PH125.9x Data Science Capstone Final Movie Project Report

Marina Ganopolsky

9/4/2020

Contents

Overview

This is Marina Ganopolsky's implementation of the **MovieLens Project** of the **Harvard: PH125.9x Data Science, Summer 2020: Capstone**.

The objectives of this project are:

1. To create a movie rating engine in R based on the data set provided by training a number of machine learning algorithms on the data.
2. To choose the best-performing algorithm, and make a final evaluation of said algorithm on the validation dataset using the **RMSE** (Root Mean Square Error) of the predicted ratings.
3. To achieve an RMSE value of **< 0.8649**, and validate this with the **validation data set**, as described below.
4. To summarize the findings, draw conclusions, and provide further recommendations, if needed.

The dataset is provided by the staff, and is wrangled for better use during the project beyond the manipulations suggested in the assignment; the modifications will be shown in the **Dataset** section below.

A quick summary of the data wrangling modifications will be provided, as well as some visual representations of patterns present in the data, in the **Method and Analysis** section. This will inform the reader about the **movie** trends, **user** trends and a variety of **rating** trends present in the dataset.

An exploratory analysis will be performed to generate predicted movie ratings, with various models, to be specified, until a final RMSE calculation fits our requirements. Results of the RMSE calculations will be analyzed and explained, and a conclusion will be provided.

The function used to evaluate algorithm performance is the **Root Mean Square Error**, or RMSE. RMSE is one of the most frequently used measures of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, and a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Because of this, the RMSE algorithm is sensitive to outliers.

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

Seven (7) models will be developed, ranging from the most naive to the most complex. These will be tested and compared using the resulting RMSE in order to assess their quality. The ideal evaluation criteria for this project, as specified by the assignment, is a RMSE that is expected to be lower than **0.8649**.

After evaluating every available algorithm I have come up with using the RMSE, the best-resulting model will be used on the **validation dataset** to evaluate the quality of the predictions of the movie ratings.

This project can be found on **GitHub** under **https://github.com/mganopolsky/r_final_harvard_data_science_project**.

Dataset

The MovieLens dataset is automatically downloaded with the code provided; The data sets can be additionally downloaded here, both in zip format and folders:

- https://grouplens.org/datasets/movielens/10m/
- http://files.grouplens.org/datasets/movielens/ml-10m.zip

As specified by the project description (and the provided code) the data is split into a validation set and the "rest". Furthermore, after it was manipulated, the data is again split into **testing** and **training** sets, so as to evaluate the quality of the predictions. The calculations and algorithm verification are done on the testing and training sets, and after the final model is chosen, this algorithm is applied to the validation test to verify the hypothesis.

A fair amount of data manipulation has been done to the datasets created with the provided code. The changes include:

- **New Field** : Adding a ratings average per movie in **avg_rating**
- **New Field** : Extracting the release year of the movie from its title. (Every movie title seems to include the release year in at the end of title, surrounded by parentheses.) and creating the field **release_year** with the information
- **New Field** : Creating a field called **rating_date** , by extracting the rating date from the **timestamp** field
- **New Field** : Creating a field called **age_of_movie**, by subtracting the year the film was released from the year parameter of the current date
- **New Field** : Creating a field called **rating_year** by extracting the rating year from the **rating_date** field.
- Removing the year from the movie title
- Removing films with **3 reviews or less**.
- Removing the parentheses from the **release_year** and converting it to a numerical field
- After the data wrangling is complete, the dataset is split up into :
  - **A validation set**, comprised of **10%** of the data complete data; this portion of the data shall not to be analyzed until the final algorithm is found to have an RMSE to be within acceptable range

- **A training dataset** - the data the algorithms will be trained on, 90% of the remaining data.
- **A testing dataset** - the data the intermediate algorithms will be tested on, 10% of the remaining data.

I will re-iterate that the algorithm development and tuning will be conducted on the **training** and **testing** data set, and the validation set will be used to evaluate the final chosen algorithm. **During no other time will the validation dataset be used.**

As a first step, in order to get some basic information about the data we're working with, we examine the `movielens` dataset. The original dataset contains six variables :

- **userId**
- **movieId**
- **rating**
- **timestamp**
- **title**
- **genres**

After massaging the data a bit, the following fields were added, as derived from the existing fields :

- **avg_rating** - the mean rating of the film (same value across the same `movieId`)
- **release_year** - the numeric release year of the film
- **rating_date** - the date the rating was submitted
- **age_of_movie** - the age of the film in years (same value for the same `movieId`)
- **rating_year** - the numeric date the rating was submitted
- **rating_count** - the cumulative number of ratings submitted for the film (same value across the same `movieId`)

Per row, this is a representation of one user's rating of a single movie. As you can tell, the **genres** per film are currently combined into **"|"**-delimited string, which I will later separate.

As a first glance, here's a snapshot of dataset:

```
     userId          movieId          rating        timestamp
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
 1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
 Median :35739   Median : 1834   Median :4.000   Median :1.035e+09
 Mean   :35869   Mean   : 4118   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53607   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
   avg_rating      rating_count      title              genres
 Min.   :0.8033   Min.   :    4   Length:9999356     Length:9999356
 1st Qu.:3.2190   1st Qu.: 1814   Class :character   Class :character
 Median :3.5913   Median : 4699   Mode  :character   Mode  :character
 Mean   :3.5125   Mean   : 7542
 3rd Qu.:3.8754   3rd Qu.:10928
 Max.   :4.7500   Max.   :34864
  release_year   rating_date                    age_of_movie     rating_year
 Min.   :1915   Min.   :1995-01-09 00:00:00   Min.   : 12.00   Min.   :1995
 1st Qu.:1987   1st Qu.:2000-01-02 00:00:00   1st Qu.: 22.00   1st Qu.:2000
 Median :1994   Median :2002-10-24 00:00:00   Median : 26.00   Median :2002
 Mean   :1990   Mean   :2002-09-21 09:03:56   Mean   : 29.78   Mean   :2002
 3rd Qu.:1998   3rd Qu.:2005-09-14 00:00:00   3rd Qu.: 33.00   3rd Qu.:2005
 Max.   :2008   Max.   :2009-01-05 00:00:00   Max.   :105.00   Max.   :2009
```

The first few lines of the dataset look like this:

```
  userId movieId rating timestamp avg_rating rating_count
1      1     122      5 838985046   2.861318         2412
2      1     185      5 838983525   3.125209        14975
3      1     231      5 838983392   2.936950        17851
4      1     292      5 838983421   3.418414        16075
5      1     316      5 838983392   3.349353        18925
6      1     329      5 838983392   3.336271        16167
                  title                         genres release_year rating_date
1             Boomerang                 Comedy|Romance         1992  1996-08-02
2              Net, The          Action|Crime|Thriller         1995  1996-08-02
3          Dumb & Dumber                         Comedy         1994  1996-08-02
4              Outbreak  Action|Drama|Sci-Fi|Thriller         1995  1996-08-02
5              Stargate       Action|Adventure|Sci-Fi         1994  1996-08-02
6 Star Trek: Generations Action|Adventure|Drama|Sci-Fi         1994  1996-08-02
  age_of_movie rating_year
1           28        1996
2           25        1996
3           26        1996
4           25        1996
5           26        1996
6           26        1996
```

A summary of the subset confirms that there are no missing values.

The total of unique movies and users in the movielens subset is **69,878** unique users and **10,331** different movies:

```
  n_users n_movies
1   69878    10331
```

If every user rated every movie, we would have **721,909,618** rows; however, we only have this many rows:

```
[1] 9,999,356
```

Therefor, the database is fairly **sparsely populated**. I will visualize this later in the document.

Methods and Analysis

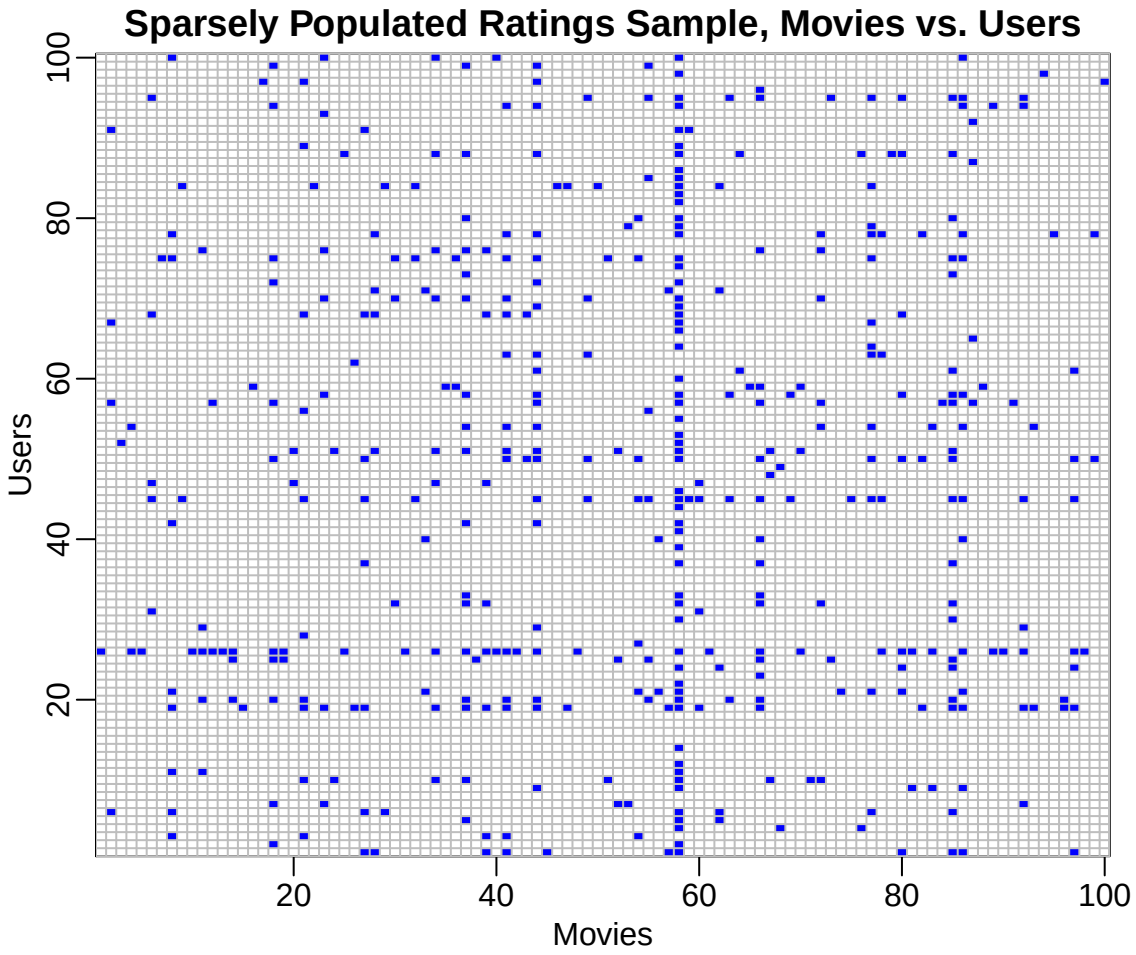
Data Insights, Cleaning & Visualizations

Sparsity

```
[1] 1.39 %
```

As noted, if every user rated every movie, we would have approximately **721,909,618** entries in the dataset; however, as shown above, we only have **9,999,356** ratings. Apparently, the data set is actually less then **1.39%** populated, which we can see from the calculation above.

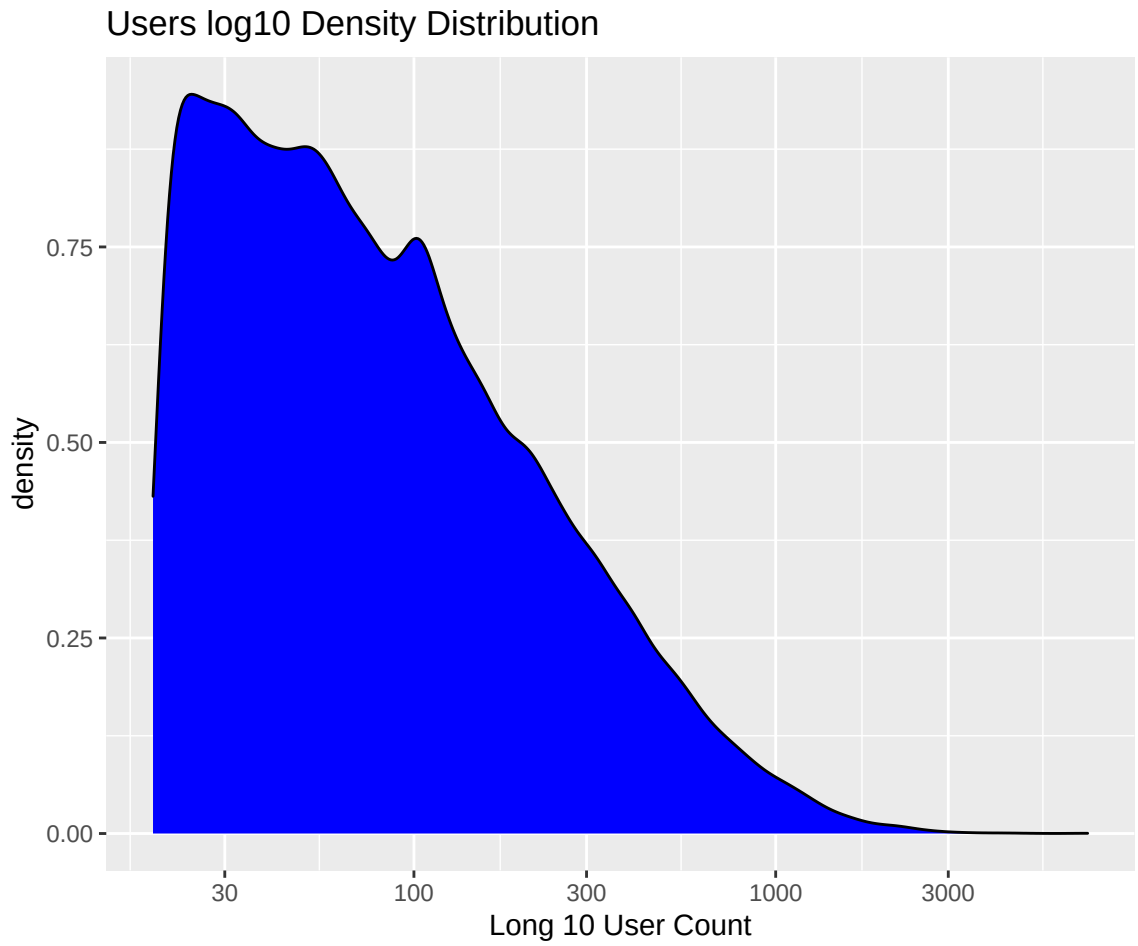
Visually, we can show the sparseness on a small sample of the data, with the following image graph:



Users

Summarizing the users in the system, we have users with **a minimum of 20 reviews**; however, the maximum is **> 7000 reviews** per user.

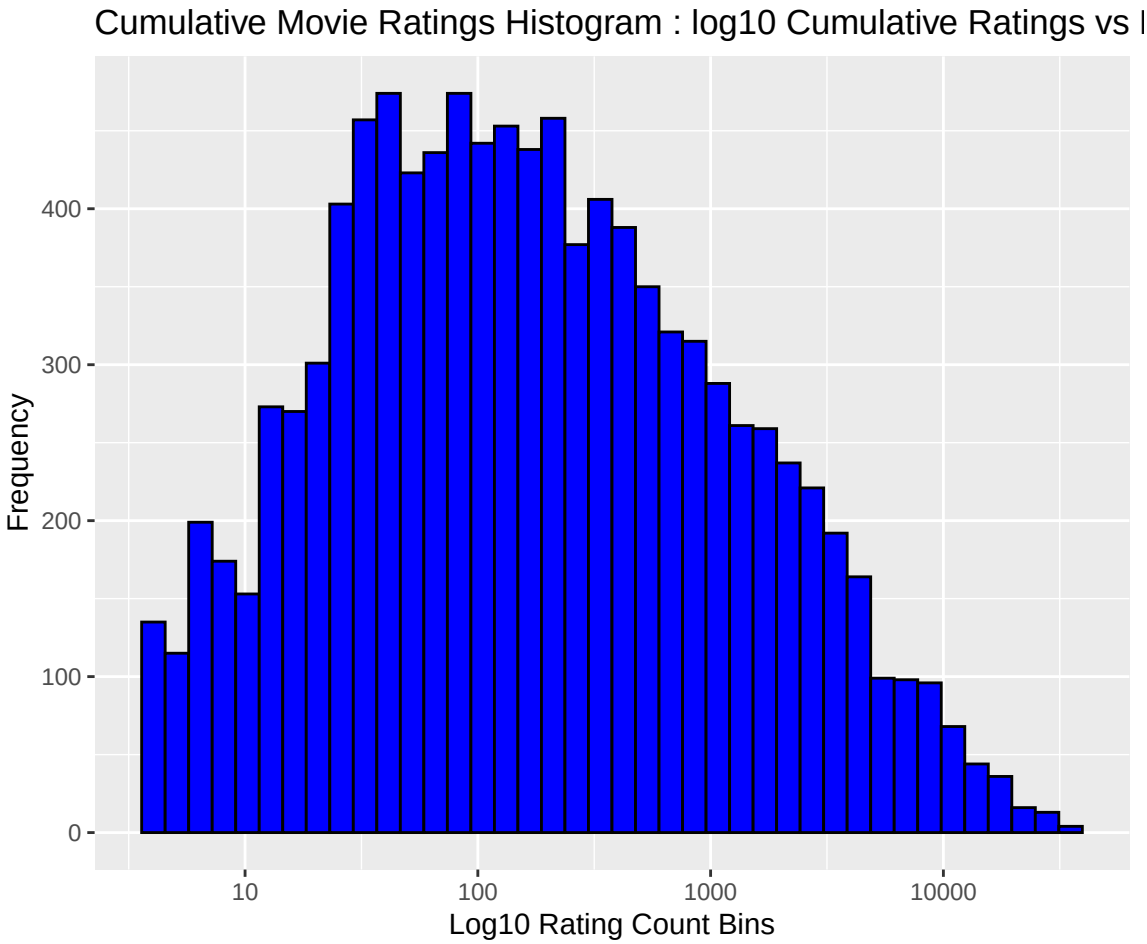
Viewing the ratings' density distribution, **it is obvious that most of the ratings come from a small percentage of users**:



If we examine the trends in the amount of ratings a user submits, versus the average rating, per user, we can clearly see that users with smaller amounts of ratings are much more likely to give extreme ratings on either side of the scale - be either very cranky (and give a rating of 0.5) or extra optimistic (and give a rating of 5). In the plots below I show the difference in the variability of user's average rating trends vs. the amount of ratings they have submitted.

Movies

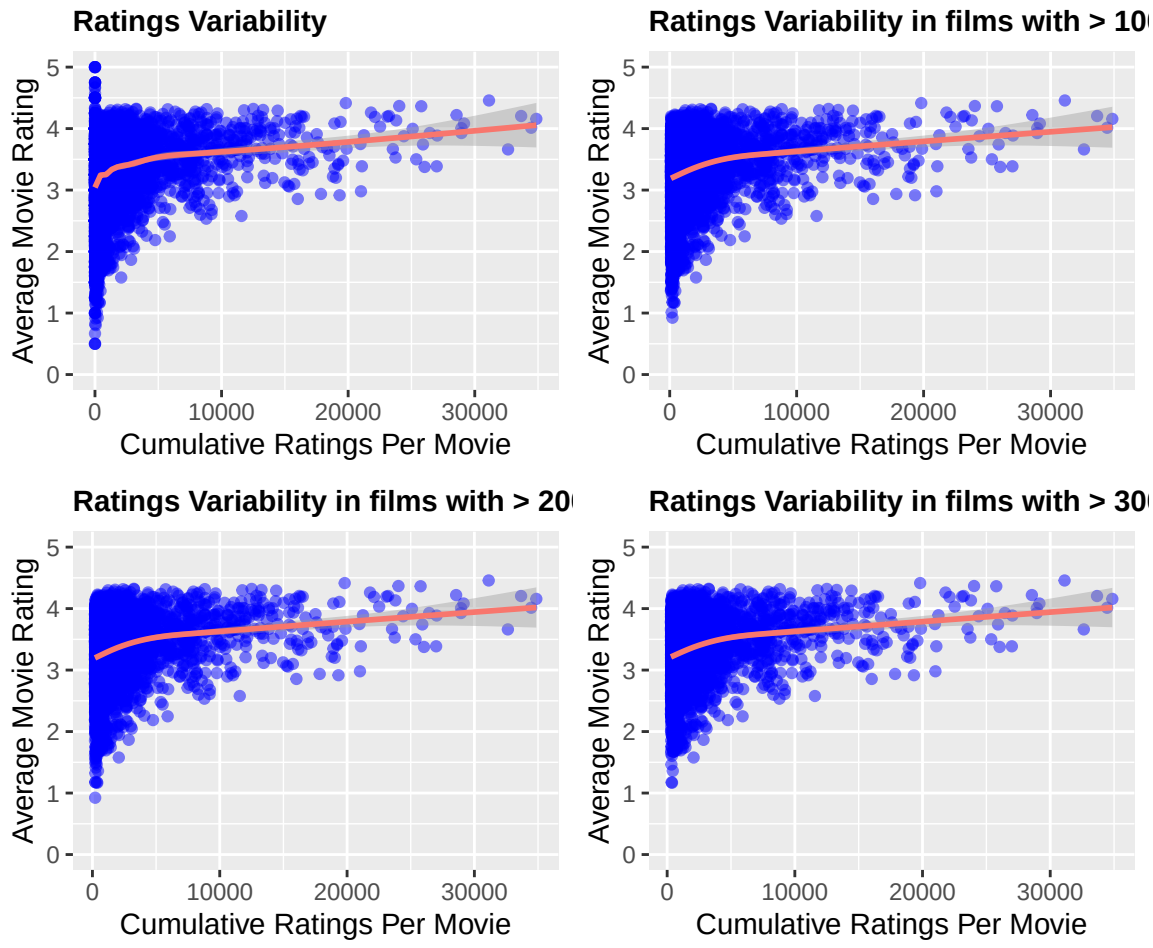
We can also show that **each movie is not rated with the same frequency**; That is, the popular movies will be seen by many more people and will therefor be rated by many more people. Meanwhile, there are also many independent films that will only be seen by a few people, and therefor will only be rated by a small percentage of those.



The **users** data came with a specifications that **only users with 20 or more reviews are included**; however, no similar filtering was done for movies. While working on this project, I have found that if the dataset includes movies with 3 reviews or less, the **`createDataPartition {caret}`** function splits up the data in a way that will keep some of the movies in the testing set and some in the training set. The documentation states :

```
Also, for createDataPartition, very small class sizes (<= 3) the classes may
not show up in both the training and test data
```

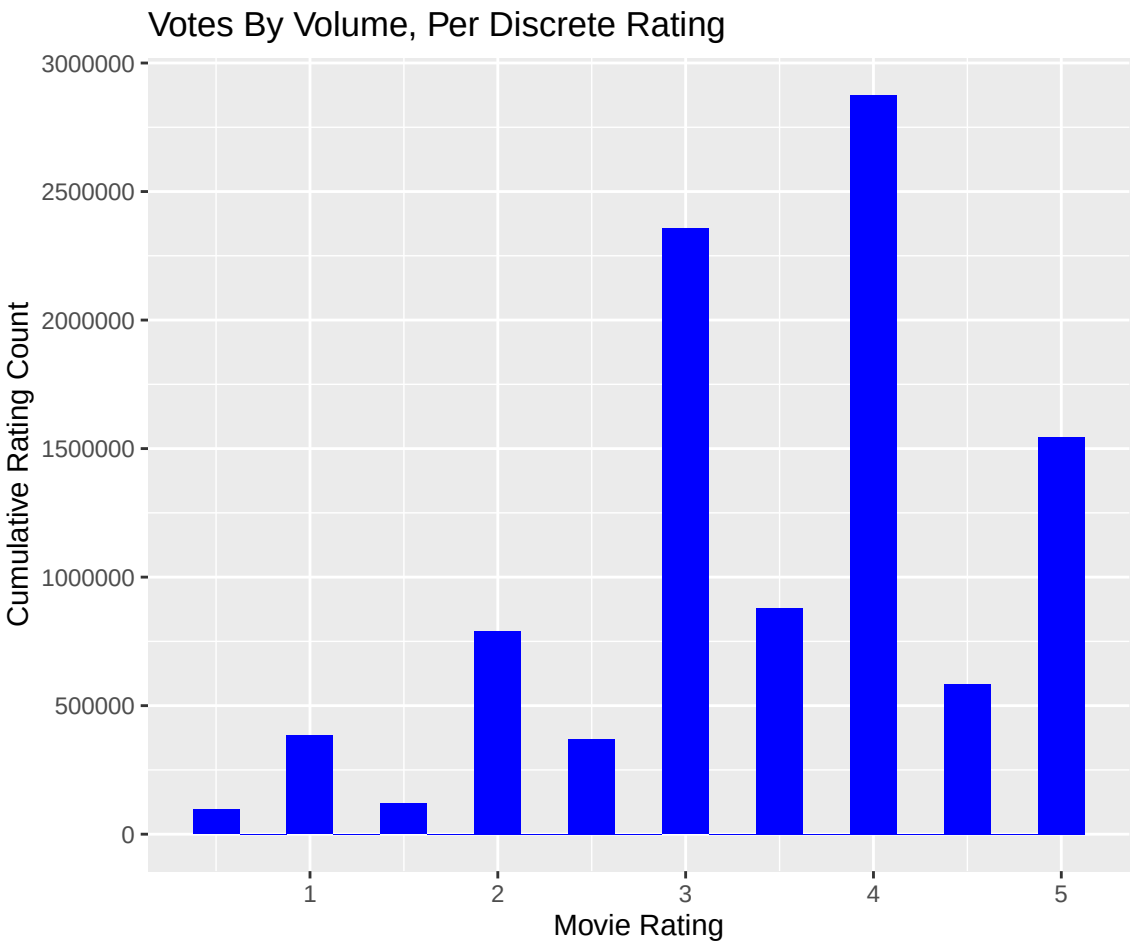
This particular quirk sets up the dataset so that the `inner_join` function used in the regularization sections returns NA values for these movies, making the RMSE calculations invalid. Therefor, I have removed the entries for movies that have 3 reviews or less, excluding some 30 ratings. If we examine the data, we can see that the variability of average film ratings is very high when the number of ratings is low, and it decreases steadily as the rating counts get into the hundreds and then thousands. This trend makes sense if we consider that an obscure film will only be seen by a small audience that may be biased for or against it, making for an an unusual amount of outlier ratings. As the film is seen by more people, it will get rated by more people, and the variability of those ratings will change as well. The plots below shows the variability of ratings when movies with less then 100 ratings are removed; We repeat the same with 200, and 300 cumulative ratings. The plots below will show what I mean:



The data shows that by removing the films with fewer reviews, we remove most of the extreme outliers of the **0.5** AND **5** average movie ratings. In general, the data also shows that movies with fewer ratings have a much higher variability in average movie ratings.

NOTE : It is of interest to note that out of curiousity I also ran the algorithms described below on the datasets adjust to match the graphs below - removing films with less then (100, 200, and 300) ratings. As expected, the RMSE's were indeed lower the more unfrequently rated films were removed from the database, and somewhat significantly lower then the RMSE's produced on the full data. However, as we are not allowed to modify the dataset, this was purely a theoretical exercise.

Ratings

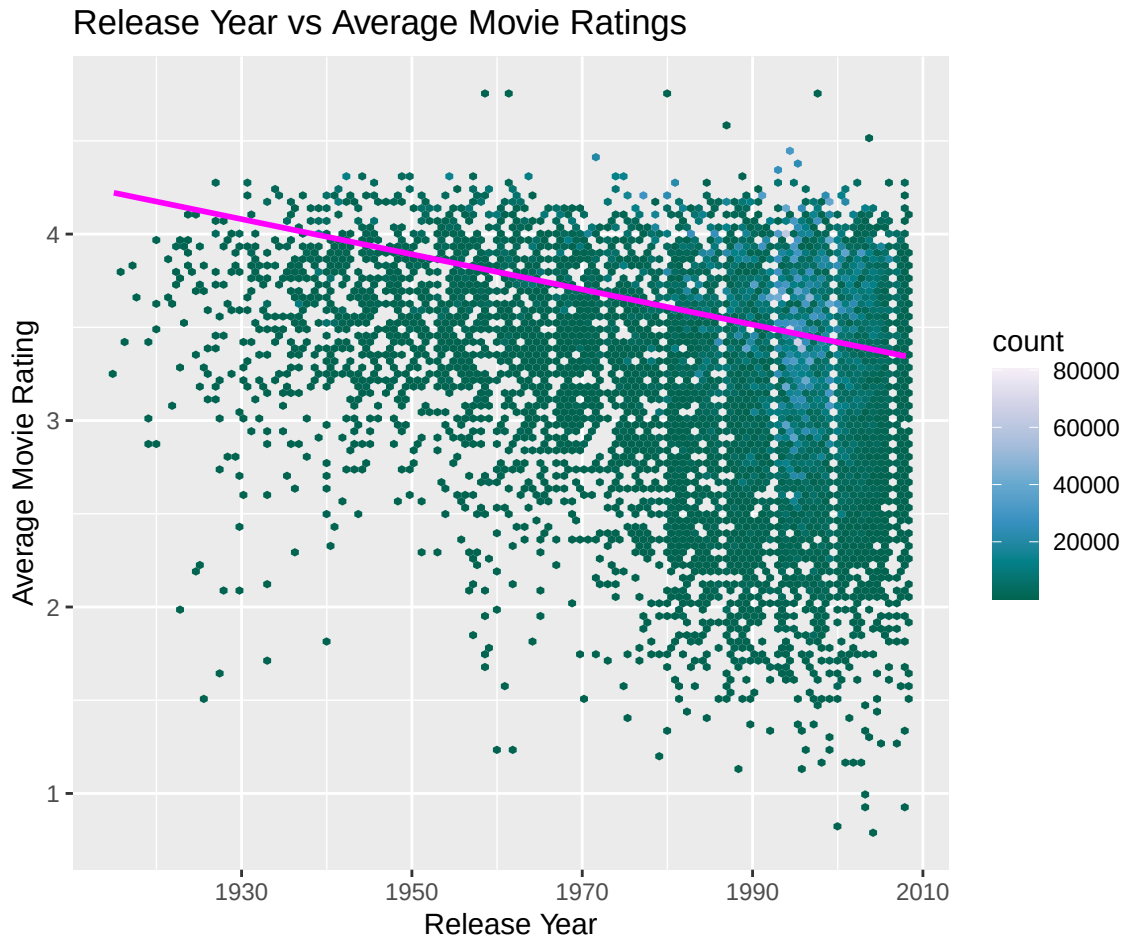


From the data and the bar graph of the ratings, we can make one these conclusions:

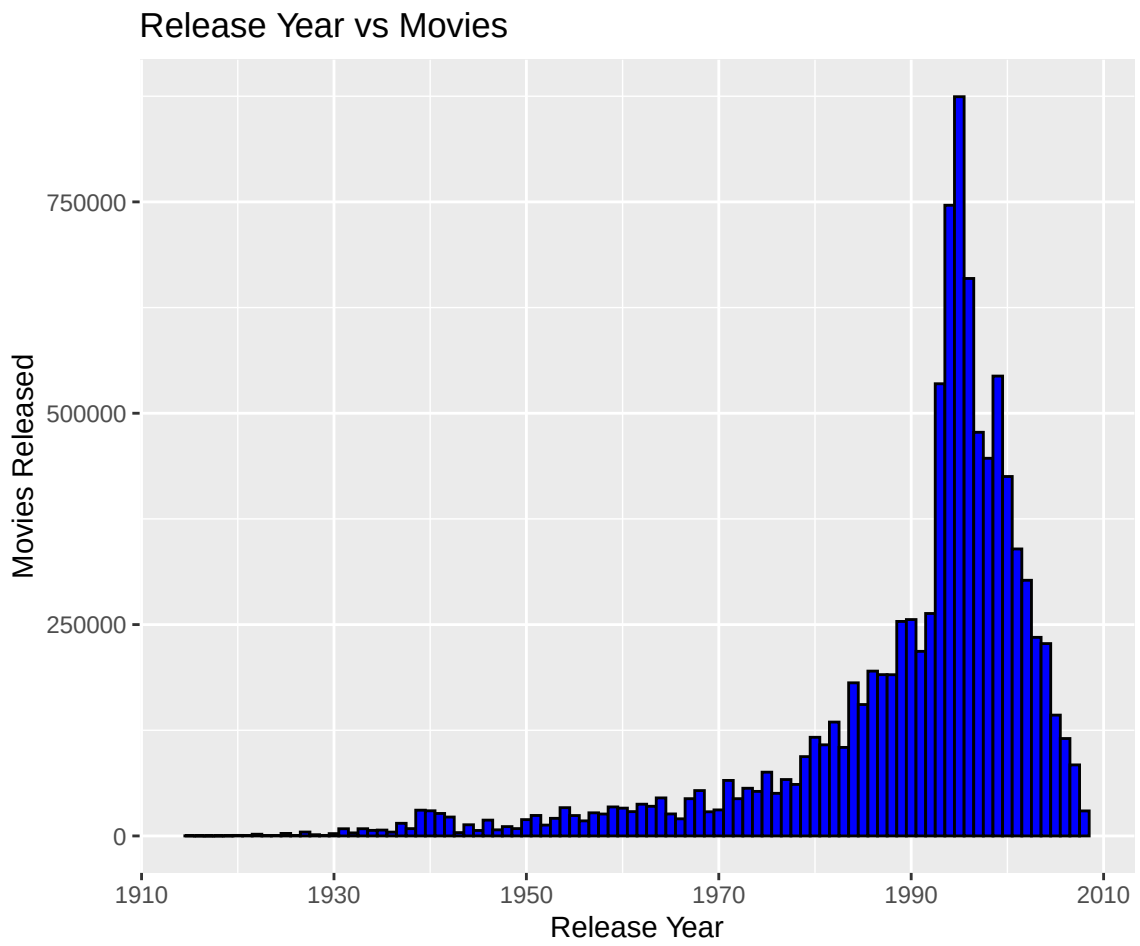
1. In terms of ratings, the values are on a **scale of 0.5-5**, in increments of 0.5.
2. There are 10 discrete options, and the ratings are **not** continuous.
3. Full-grade ratings are much more common then the half-grade ratings.
4. **4** is the most common rating.

Plotting release year vs. avg movie ratings - it seems that either:

1. Movies are getting worse with time, or
2. Reviewers are getting pickier with time.
3. More reviews are being submitted with time.



It is indeed obvious that more reviews are being submitted with time. **But is this because more movies are being made with time?**
Based on the histogram below, the answer to this is a most definite **yes** - the numbers increase steadily and peak in the mid-1990's, and start to fall off again. Does this mean that less movies are being made after 2008? That is doubtful. The dataset stopped adding movies in 2008, and were probably excluding many movies.



In fact, if we examine the top 10 release years by volume, the movies from the 1990's and 2000 are clear winners in terms of volume in this dataset.

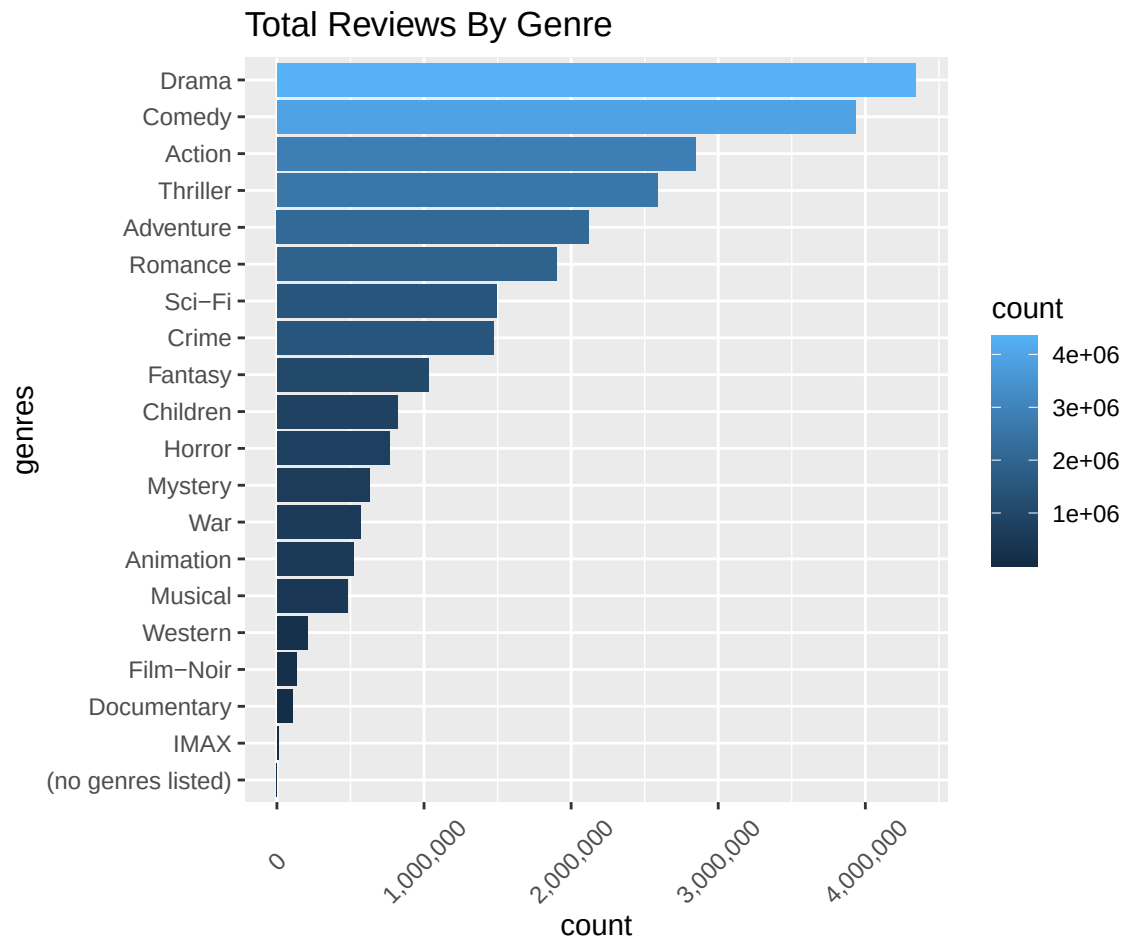
	release_year	count
1	1995	874428
2	1994	746037
3	1996	659418
4	1999	543974
5	1993	534892
6	1997	477461
7	1998	446725
8	2000	425188
9	2001	339477
10	2002	302401

Genres

If the current `genres` field is separated by the "|"-character, we can examine the individual genres a movie might fall into:

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

Below one can examine how genres generally tend to be reviewed, by volume:

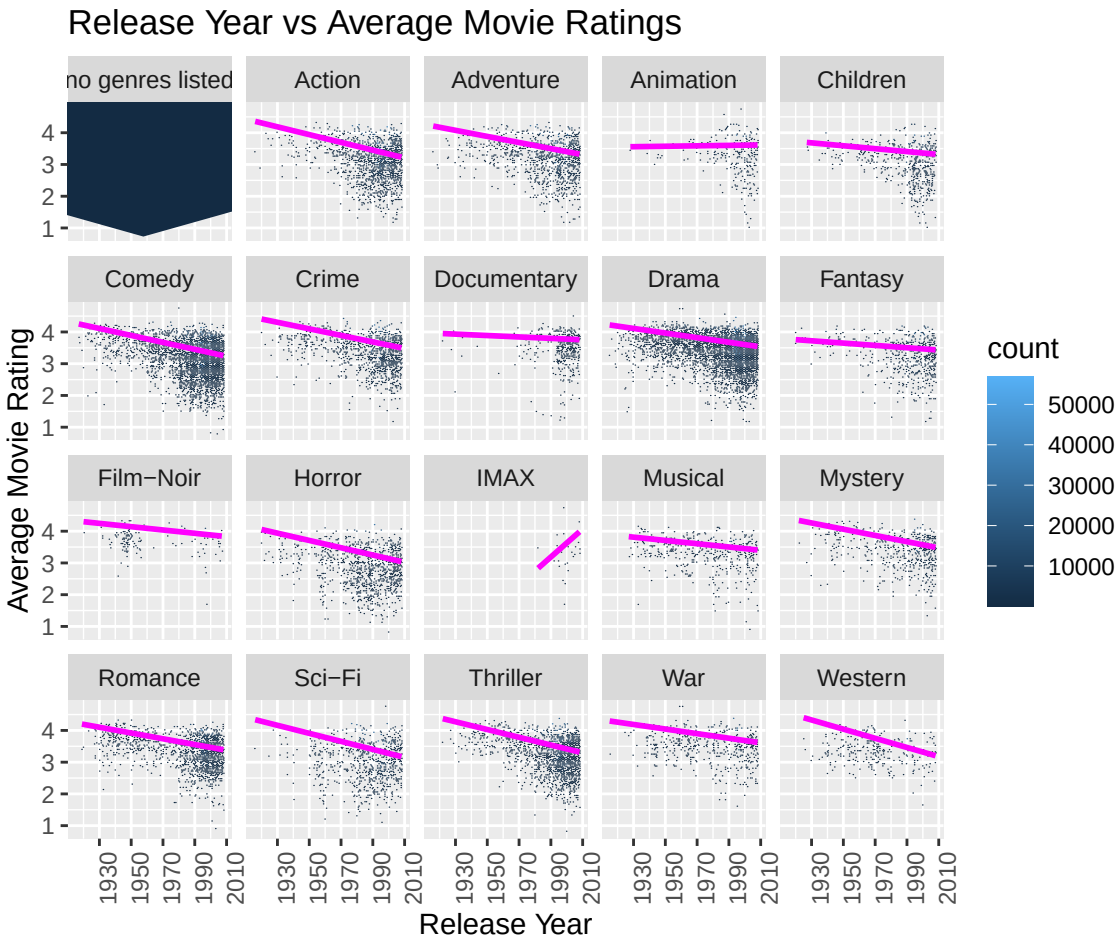


Do people tend to review certain genres more then others?

The answer seems to be a resounding **YES**!

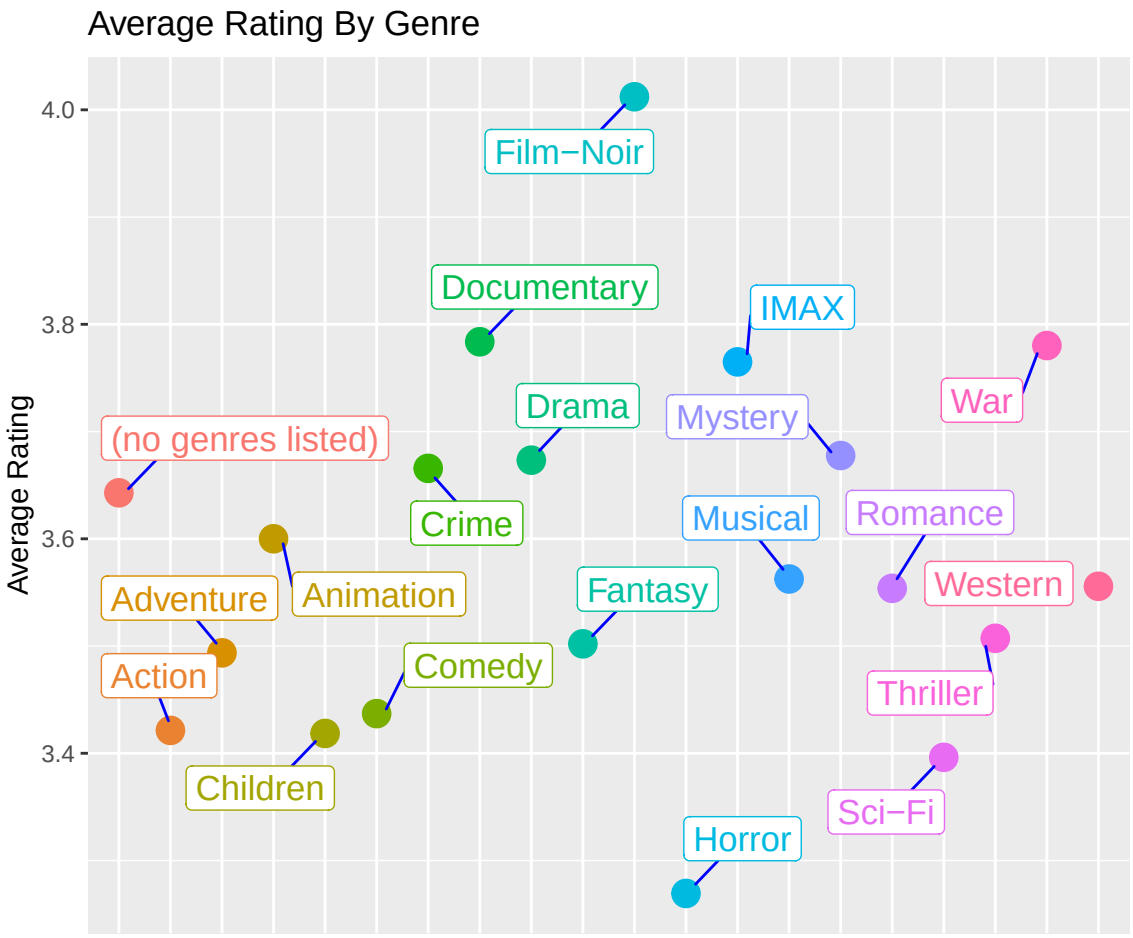
Dramas seems to have the most reviews, while **IMAX** has by far the least; this pattern makes sense, since only a small percentage of movies get released in IMAX (although the ones that are are super popular, and will thus get more reviews.)

Next, we'll explore how ratings have fared over time by **genre**. Based on the visualizations provided here, it is fairly obvious that **most of the average ratings across genres have declined over the years**. The notable exception to this is **IMAX** movies, and in a small way, **Animation**. These trends make sense, as both IMAX and animated films have benefited greatly from technological advancements over the years.



Ratings By Genre

In general, the average rating by genre can be shown on this plot; **Film Noir** seems to have the highest average rating, while **Horror** films have the lowest.



Genre Combinations

So far I have analyzed the genre information as if it was separated by individual genres. However, each film is actually listed as a combination of several genres, and shown in the `genres` field. Below, I show the minimum and maximum count of genres any movie exibits.

max_genres	min_genres
8	1

As every film can fall into multiple genres, there are this many **unique genre combinations per film** available in the dataset:

```
[1] 797
```

Rating Models

1. Average movie rating model

Earlier in the document I described the logic behind the RMSE function. Here I am enclosing it's implementation in R.

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

To start out, we will examine the **mean rating**, and use that as the benchmark for evaluations, creating a model that assumes the same rating for all movies and users with all the differences explained by random variation, $\varepsilon_{u,i}$. The formula used for this is:

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

$Y_{u,i}$ is the predicted rating per user, per movie and μ is the estimate that minimizes the root mean squared error — is the average rating of all movies across all users. The code below calculates μ on the training set of the data, and then uses the RMSE to predict the error of ratings of the test set.

This value will be used with the RMSE function to test the accuracy with most naive version of the algorithm. The output is below:

method	RMSE
Just the average	1.059136

As I continue with the project, I will display results of the RMSE evaluation of every model, and print out the table as I have above - so that the data can be reviewed as the project progresses. Taking the average of the existing ratings is a very naive predictor. We can see that the RMSE is > 1.0, which means this isn't a great prediction - as expected. In the rest of the project I will examine several better approaches.

2. Movie effect model

From the data analysis performed earlier, we've surmised that some movies will be rated higher then others - making the naive RMSE above clearly inferior to a more detailed approach. I will now build a model with accounting for bias per movie, b_i (since every movie will have some inherent bias, and some movies are clearly better (or worse) then others).

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

b_i is the average of $Y_{u,i}$, minus the overall mean for each movie i. To calculate the movie bias, we transform the formula to :

$$b_i = Y_{u,i} - \mu - \varepsilon_{u,i}$$

And run the RMSE algorithm on the test set again. The predictions improve significantly when we use this model, as shown below.

method	RMSE
Just Movie Effect Model b_i average	0.9426456

Adding a single **movie effect** to the model not only significantly reduces the RMSE to **0.9426456**, but brings it below 1.0, which is a great first step. However, the number is still not low enough, so we continue on.

3. Movie and user effect model

We compute the user bias b_u per user. Some users are cranky, and others are optimistic - so the Effect of the user's crankiness can change the ratings in either direction, positive or negative. The formula for the predicted rating would then be, building on what we have already seen in models 1 and 2 above:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$b_u = Y_{u,i} - \mu - b_i - \epsilon_{u,i}$$

method	RMSE
User Affect + Movie Effect Model	0.8649698

4. Regularized Models

So far we have been computing estimates with certain levels of uncertainty, but we have not been taking into accounts that problems may exist in the data where there is not enough information. There is always a risk of over-fitting the data, and to reduce this risk, we introduce the concept of **regularization**. Regularization will allow us to use a tuning parameter, λ, that when included in the calculation will minimize the mean squared error. This will reduce the b_i and b_u in case of small number of ratings.

Since I am attempting to narrow down a λ value quite frequently during this project over various models, I have created a function that will be used to create, calculate, plot, and narrow down

the lambda range for each model I attempt to train.

The function takes in parameters for the start and end of the range of the λ, an increment to step through the λ values, as well as another function that is passed in to do the calculations that vary between the models. The function then recursively calls itself only **once** to get an even finer-grained value of λ, with a smaller, narrower set of lambdas once a rough range has been established in the first run-through. The function returns a minimum lambda for the model in question; This value, in turn, can then be used to calculate the RMSE for the model. The testing set is very large so if we want to evaluate lambdas somewhat efficiently, we first run the data set with broad intervals, and then zoom in on the best performing section - this is the point of the recursive call of the function, named `find_generic_lambda`. The function will also plot the λ vs. RMSE values for both the wide and narrow iteration of the function, choosing the lowest-available value.

The function will also plot both sets of attempted λ's - first, the **wide range** that is set by the first call of the function, and then, the **narrowed range** with much more granular steps. Since this function will be called by every regularized model, we will see the λ ranges plotted appropriately.

4.1 Movie Effect Regularized Model

```
find_generic_lambda <- function(seq_start, seq_end, seq_increment, FUN,
        detailed_flag = FALSE, training_set, testing_set, plot_title="")
{
  lambdas <- seq(seq_start, seq_end, seq_increment)
  #calculate the RMSEs for the FUN - regularized function passed in -
    #with the lambda range created above
  RMSE <- sapply(lambdas, FUN)
  #print(RMSE)
  #plot the output to see the smallest lamdba
  plot <- qplot(lambdas, RMSE, main=plot_title)
  print(plot)
  saveToLocalRepo(plot, repoDir = repo, userTags = c(plot_title))
  #saving the first round lambda - the lowest so far
  min_lambda_first_try <- lambdas[which.min(RMSE)]

  if (detailed_flag)
  {
    #if this is the first iteration of the function, continue with taking a
    #10% lower and 10% higher lambda value to iterate through new lambdas
    #that are much more granuluar, with increments at 10% of what they were previously
    new_lambda_range = (seq_end - seq_start)/40
    min_lambda_first_try <- find_generic_lambda(
     seq_start = min_lambda_first_try - new_lambda_range,
     seq_end = min_lambda_first_try + new_lambda_range,
     seq_increment = seq_increment/10, FUN, detailed_flag = FALSE,
```

```
      training_set = training_set,  testing_set = testing_set,
     plot_title=
       get_narrowed_plot_name(plot_title))
  }
  return (min_lambda_first_try)
}
```

The first model to use this function will be the **Regularized Movie Effect Model** - essentially we are re-using the movie-effect model from above with regularization. The general idea of penalized regression is to control the total variability of the movie effects; Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2+\lambda\sum_i b_i^2$$

The first term of this equation is the least squares estimate we've been working with throughout; the second term is basically a penalty that gets larger when many b_i are large. Using calculus, we can re-work this equation to show that the values of b_i that minimize the equation are:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}\left(Y_{u,i}-\hat{\mu}\right)$$

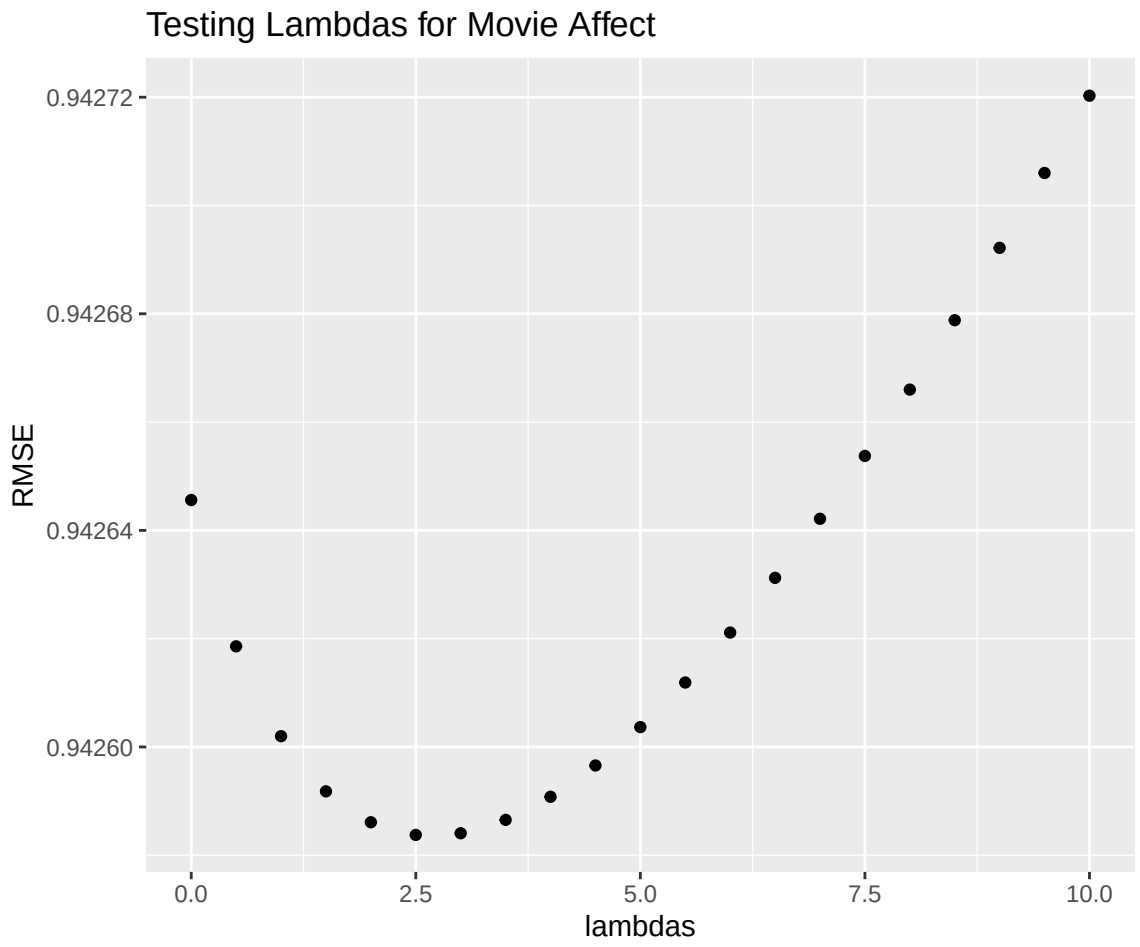
where n_i is the number of ratings made for movie i. When the sample size is large, the penalty λ is so low that it's basically ignored; however, when the sample size is small, the estimate $\hat{b}_i(\lambda)$ diminishes to 0.

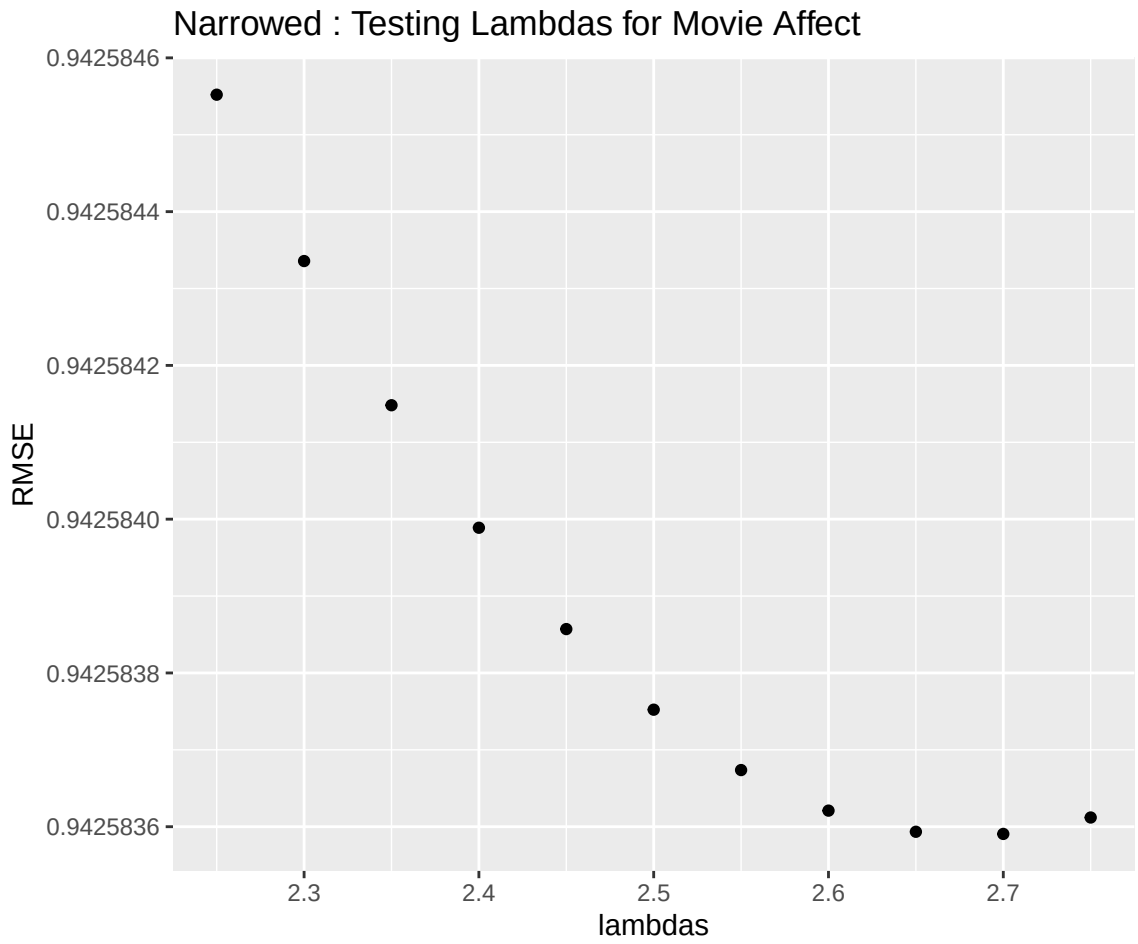
λ is a tuning parameter, and we use cross-validation to choose the correct lambda for each case we decide to plug in regularization into. This, in essence, is what the function **find_generic_lambda** shown above is all about.

```
#trying regularization next; first need to find the correct lambda -
#tuning parameter: The testing set is very large so if we want to evaluate
#lambdas, we first run the data set with broad intervals, and then
#zoom in on the best performing section
regularized_rmse_3 <- function(l, training_set, testing_set)
{
  #print(l)
  mu <- mean(training_set$rating)
  just_the_sum <- training_set %>%
    group_by(movieId) %>%
    summarize(s = sum(rating - mu), n_i = n())

  predicted_ratings <- testing_set %>%
    left_join(just_the_sum, by='movieId') %>%
```

```
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  l_rmse <- RMSE(predicted_ratings, testing_set$rating)
  return (l_rmse)
}
```

The λ values generated with this function, both the wide and narrowed ranges, can be seen on these plots:





The lowest-chosen λ value is: **2.7** .

The function **`regularized_rmse_3`** was passed as a parameter into the **`find_generic_lambda`** function described above, and will produce the RMSE for the regularized movie effect model. It is of note that the result of this is only slightly better then the non-regularized Movie Effect model (model # 2) and significantly worse then the non-regularized model # 3. Moreover, an RMSE of **0.9425836** is very high, and does not fit the criteria of being **< 0.8649**. Therefor, the search for a fitting RMSE continues.

method	RMSE
Regularized + Movie Effect	0.9425836

4.2 Movie + User Bias Effect Regularized Model

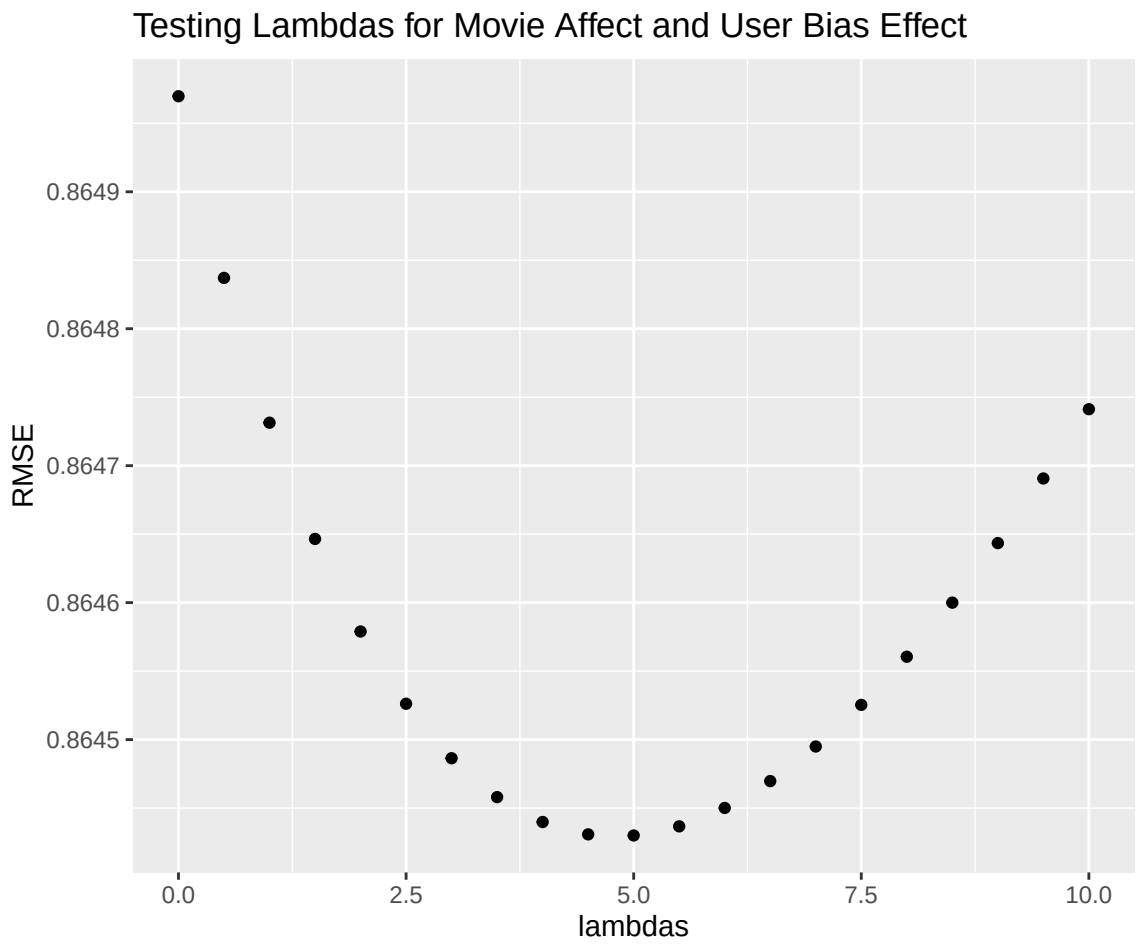
The results from the regularization approach don't seem to be doing any better then the movie Effect model, and significantly worse then the user Effect + movie effect model. We will now try the regularization option with both movie and user bias effect. Therefor, I will next attempt to tune the model more with the both the **Movie Effect** AND the **User Effect** parameter.

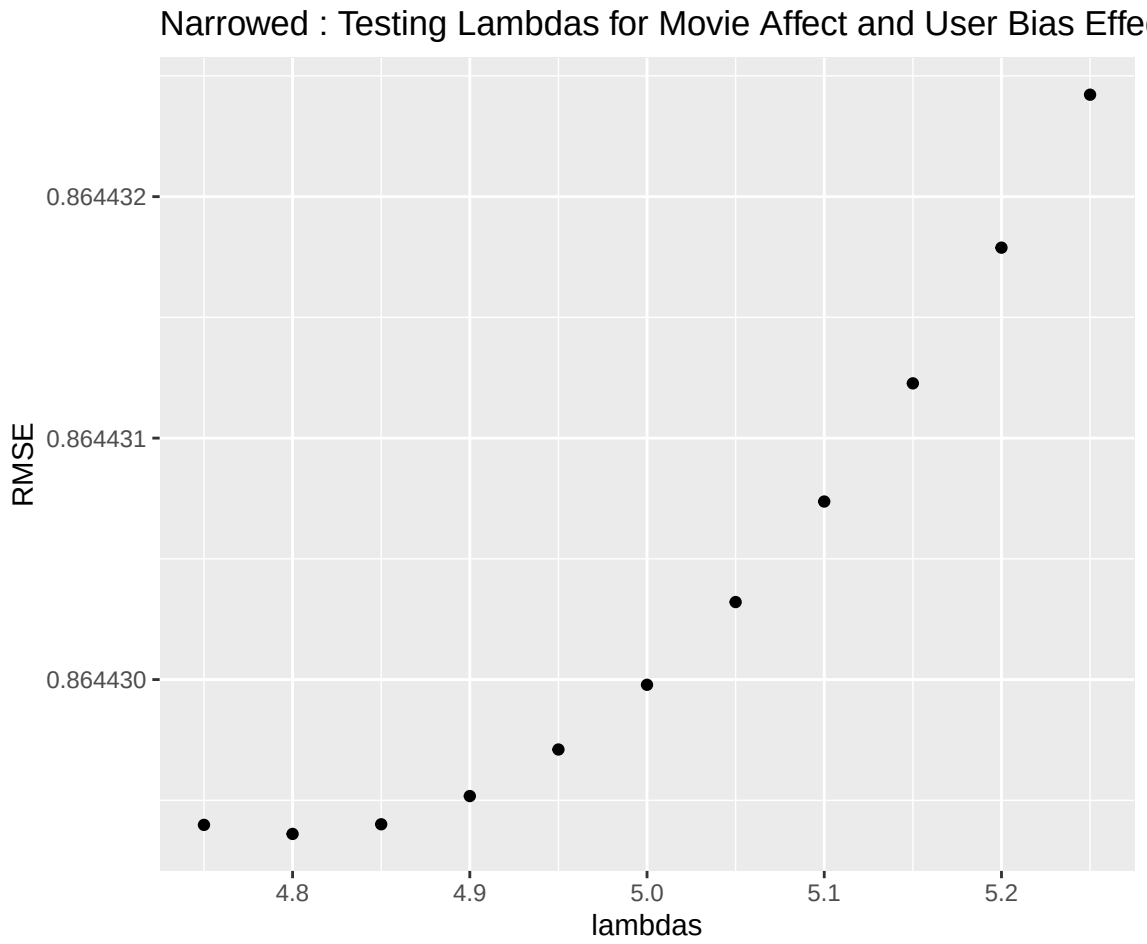
This time, the formula we will follow is, with b_u as the user Effect:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

```
#the lambda needs to be selected using cross-validation, as well.
#We do this as follows, using the same lambda-creation sequence:
regularized_movie_and_user <- function(l, training_set, testing_set)
{

  #print(l)
  mu <- mean(training_set$rating)

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

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

  predicted_ratings <-
    testing_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testing_set$rating))
}
```

The λ's from this function can be seen here:





The lowest-chosen λ value is: **4.8** .

method	RMSE
Regularized Movie + User Effect Model	0.8644294

It seems like the regularization model with both user and movie effects works very well with the RMSE, and even performed slightly better then the current winner, **the user Effect + movie effect model**. But is an even better approach possible? I will next attempt to add the film's release year into the mix and examine whether this will help making the predictions more accurate.

4.3 Movie + User + Release Year Effect Regularized Model

But can we do even better? Next, I will attempt to add in the year into the mix, testing whether the age of the movie makes a difference. For this we have created the field "age_of_movie"

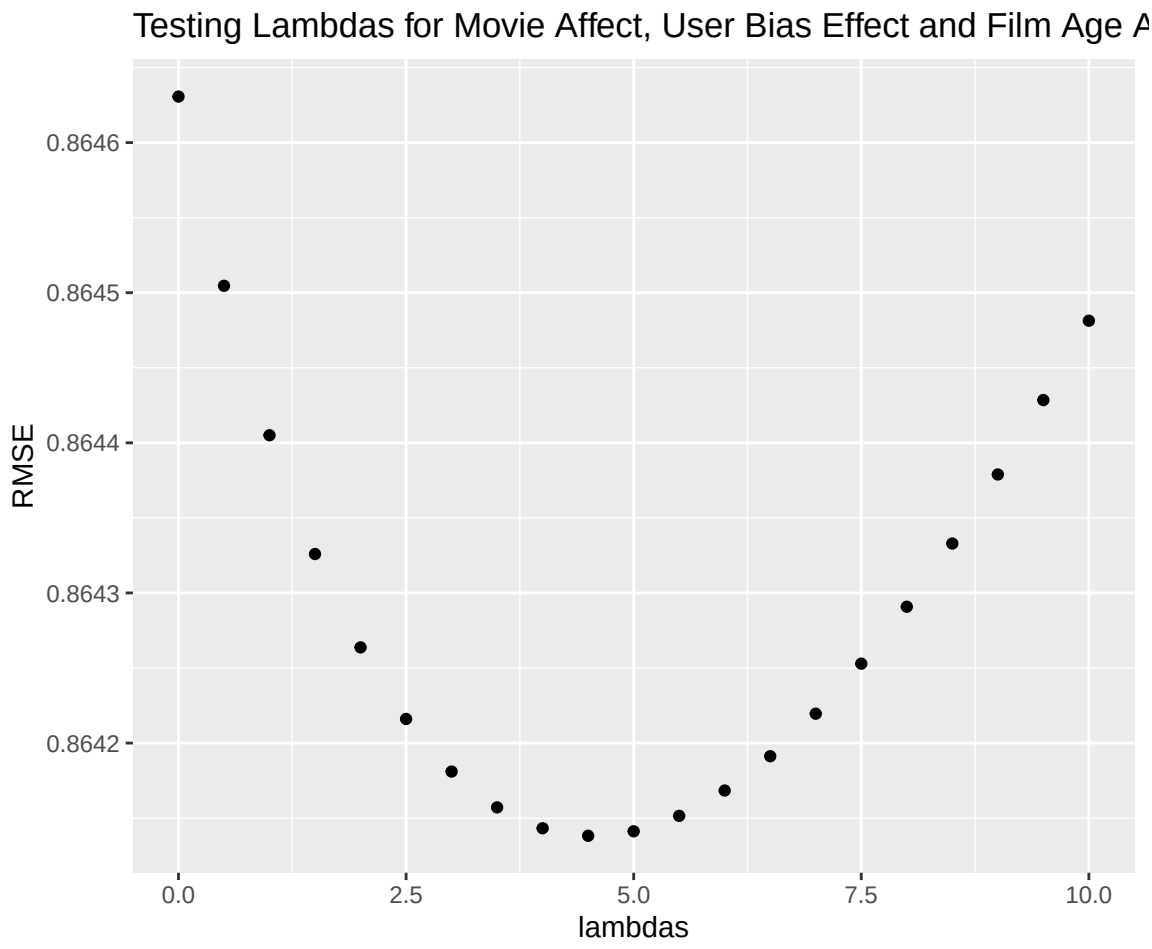
$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u-b_y\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_i b_y^2\right)$$

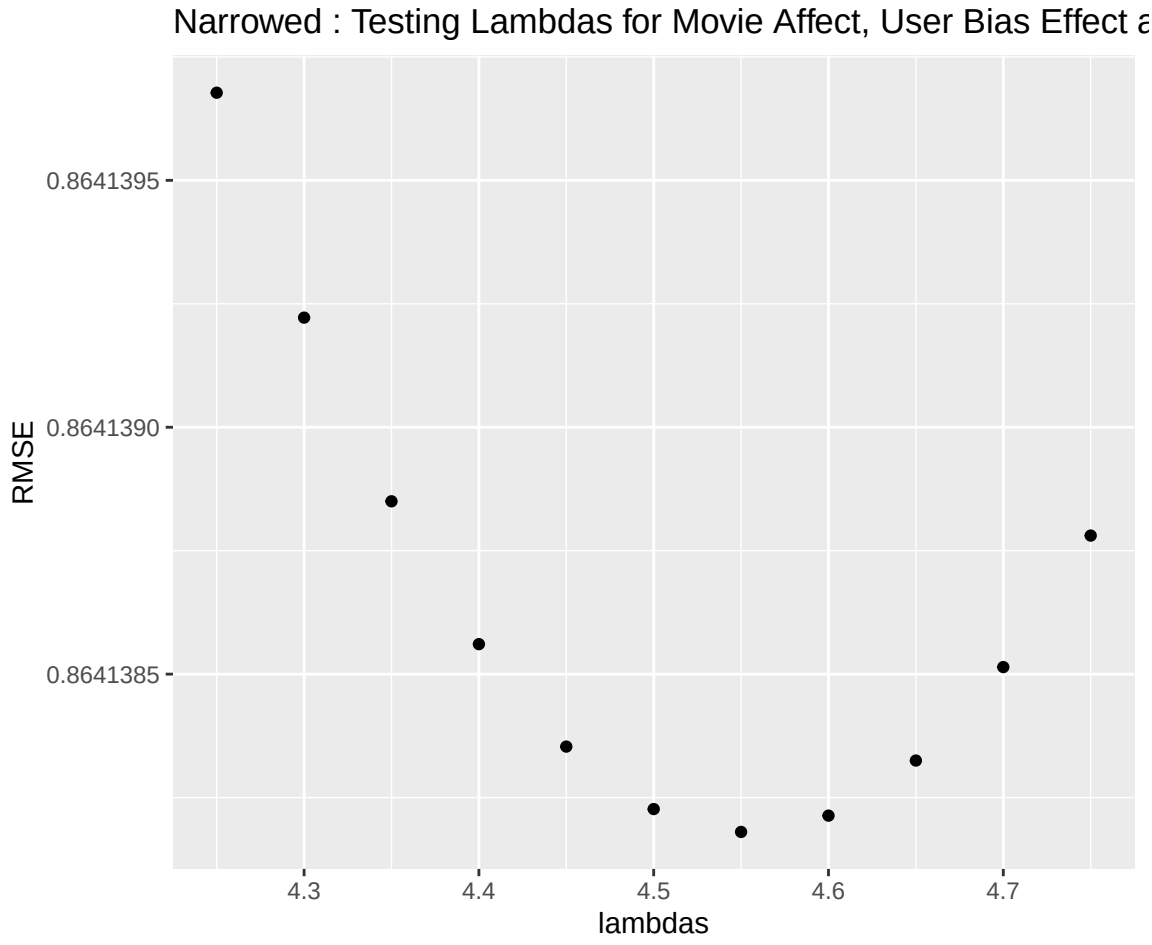
with b_y as the **release year effect**.

```
#Next, I will attempt to add in the year into the mix,
#testing whether the age of the movie makes a difference
#For this we have created the field "age_of_movie"

regularized_movie_and_user_and_year <-
  function(l, training_set, testing_set)
{
  mu <- mean(training_set$rating)
  #print(l)

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

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

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

  predicted_ratings <-
    testing_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "age_of_movie") %>%
    mutate(pred = mu + b_i + b_u + b_y) %>%
    pull(pred)

  rmse <- RMSE(predicted_ratings, testing_set$rating)
```

```
  return(rmse)
}
```

The λ's from this function can be seen here:





The lowest-chosen λ value is: **4.55** .

method	RMSE
Regularized Movie + User Effect Model + Year Effect Model	0.8641382

The added year does indeed seem to have made a difference in the calculations; The next item to be tested is the genre.

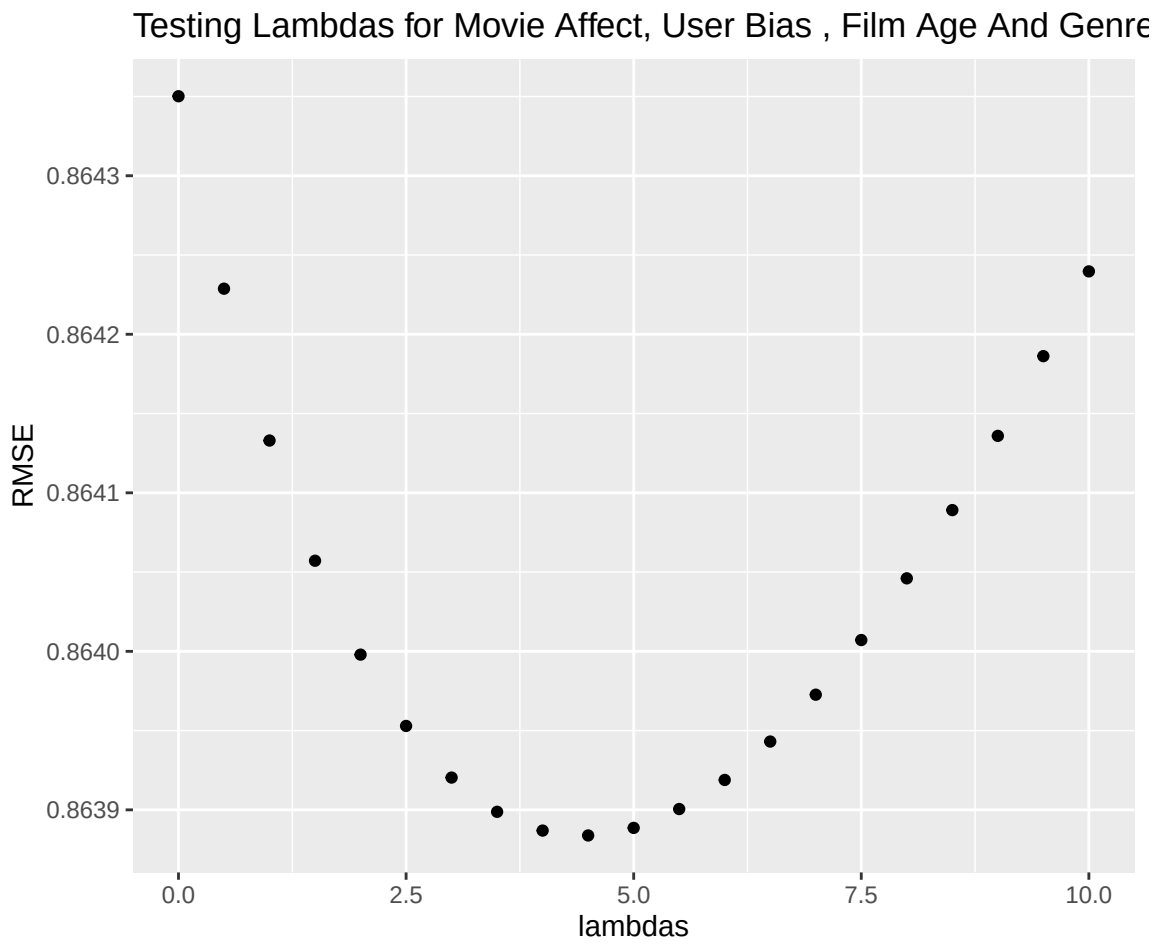
4.4 Movie + User + Release Year + Genre Effect Regularized Model

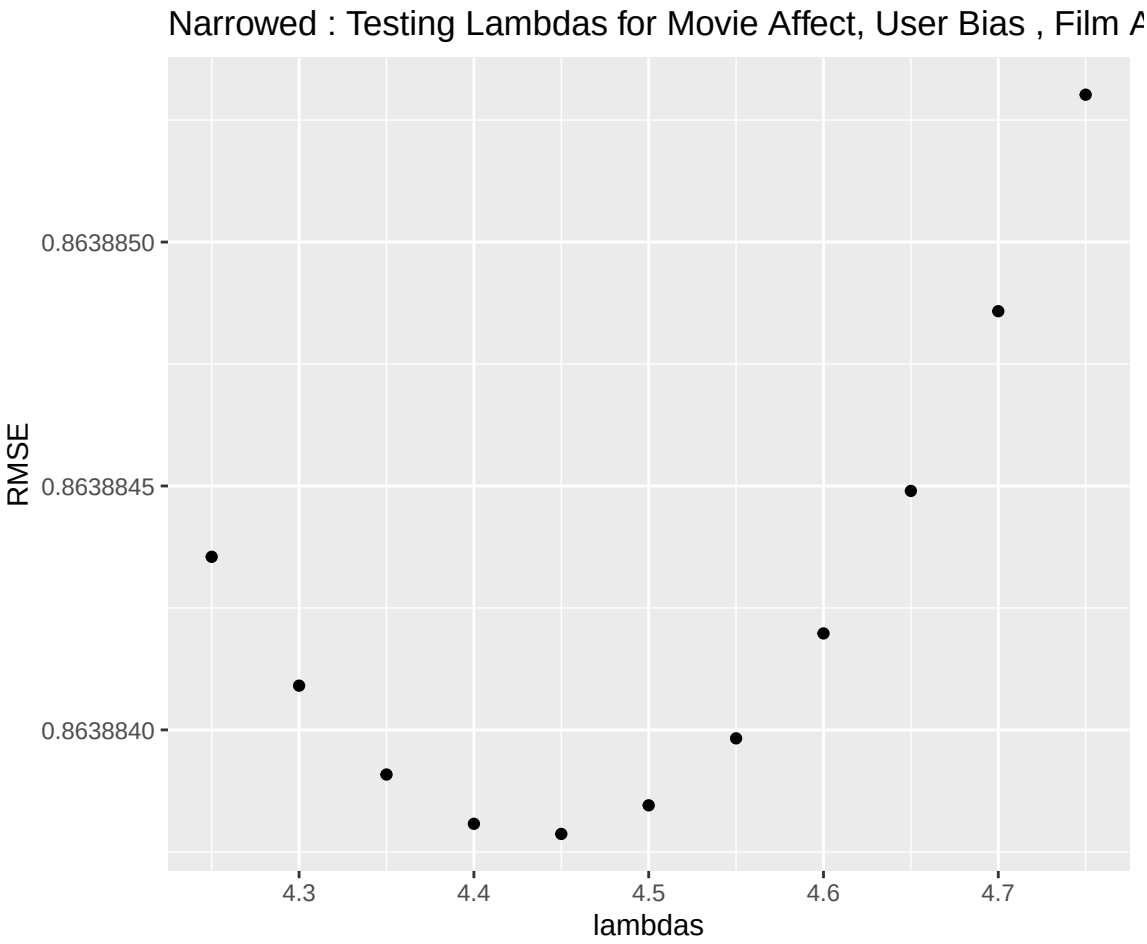
Following the pattern of our previous regularized models, we add in b_g to represent the **genre effect** into the regularization mix, using the following formula:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u-b_y-b_g\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_i b_y^2+\sum_i b_g^2\right)$$

```r
regularized_movie_and_user_and_year_and_genre <-
  function(l, training_set, testing_set)
{

  mu <- mean(training_set$rating)

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

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

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

  #adding in the genre bias here
  b_g <- training_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="age_of_movie") %>%
    group_by(genres) %>%
    summarize( b_g = sum(rating - b_i - b_u - b_y - mu)/(n()+l))


  predicted_ratings <-
    testing_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "age_of_movie") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    pull(pred)

  rmse <- RMSE(predicted_ratings, testing_set$rating)
  return(rmse)
}
```

The λ's from this function can be seen here:





The lowest-chosen λ value is: **4.45**.

method	RMSE
Regularized Movie + User Effect Model + Year Effect Model + Genre Effect Model	0.8638838

Results

After creating a number of predictive algorithms, the best model as per the project requirements is the one with the least RMSE : **'Regularized Movie, User, Release Year and Genre Effect Model'** (This is model **#7** from the list enclosed below). Now that we've selected this model, it will be applied to the **validation data set** (which has been untouched as of yet) with the λ value we calculated with the last function.

The final RMSE value calculated from the validation set is: 0.8646469.

0.8646469 is well below the required threshold of **0.8649**.

We can now review the complete set of RMSE calculations :

	method	RMSE
1	Just the average	1.0591359
2	Just Movie Effect Model b_i average	0.9426456
3	User Affect + Movie Effect Model	0.8649698
4	Regularized + Movie Effect	0.9425836
5	Regularized Movie + User Effect Model	0.8644294
6	Regularized Movie + User Effect Model + Year Effect Model	0.8641382
7	Regularized Movie + User Effect Model + Year Effect Model + Genre Effect Model	0.8638838
8	Final Model Tested on Validation Set	0.8646469

It is apparent from the numbers presented above that without regularization we cannot achieve RMSE numbers less then the threshold value of **0.8649**.

I wanted to examine options for improvement of the algorithm and introduced regularization into the models. The simplest regularization model, using only the movie affect with an optimized lambda, was not good enough for our calculations, and actually regressed from our third model (shown above). However, regularization with movie AND user effect beat the previously best-performing model. Adding in the release year in model # 6 increased our performance even further, and finally, using regularization combined with the movie, user, release year and genre effects proved to be the most efficient model of the models tested.

Applying this final model, #7, to the validation set, gave us a validated RMSE of 0.8646469.

Conclusion

Report Summary

In this project, I have built a machine learning algorithm to predict movie ratings with MovieLens dataset, and estimate their accuracy with RMSE. The **regularized model including the effect of user, movie, release year and genre** is characterized by the lowest RMSE value of all I have examined, and is hence the optimal model to use for the present project - out of the models presented.

The final rating model uses **regularization**, which constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. With a tuning parameter λ to generate a rating $Y_{u,i}$ for movie i by user u. These calculations can be summarized with the following formula:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u-b_y-b_g\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_i b_y^2+\sum_i b_g^2\right)$$

where the variables are:

- μ is the average rating across all movies
- b_i is the per-movie movie bias
- b_u is the per-user movie bias
- b_y is the age of movie (year) bias
- b_g is the genre bias

The first term in the equation above is the mean squared error calculation , and the second term is the penalty term that gets larger when all the effects listed above (movie effect, user effect, release year effect, and genre effect) are large. Re-working the equation using calculus we can show that we need a λ value that will minimize the equation that adds the penalty for all of the effects mentioned.

The **RMSE** arrived at with this model on the **validation dataset** is **0.8646469** - which is sufficiently accurate, as it is lower than the initial evaluation criteria **0.8649** given by the goal of the present project.

As mentioned above, due to the way the function `createDataPartition` works, movies with 3 reviews or less have been excluded from the dataset in order to prevent NA errors in the RMSE calculations.

Other Options

Using Various Regression Functions.

During this process I have also attempted to run the predictions using the `train()` R function, with methods that included Linear Regression ("lm"), LASSO regression ("glmnet"), and Generalized Linear Regression ("glm"). When conducted on a small part of the data set, these functions produced an inferior RMSE result, not in line with the threshold required. More importantly, all 3 of these methods crashed the system when attempted to be run on the full dataset - showing that these are not realistic tools to be used for datasets of this size (in the millions).

Possible Improvements

I think improvements on the RMSE could be achieved by evaluating the project with other means. Different machine learning models could also improve the results further, but hardware limitations (memory and processing power) may be a constraint.

The possible evaluation models we can also attempt to get even better results would include:

- Matrix factorization / Single Value Decomposition (SVD) / Principal Component Analysis (PCA)
- Gradient Descent
- SGD Code Walk

Appendix - Environment

```
print("Operating System:")
```

```
[1] "Operating System:"
```

```
version
```

```
               _
platform       x86_64-apple-darwin17.0
arch           x86_64
os             darwin17.0
system         x86_64, darwin17.0
status
major          4
minor          0.2
year           2020
month          06
day            22
svn rev        78730
language       R
version.string R version 4.0.2 (2020-06-22)
nickname       Taking Off Again
```