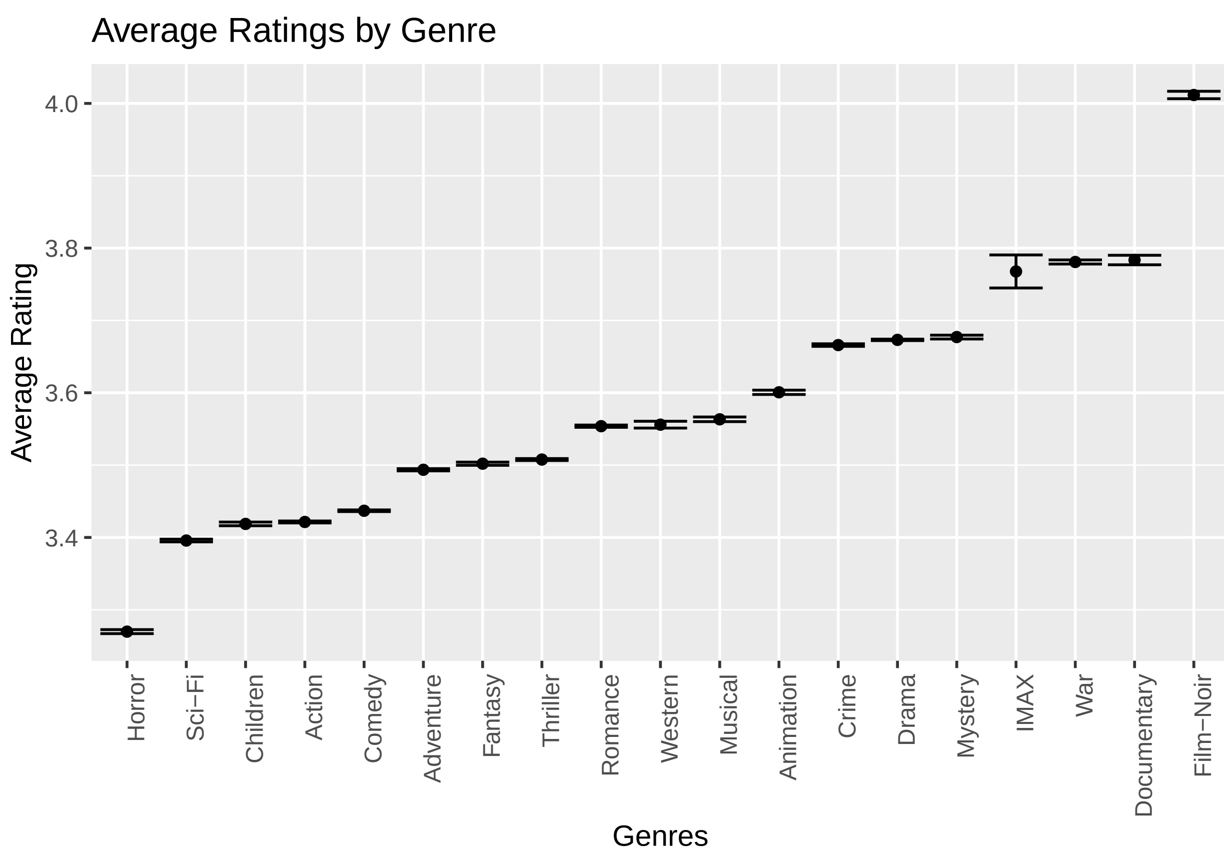
```
# Find the average and SE of the ratings for each genre
avg_rating_by_genre <- map_df(list_of_genres, function(genre){
  edx %>%
    select(genres, rating) %>%
    filter(str_detect(genres, genre)) %>%
    summarize(genres = genre, avg_rating = mean(rating), se = sd(rating)/sqrt(n()))
}) %>%
  filter(genres != "(no genres listed)") %>%
  arrange(desc(avg_rating))

avg_rating_by_genre
```

```
##         genres avg_rating           se
## 1    Film-Noir   4.011625 0.0025767396
## 2  Documentary   3.783487 0.0032905826
## 3          War   3.780813 0.0014152178
## 4         IMAX   3.767693 0.0114132659
## 5      Mystery   3.677001 0.0013268232
## 6        Drama   3.673131 0.0005033857
## 7        Crime   3.665925 0.0008781925
## 8    Animation   3.600644 0.0014913317
## 9      Musical   3.563305 0.0016059709
## 10     Western   3.555918 0.0023524102
## 11     Romance   3.553813 0.0007874936
```

```
## 12    Thriller   3.507676 0.0006761236
## 13     Fantasy   3.501946 0.0011074351
## 14   Adventure   3.493544 0.0007620983
## 15      Comedy   3.436908 0.0005710956
## 16      Action   3.421405 0.0006665433
## 17    Children   3.418715 0.0012716112
## 18      Sci-Fi   3.395743 0.0009435986
## 19      Horror   3.269815 0.0013828957
```

A visualization of the data above is presented below:

```r
# Plot boxplots of ratings for each genre
avg_rating_by_genre %>%
  mutate(genres = reorder(genres, avg_rating)) %>%
  ggplot(aes(genres, avg_rating, ymin = avg_rating - 2 * se, ymax = avg_rating + 2 * se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Average Ratings by Genre") +
  xlab("Genres") +
  ylab("Average Rating")
```



We can see that the most common genres don't have the highest average ratings. More surprisingly, `Comedy` ranks lower than most of the other genres. The only genre that surpassed a 4.0 rating was `Film-Noir`, however, not a lot of movies have this genre. The genre with the lowest average ratings was `Horror`, which was the only genre to fall under a 3.3 average rating.

Exploring the Dataset - Ratings

The following table shows the different types of ratings, ordered from most prevalent to least:

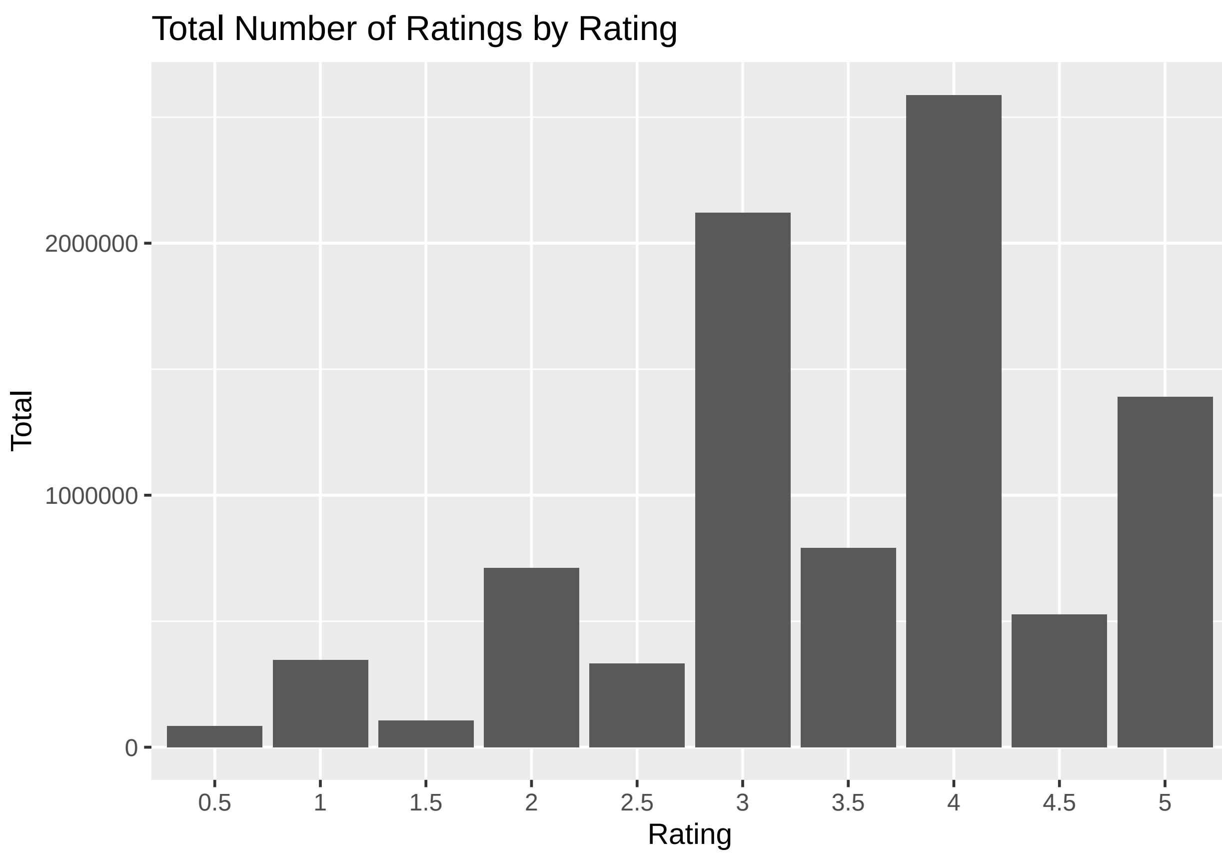
```
# Order the most commonly given ratings from greatest to least
frequency_of_ratings <- edx %>%
  select(rating) %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

frequency_of_ratings
```

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

It appears that the 3 most common ratings are all whole-star ratings: 4.0, 3.0, and 5.0. Similarly, the 3 least common ratings are partial-star ratings: 0.5, 1.5, and 2.5.

A histogram of the ratings is shown below:

```
# Plot the totals of each rating
frequency_of_ratings %>%
  mutate(rating = factor(rating)) %>%
  group_by(rating) %>%
  ggplot(aes(rating, count, label = rating)) +
  geom_bar(stat = "identity") +
  ggtitle("Total Number of Ratings by Rating") +
  xlab("Rating") +
  ylab("Total") +
  scale_y_continuous(labels = function(y) format(y, scientific = FALSE))
```



We observe that the whole-star ratings are much more prevalent. The total number of ratings that whole-star ratings and those that are not, along with their proportions, are:

```
## # A tibble: 2 x 3
##   whole_star_rating   count proportion
##   <chr>               <int>      <dbl>
## 1 No                1843170      0.205
## 2 Yes               7156885      0.795
```

This leaves us wondering why this phenomenon is happening. A further analysis of the rating types by year show even more surprising results:

```
# Show number of ratings by year and by whether it is a whole-star rating or not
edx %>%
  select(dateTimeOfRating, rating) %>%
  mutate(whole_star_rating = ifelse(rating %in% 1:5, "Yes", "No")) %>%
  mutate(yearOfRating = year(dateTimeOfRating)) %>%
  group_by(yearOfRating, whole_star_rating) %>%
  summarize(count = n()) %>%
  spread(whole_star_rating, count)
```

```
## # A tibble: 15 x 3
## # Groups:   yearOfRating [15]
##    yearOfRating     No     Yes
##           <int>  <int>   <int>
```

```
## 1           1995      NA       2
## 2           1996      NA  943072
## 3           1997      NA  413849
## 4           1998      NA  181586
## 5           1999      NA  710105
## 6           2000      NA 1144526
## 7           2001      NA  683961
## 8           2002      NA  524785
## 9           2003 179634  440163
## 10          2004 307349  384099
## 11          2005 469963  589165
## 12          2006 301216  387690
## 13          2007 273783  355716
## 14          2008 305683  391218
## 15          2009   5542    6948
```

The **partial-star ratings were not implemented until 2003**, which explains why there are significantly more whole-star rating than partial-star ratings. Even when the partial-star ratings were introduced, there were still more whole-star ratings than partial-star ratings each and every year.

Another observation made is that there are very few ratings in 1995 and 2009. The initial question presented was whether the dataset included all ratings for those years or not.

The earliest rating in the dataset occurred on 1995-01-09 06:46:49. Similarly, the latest rating occurred on 2009-01-05 00:02:16.

Models - Overview

After exploring the dataset, we began making models to predict the ratings of the movies based on particular effects.

Loss Function - Residual Mean Squared Error (RMSE)

Since the predictions are continuous values, we cannot determine accuracy by checking if the predicted values are exactly equal to the actual values. Instead, it is more useful to use a function that summarizes the differences overall. For this project, we used the **residual mean squared error (RMSE)** to determine acccuracy. The RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{m,u}(\hat{y}_{m,u} - y_{m,u})^2}$$

Where N is the number of predictions, m represents the movie, u represents the user, $\hat{y}_{m,u}$ represents the predicted value, and $y_{m,u}$ represents the actual value. Note that as the predictions become more accurate, the RMSE gets closer to 0.

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

The goal was to produce a model in which the RMSE is less than 0.8649. This is considered to be very good for our movie recommendation system.

Models - Just the Average

A simple but undesirable model is to compute the average rating in the entire dataset and predict every rating to be the average. The following formula represents the model:

$$y_{m,u} = \hat{\mu} + \varepsilon_{m,u}$$

Where $\hat{\mu}$ is the mean rating and $\varepsilon_{m,u}$ is the error of the rating for movie m and user u.

```
# Compute the mean rating in the edx dataset
mu_hat <- mean(edx$rating)

# Calculate RMSE using the average
results <- data.frame(model = "Only The Average",
                      RMSE = RMSE(validation$rating, mu_hat))
```

Note that the average rating is 3.5124652. By predicting every rating to be the average, we calculated an RMSE of 1.0612018. This can certainly be improved by adding more features to the model.

Models - Movie Effect

This model considers the effect of movies on the dataset. This is also called movie bias. The model is represented by the following formula:

$$y_{m,u} = \hat{\mu} + b_m + \varepsilon_{m,u}$$

Where b_m represents the movie bias. The greater the value of the movie bias, the more favorable the movie is.

```
# Compute the mean difference between the movie's rating and the average
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu_hat))

# Predict the movie ratings using the mean difference for each movie (movie effect)
y_hat_movies <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(prediction = mu_hat + b_m) %>%
  .$prediction

# Calculate RMSE using the movie effect model
results <- results %>%
  add_row(model = "Movie Effect",
  RMSE = RMSE(validation$rating, y_hat_movies))
```

By ading the movie effect to the model, we calculated an RMSE of 0.9439087, which is a substantial improvement from just using the average. However, this doesn't meet the desired RMSE. We can try regularization to get better results.

Models - Movie Effect (Regularized)

An observation that was made from exploring the dataset was that some movies tend to get more ratings than others. However, some movies don't have a lot of ratings, which can lead to large variations in the movie biases. To control this, we can use **regularization**. Regularization helps control variability of the effects by penalizing large values of b_m from smaller sample sizes. We use the following formula:

$$\frac{1}{N}\sum_{m,u}(y_{m,u} - \mu - b_m)^2 + \lambda\sum_{m} b_m^2$$

Where λ is a parameter that represents the penalty. However, we want to ensure that λ minimizes the equation above. To do this, we want to find the lambdas that do so, using the following formula:

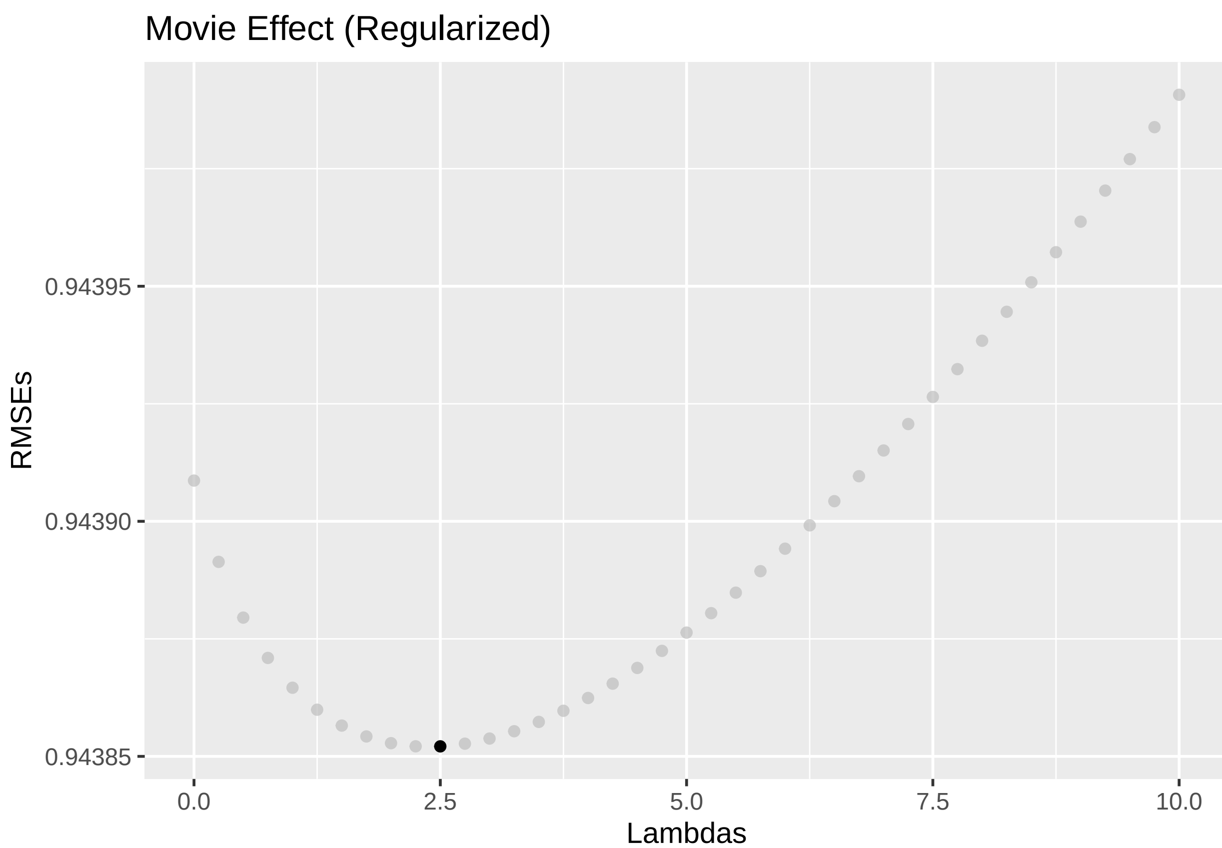
$$\hat{b}_m(\lambda) = \frac{1}{\lambda + n_m}\sum_{u=1}^{n_m}(Y_{m,u} - \hat{\mu})$$

Where n_m is the number of ratings for movie m.

```r
# Try this sequence of lambdas
lambdas <- seq(0, 10, 0.25)

# Returns the RMSEs for each lambda
rmses_1 <- sapply(lambdas, function(lambda) {
  # Print lambda to keep track of which lambda the function is using
  print(paste("Lambda:", lambda))

  movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat) / (n() + lambda))

  y_hat_movies_regularized <- validation %>%
    left_join(movie_avgs, by='movieId') %>%
    mutate(prediction = mu_hat + b_m) %>%
    .$prediction

  return(RMSE(validation$rating, y_hat_movies_regularized))
})
```

For regularized movie model, we used multiples of 0.25 up to, and including, 10. So $\lambda = 0, 0.25, 0.5, ..., 10$. The RMSEs for each λ are plotted in the following graph, with the minimum RMSE highlighted:

```r
# Plot the lambdas and their respective RMSEs
data.frame(Lambdas = lambdas, RMSEs = rmses_1) %>%
  ggplot(aes(Lambdas, RMSEs)) +
  geom_point() +
  gghighlight(RMSEs == min(RMSEs)) +
  ggtitle("Movie Effect (Regularized)")
```




```r
# Add the regularized model to results
results <- results %>%
  add_row(model = "Regularized Movie Effect",
          RMSE = min(rmses_1))
```

Therefore, the most optimal λ is 2.5 since it produces the lowest RMSE, which is =1.0612018. This is a slight improvement, however this is only using the movie effects.

Models - Movie + User Effect

In the previous model, we used only the movie effect. This model will include the user effect as well. The formula that represents the model is the following:

$$y_{m,u} = \hat{\mu} + b_m + b_u + \varepsilon_{m,u}$$

Where b_u is the user effect. As shown previously, some users tend to rate movies higher than others and vice versa, so to include this in the model should improve our results even more.

```r
# Compute the mean difference between the user's rating
# and the average with movie effect.
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_m))
```

```r
# Predict the movie ratings using movie and user effects
y_hat_movies_users <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(prediction = mu_hat + b_m + b_u) %>%
  .$prediction

# Calculate RMSE using the movie and user effects model
results <- results %>%
  add_row(model = "Movie + User Effect",
          RMSE = RMSE(validation$rating, y_hat_movies_users))
```

After generating the model and predicting the ratings on the `validation` set, the RMSE of the model is 0.8653488, which brings us much closer to the targetted RMSE.

Models - Movie + User Effect (Regularized)

Using regularization on the previous model should help improve the RMSE a bit more. The formula we used for this model is:

$$\frac{1}{N}\sum_{m,u}(y_{m,u}-\mu-b_m-b_u)^2+\lambda(\sum_m b_m^2+\sum_u b_u^2)$$

And the formula used to minimize the formula above is:

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

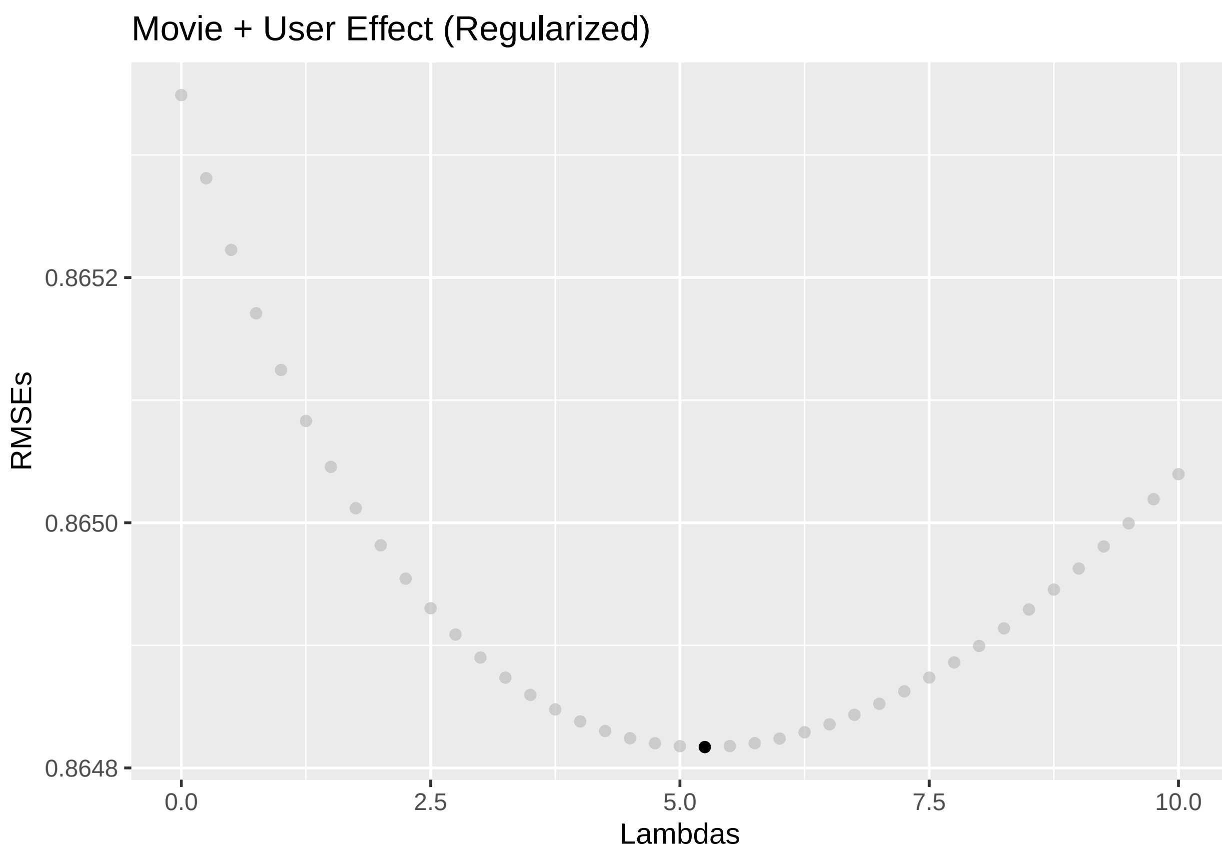
Where n_u represents the number of ratings by that user.

```r
# Try this sequence of lambdas
lambdas <- seq(0, 10, 0.25)

# Returns the RMSEs for each lambda
rmses_2 <- sapply(lambdas, function(lambda) {
  # Print lambda to keep track of which lambda the function is using
  print(paste("Lambda:", lambda))

  movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat) / (n() + lambda))

  user_avgs <- edx %>%
    left_join(movie_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_m)/(n() + lambda))

  y_hat_movies_users_regularized <- validation %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    mutate(prediction = mu_hat + b_m + b_u) %>%
```

```
    .$prediction

  return(RMSE(validation$rating, y_hat_movies_users_regularized))
})
```

We also used $\lambda = 0, 0.25, 0.5, ..., 10$ for this regularized model. The RMSEs for each λ are plotted in the following graph, with the minimum RMSE highlighted:

```
# Plot the lambdas and their respective RMSEs
data.frame(Lambdas = lambdas, RMSEs = rmses_2) %>%
  ggplot(aes(Lambdas, RMSEs)) +
  geom_point() +
  gghighlight(RMSEs == min(RMSEs)) +
  ggtitle("Movie + User Effect (Regularized)")
```




```
# Add the regularized model to results
results <- results %>%
  add_row(model = "Regularized Movie + User Effect",
          RMSE = min(rmses_2))
```

Here, the optimal λ is 5.25, which produces an RMSE of 0.864817. We have achieved our goal of producing an RMSE under 0.8649!

Models - Movie + User + Release Year Effect

The next feature that was considered was the release year, since there seems to be an effect on the ratings based on the year the movie was released. The data suggested that older movies tend to have higher ratings than newer ones. In this model, the release year will be accounted for as well using the following formula:

$$y_{m,u} = \hat{\mu} + b_m + b_u + b_y + \varepsilon_{m,u}$$

Where b_y is the release year effect.

```r
# Compute the mean differences
releaseyear_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(releaseYear) %>%
  summarize(b_y = mean(rating - mu_hat - b_m - b_u))

# Predict the movie ratings using movie, user, and year effects
y_hat_movies_users_year <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(releaseyear_avgs, by='releaseYear') %>%
  mutate(prediction = mu_hat + b_m + b_u + b_y) %>%
  .$prediction

# Calculate RMSE using the movie, user, & year effects model
results <- results %>%
  add_row(model = "Movie + User + Release Year Effect",
          RMSE = RMSE(validation$rating, y_hat_movies_users_year))
```

The results obtained from generating this model was 0.8650043.

Models - Movie + User + Release Year Effect (Regularized)

The formula used for this regularized model was:

$$\frac{1}{N}\sum_{m,u}(y_{m,u} - \mu - b_m - b_u - b_y)^2 + \lambda(\sum_m b_m^2 + \sum_u b_u^2 + \sum_y b_y^2)$$

The formula used to minimize the formula above is:

$$\hat{b}_y(\lambda) = \frac{1}{\lambda + n_y}\sum_{u=1}^{n_y}(y_{m,u} - \hat{\mu} - b_m - b_u)$$

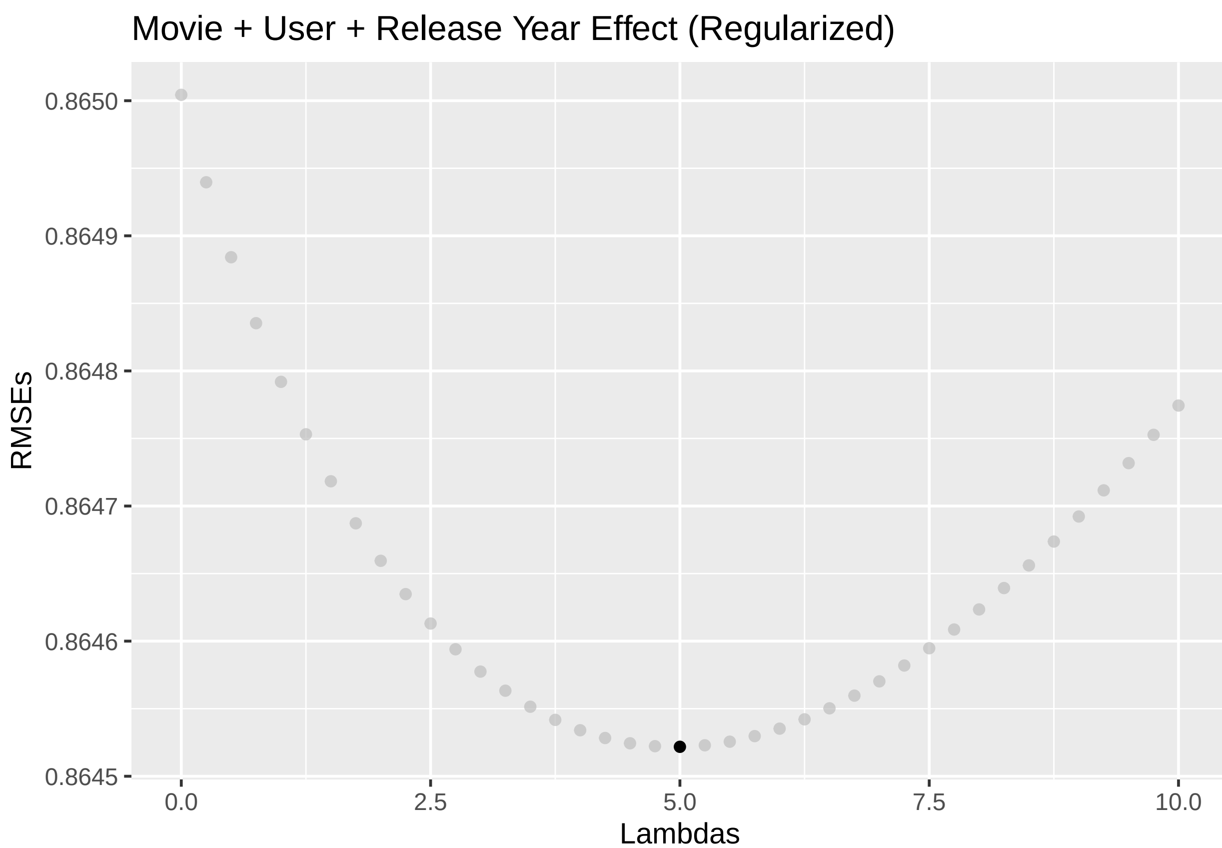
Where n_y represents the number of ratings given to all movies of a particular release year.

```r
# Try this sequence of lambdas
lambdas <- seq(0, 10, 0.25)

# Returns the RMSEs for each lambda
rmses_3 <- sapply(lambdas, function(lambda) {
```

```r
  # Print lambda to keep track of which lambda the function is using
  print(paste("Lambda:", lambda))

  movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat) / (n() + lambda))

  user_avgs <- edx %>%
    left_join(movie_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_m)/(n() + lambda))

  releaseyear_avgs <- edx %>%
    left_join(movie_avgs, by="movieId") %>%
    left_join(user_avgs, by="userId") %>%
    group_by(releaseYear) %>%
    summarize(b_y = sum(rating - mu_hat - b_m - b_u)/(n() + lambda))

  y_hat_movies_users_years_regularized <- validation %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(releaseyear_avgs, by='releaseYear') %>%
    mutate(prediction = mu_hat + b_m + b_u + b_y) %>%
    .$prediction

  return(RMSE(validation$rating, y_hat_movies_users_years_regularized))
})
```

Below is the plot of the RMSEs using $\lambda = 0, 0.25, 0.5, ..., 10$.

```r
# Plot the lambdas and their respective RMSEs
data.frame(Lambdas = lambdas, RMSEs = rmses_3) %>%
  ggplot(aes(Lambdas, RMSEs)) +
  geom_point() +
  gghighlight(RMSEs == min(RMSEs)) +
  ggtitle("Movie + User + Release Year Effect (Regularized)")
```




```
# Add the regularized model to results
results <- results %>%
  add_row(model = "Regularized Movie + User + Release Year Effect",
          RMSE = min(rmses_3))
```

The optimal λ for this model is 5, which produces an RMSE of 0.8645218. This is a small improvement, but it's still under the target RMSE.

Models - Movie + User + Release Year + Genre Effect

Due to the large amount of memory required to generate the next two models, it is advised that users who would like to generate the models themselves should ensure that they can **allocate nearly 25GB of memory for just this model**. To accurately produce a model that considers the genres of the movies, a copy of the `edx` and `validation` datasets were made, but the copies had the genres of each movie in their own rows. This results in a dataset with as many as over 25 million rows between the two datasets!

```
# Separate the genres into their own rows and rename the column to 'genre'.
# WARNING: THIS WILL CONSUME ROUGHLY 25GB OF MEMORY!!!
edx_genres_split <- edx %>%
  rename(genre = genres) %>%
  separate_rows(genre, sep = "\\|")

validation_genres_split <- validation %>%
  rename(genre = genres) %>%
  separate_rows(genre, sep = "\\|")
```

The first 12 rows of the `edx_genres_split` table are shown below:

```
# Show the first 20 rows of the edx_genres_split table.
head(edx_genres_split, 12)
```

```
##    userId movieId rating     genre     title releaseYear     dateTimeOfRating
## 1       1     122      5    Comedy Boomerang        1992 1996-08-02 07:24:06
## 2       1     122      5   Romance Boomerang        1992 1996-08-02 07:24:06
## 3       1     185      5    Action  Net, The        1995 1996-08-02 06:58:45
## 4       1     185      5     Crime  Net, The        1995 1996-08-02 06:58:45
## 5       1     185      5  Thriller  Net, The        1995 1996-08-02 06:58:45
## 6       1     292      5    Action  Outbreak        1995 1996-08-02 06:57:01
## 7       1     292      5     Drama  Outbreak        1995 1996-08-02 06:57:01
## 8       1     292      5    Sci-Fi  Outbreak        1995 1996-08-02 06:57:01
## 9       1     292      5  Thriller  Outbreak        1995 1996-08-02 06:57:01
## 10      1     316      5    Action  Stargate        1994 1996-08-02 06:56:32
## 11      1     316      5 Adventure  Stargate        1994 1996-08-02 06:56:32
## 12      1     316      5    Sci-Fi  Stargate        1994 1996-08-02 06:56:32
```

Now that the genres are in their own rows, we can calculate the genre effects and find the predicted mean for the movies for each genre. To combine the genres' ratings to one rating, we will just take the average of the ratings given to each genre for each particular movie. The formula for this model is shown below:

$$y_{m,u} = \hat{\mu} + b_m + b_u + b_y + \frac{1}{n_{m,g}}(\sum_{g}^{m_g} b_g) + \varepsilon_{m,u}$$

Where $n_{m,g}$ is the number of genres in a particular movie, m_g represents the genres of movie, and $b_{m,g}$ represents the genre effect of that movie.

```
# This computes the mean differences.
genre_avgs <- edx_genres_split %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(releaseyear_avgs, by='releaseYear') %>%
  group_by(genre) %>%
  summarize(b_g = mean(rating - mu_hat - b_m - b_u - b_y))

# Predict the movie ratings using movie, user, and year effects
y_hat_movies_users_year_genre <- validation_genres_split %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(releaseyear_avgs, by='releaseYear') %>%
  left_join(genre_avgs, by='genre') %>%
  group_by(userId, movieId) %>%
  summarize(prediction = mu_hat[1] + b_m[1] + b_u[1] + b_y[1] + mean(b_g)) %>%
  arrange(userId, movieId) %>%
  .$prediction

# Calculate RMSE using the movie, user, year, & genre effects model
results <- results %>%
  add_row(model = "Movie + User + Release Year + Genre Effect",
          RMSE = RMSE(validation$rating, y_hat_movies_users_year_genre))
```

Since the model had to predict the rating based on each genre that a movie has, this took a while to complete. However, the results show an RMSE of 0.8648998.

Models - Movie + User + Release Year + Genre Effect (Regularized)

Here, we use the formula:

$$\frac{1}{N}\sum_{m,u}(y_{m,u} - \mu - b_m - b_u - b_y - \frac{1}{n_{m,g}}(\sum_g b_g))^2 + \lambda(\sum_m b_m^2 + \sum_u b_u^2 + \sum_y b_y^2 + \sum_g [\frac{1}{n_{m,g}}\sum_g^{m_g} b_g]^2)$$

The formula used to minimize the formula above is:

$$\hat{b}_g(\lambda) = \frac{1}{\lambda + n_g}\sum_{u=1}^{n_g}(y_{m,u} - \hat{\mu} - b_m - b_u - b_y)$$

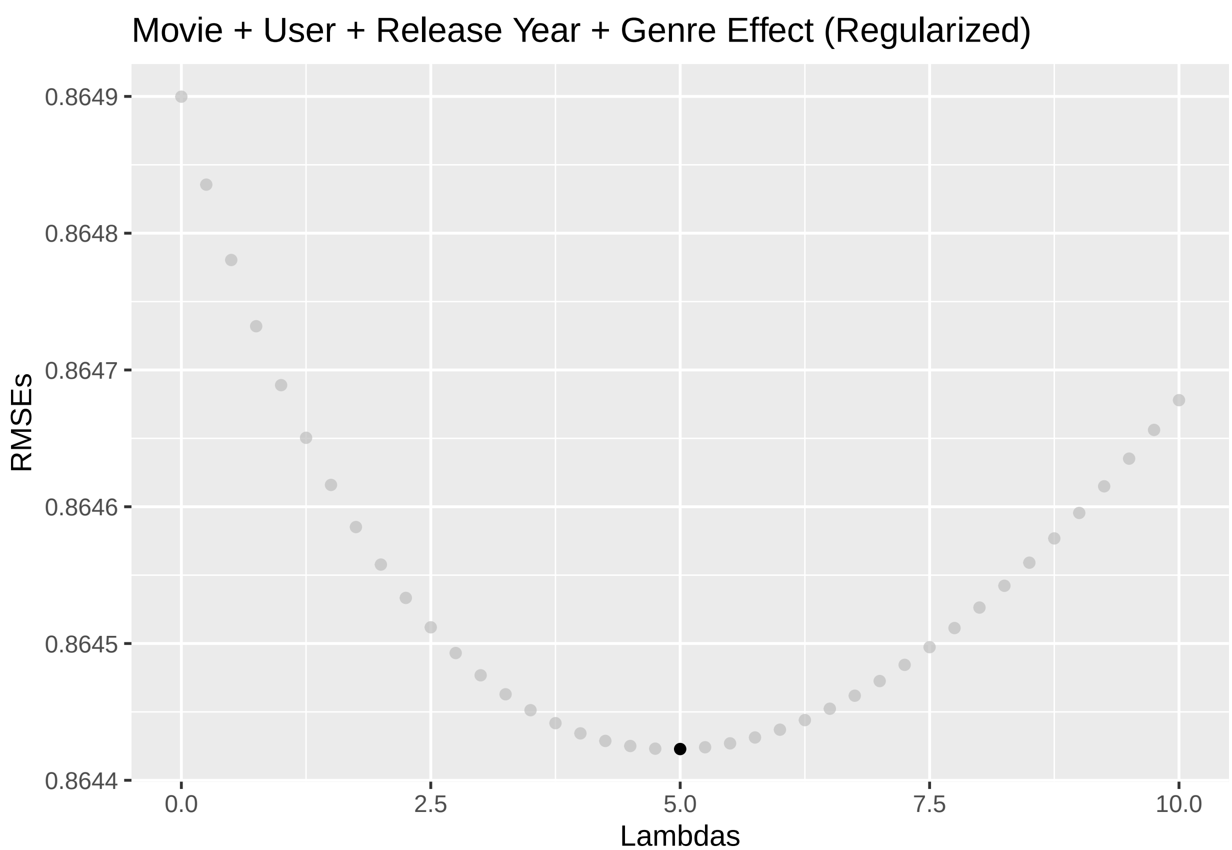
Where n_g is the number of ratings for that genre.

```
# Try this sequence of lambdas
lambdas <- seq(0, 10, 0.25)

# Returns the RMSEs for each lambda
rmses_4 <- sapply(lambdas, function(lambda) {
  # Print lambda to keep track of which lambda the function is using
  print(paste("Lambda:", lambda))

  movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu_hat) / (n() + lambda))

  user_avgs <- edx %>%
    left_join(movie_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_m)/(n() + lambda))

  releaseyear_avgs <- edx %>%
    left_join(movie_avgs, by="movieId") %>%
    left_join(user_avgs, by="userId") %>%
    group_by(releaseYear) %>%
    summarize(b_y = sum(rating - mu_hat - b_m - b_u)/(n() + lambda))

  genre_avgs <- edx_genres_split %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(releaseyear_avgs, by='releaseYear') %>%
    group_by(genre) %>%
    summarize(b_g = sum(rating - mu_hat - b_m - b_u - b_y)/(n() + lambda))

  y_hat_movies_users_years_genres_regularized <- validation_genres_split %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(releaseyear_avgs, by='releaseYear') %>%
    left_join(genre_avgs, by='genre') %>%
```

```r
    group_by(userId, movieId) %>%
    summarize(prediction = mu_hat[1] + b_m[1] + b_u[1] + b_y[1] + mean(b_g)) %>%
    arrange(userId, movieId) %>%
    .$prediction

  return(RMSE(validation$rating, y_hat_movies_users_years_genres_regularized))
})
```

The following plot shows the RMSEs using $\lambda = 0, 0.25, 0.5, ..., 10$:

```r
# Plot the lambdas and their respective RMSEs
data.frame(Lambdas = lambdas, RMSEs = rmses_4) %>%
  ggplot(aes(Lambdas, RMSEs)) +
  geom_point() +
  gghighlight(RMSEs == min(RMSEs)) +
  ggtitle("Movie + User + Release Year + Genre Effect (Regularized)")
```




```r
# Add the regularized model to results
results <- results %>%
  add_row(model = "Regularized Movie + User + Release Year + Genre Effect",
          RMSE = min(rmses_4))
```

We see that the optimal λ for this model is 5 and the resultant RMSE is **0.8644229**.

Results

The results of all the models used in the report, along with the RMSEs, are shown below:

```
##                                                   model      RMSE
## 1                                      Only The Average 1.0612018
## 2                                          Movie Effect 0.9439087
## 3                              Regularized Movie Effect 0.9438521
## 4                                   Movie + User Effect 0.8653488
## 5                       Regularized Movie + User Effect 0.8648170
## 6                      Movie + User + Release Year Effect 0.8650043
## 7          Regularized Movie + User + Release Year Effect 0.8645218
## 8              Movie + User + Release Year + Genre Effect 0.8648998
## 9 Regularized Movie + User + Release Year + Genre Effect 0.8644229
```

Only 4 models met the objective, which was to generate an RMSE of under 0.8649. The regularized movie and user effect model is the first to do so. Conveniently, it is much faster to generate than the other 3 models. However, it has the highest RMSE out of the 4 models that met the desired RMSE. The final model produced the lowest RMSE, but at the expense of performing much slower and demanding a lot of RAM.

It is also noted that movie and user effect had the biggest effect on the RMSE. Just by adding the movie effect, we were able to reduce the RMSE by approximately 0.12. Adding in the user effect reduced the RMSE by almost 0.08.

Regularization helped us improve our results, but only slightly. However, it managed to bring the movie and user effect as well as the movie, user, and release year effect models within the desired RMSE.

Conclusion

The analysis of the data showed some surprising observations. For instance, while `Drama` and `Comedy` are common genres, the movies that contain these genres don't necessarily rate as high as others on average. Also, users tend to rate older movies higher than newer movies, but newer movies in the dataset tend to have more ratings.

When it comes to particular movies, `The Shawshank Redemption` was consistently ranked as one of the best movies based on number of ratings, average ratings, and median ratings.

By using a regularized movie & user effect model, we can achieve an RMSE under 0.8649. It seems that additonal features added to this only improve the RMSE slightly, but at the cost of a slower model that demands more resources to complete.