Capstone Project HarvardX PH125.9x:Report
July 26, 2021

Contents

HarvardX PH125.9x: Capstone Project: MovieLens Modeling

1 Introduction

The objective of this capstone project is to develop/identify a machine learning (ML) model that will predict movie ratings based on the data contained in the MovieLens database.

The MovieLens database utilized for this capstone project contains 10 Million ratings of 10,000 movies by 72,000 users (SA Ignite 2021). For the history and context of the MovieLens database see Harper and Konstan (2016). Movies are classified by genre, e.g. *Romance*, or by multiple genre, e.g. *Adventure/Romance/Science Fiction.* Data is captured for each review of a movie with each entry (observation) containing the movie name, reviewers Id and the discrete numerical ratings assigned by the reviewers which quantify their 'recommendation' for each movie. Additional data, such as movie year of release and the date/time of the review, are also captured.

In general, the MovieLens database will be divided into two subsets: edx and validation. The edx subset will then be further divided into training and testing subsets. After an exploratory data analysis (EDA) of the training subset is performed, ML models will be developed on that subset to predict ratings of movies. Various ML techniques covered in the edX coursework will be examined. These various models will then be tested on the test subset so that their ability to correctly predict the recommendation can be quantified using the the root mean squared error (RMSE) metric. The models will be iteratively developed and fine-tuned during this process. A final-model will then be selected based on RMSE results. Finally, we will hold our breath and run the final-model on the validation/final-testing subset, calculate the RMSE and submit the results in this paper! Re-runs/fine-tuning on the validaton/final-testing subset is not allowed so as to simulate a real-world situation. This project report concludes with some thoughts and recommendations on model(s) performance.

The report is organized as follows:

- *Methods and Analysis* - This section of the report discusses the exploratory data analysis of the training subset, develops, presents and evaluates various models developed utilizing the test subset and reports the RMSE accuracy of each model. Based on this analysis a final model is chosen for use.
- *Results* - This section discusses the performance of the final model in predicting the reviewer recommendations in the validation/final-testing subset. This model will be run only once utilizing the validation/final-testing subset and the performance metric will be reported.
- *Conclusion* This final section provides a summary discussing this work and its limitations with the suggestion for possible future work.

All programming is conducted in R (R Core Team 2019) with various additional library packages such as *caret*. This document was created in RMarkdown (R Studio 2020) and this is my first paper that utilizes it – so I learned a lot, and still have a lot to learn!

2 Methods and Analysis

Within this section, the *Load Data* subsection briefly discusses the creation of the three MovieLens subsets. Then *Exploratory Data Analysis (EDA)* subsection loads the training and testing subsets and analyzes the training subset to gain insights that might help determine the best ML modeling approach. Various candidate models are then presented and discussed in the *Models* section. The candidate models will be tuned and

tested on the testing subset with performance data calculated and tabulated in an iterative process. Based on this process, a final model will be chosen in the *Final Model* subsection.

2.1 Load Data

The MovieLens database was first divided into two sets: *edx* and *validation* based on directions from and programming code supplied by Professor Rafael A. Irizarry (Irizarry 2009) which is presented here for completeness, but not discussed. The edx subset is then partitioned randomly on an 80%/20% basis into the *train/training* and the *test/testing* subsets. These two subsets will be utilized to develop and compare the performance of various models run with the testing data, and to chose a final-model. The *validation* subset is held-out for one-time (and only one-time!) use to assess the performance of the chosen final-model.

The *validation* subset contains approximately 10% of the data and is set aside for later use; the *edx* subset contains 9,000,055 rows of data; the *train* subset and *test* subsets created from *edx* contain 7,200,045 and 1,800,010 rows, respectively. The *validation* subset just created is set aside for one-time use in validating the final model prior to submission of this paper.

2.2 Exploratory Data Analysis (EDA)

The structure of the *train* data frame is given as:

```
str(train)
```

```
## Classes 'data.table' and 'data.frame':   7200045 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  185 292 316 329 355 356 362 364 370 377 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838983525 838983421 838983392 838983392 838984474 838983653 838984885 838983707 838984596 838983834 ...
##  $ title    : chr  "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" "Star Trek: Generations (1994)" ...
##  $ genres   : chr  "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" "Action|Adventure|Drama|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

There are 7,200,045 observations of six variables in the *train* subset as specified above. All variables are discrete. The following subsections discuss each of the six variables. Note that the metrics provided, e.g. counts or percentages, are derived from the *train* subset.

2.2.1 The *movieId* and *title* Fields in the *train* Subset

The variable *movieID* is a nominal, albeit numerical, value to uniquely identify the movie. The numerical value of this variable does not indicate any order or interval it simply serves as a reference or name. (In one of his lectures, Professor Iziarry discussed Nominal, Ordinal, Interval and Ratio data. The canonical reference is Stevens 1946 which is hilarious to read.) Movie Titles are character strings with names of the movie and (at least some, maybe all, include) the calendar year of release. Note that some title data is entered in a way to facilitate sorting, e.g. the movie *The Net* is listed as *Net, The*.

The rating for a movie may change over time; for example, I favor black & white film-noir movies which were once popular in the United States from the late 1930's into the early/mid 1950's, but these days, this genre is generally out-of-favor. Since the *timestamp* of the movie review is provided (discussed later) which includes the year the rating was made, the 'spread' between the date of the movie release and the date of the movie review might be something utilized to support the models. As such, a new field will be added to the dataframe, *myr*, which will hold the year of the movie's release. The code creating *myr* will be included as a pre-processor section of the models, including the final model.

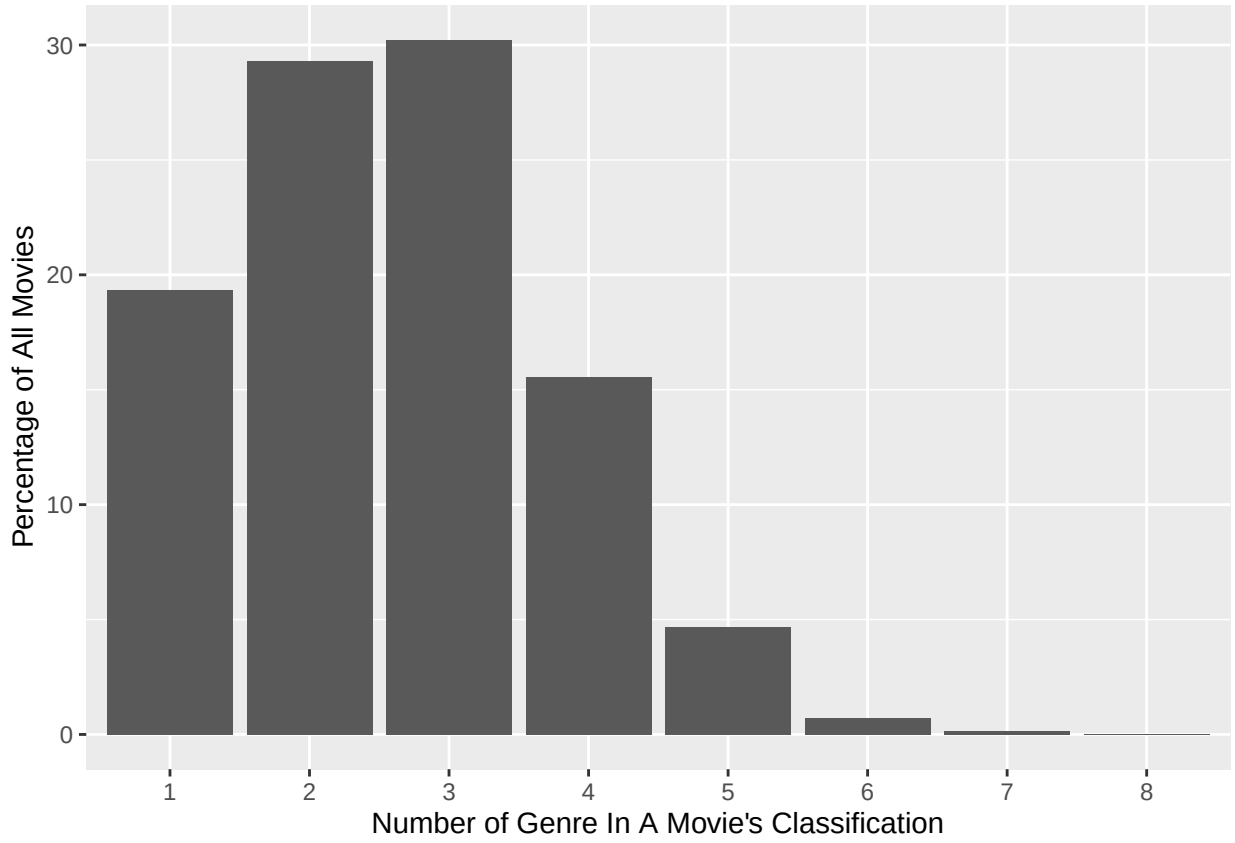
2.2.2 The Movie *genre* Field in the *train* Subset

Movies are classified into single or multiple genres and for some movies the genre is *no genre is listed*.

When multiple genre are listed for one movie, they appear to be listed in alphabetic order for each movie. There are unique genre (or combinations of genre) in the *train* data, for example:

```
uniquegenre[1:10]
```

```
##  [1] "Action|Crime|Thriller"
##  [2] "Action|Drama|Sci-Fi|Thriller"
##  [3] "Action|Adventure|Sci-Fi"
##  [4] "Action|Adventure|Drama|Sci-Fi"
##  [5] "Children|Comedy|Fantasy"
##  [6] "Comedy|Drama|Romance|War"
##  [7] "Adventure|Children|Romance"
##  [8] "Adventure|Animation|Children|Drama|Musical"
##  [9] "Action|Comedy"
## [10] "Action|Romance|Thriller"
```



The number of genre assigned to an individual movie is shown above. The percentage of movies assigned a single genre is 19.34%; while those movies with 2 or 3 genres is 59.54%. The 20 single genres in the train subset, listed alphabetically, are:

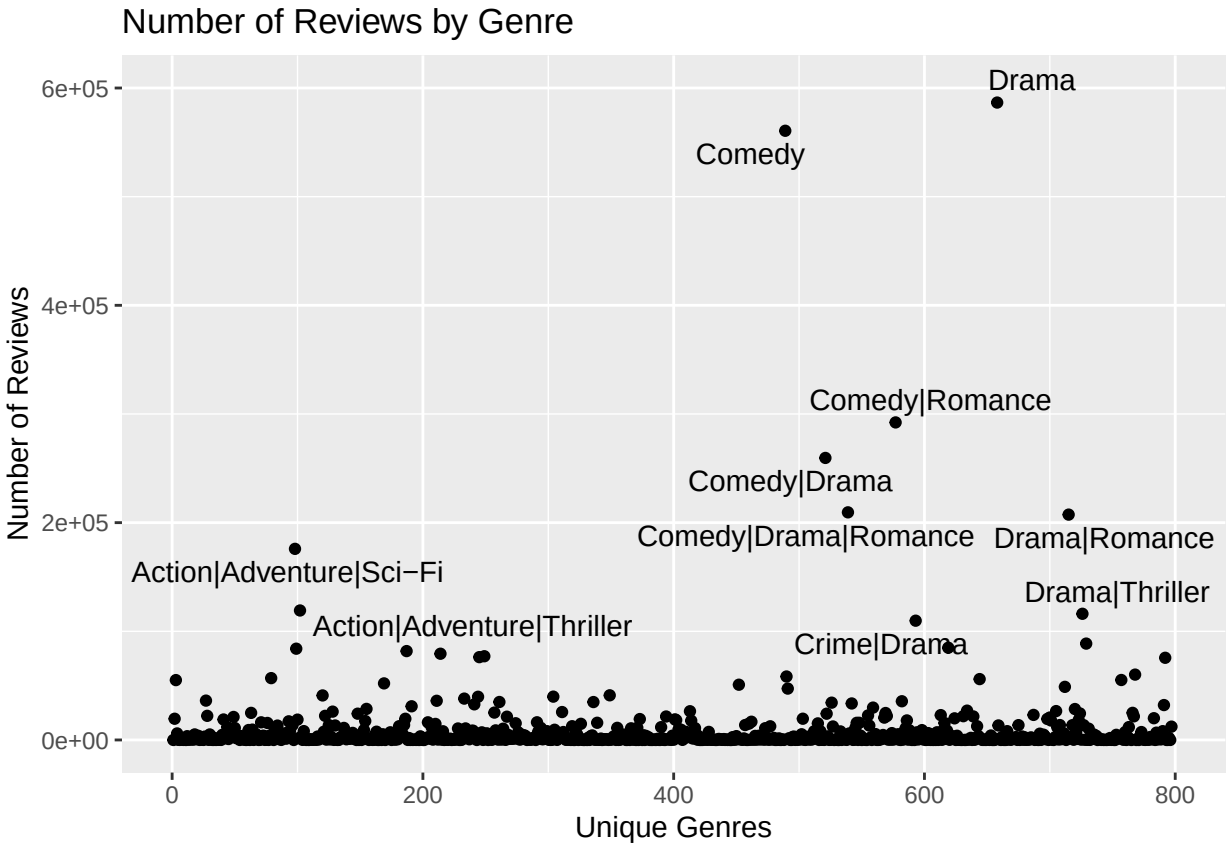
```
##  [1] "(no genres listed)" "Action"             "Adventure"
##  [4] "Animation"          "Children"           "Comedy"
##  [7] "Crime"              "Documentary"        "Drama"
## [10] "Fantasy"            "Film-Noir"          "Horror"
## [13] "IMAX"               "Musical"            "Mystery"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

For those movies with 2 genres, there are 109 unique genre combinations; for those movies with 3 genres, there are 257 unique genre combinations.

For each movie with a multiple genres, e.g. “Children|Comedy|Fantasy”, because the genre appear to be listed alphabetically it cannot be assumed that this movie is primarily for “Children”, secondarily a “Comedy” and thirdly a “Fantasy”: the genre are not ordered by importance or preponderance, but simply

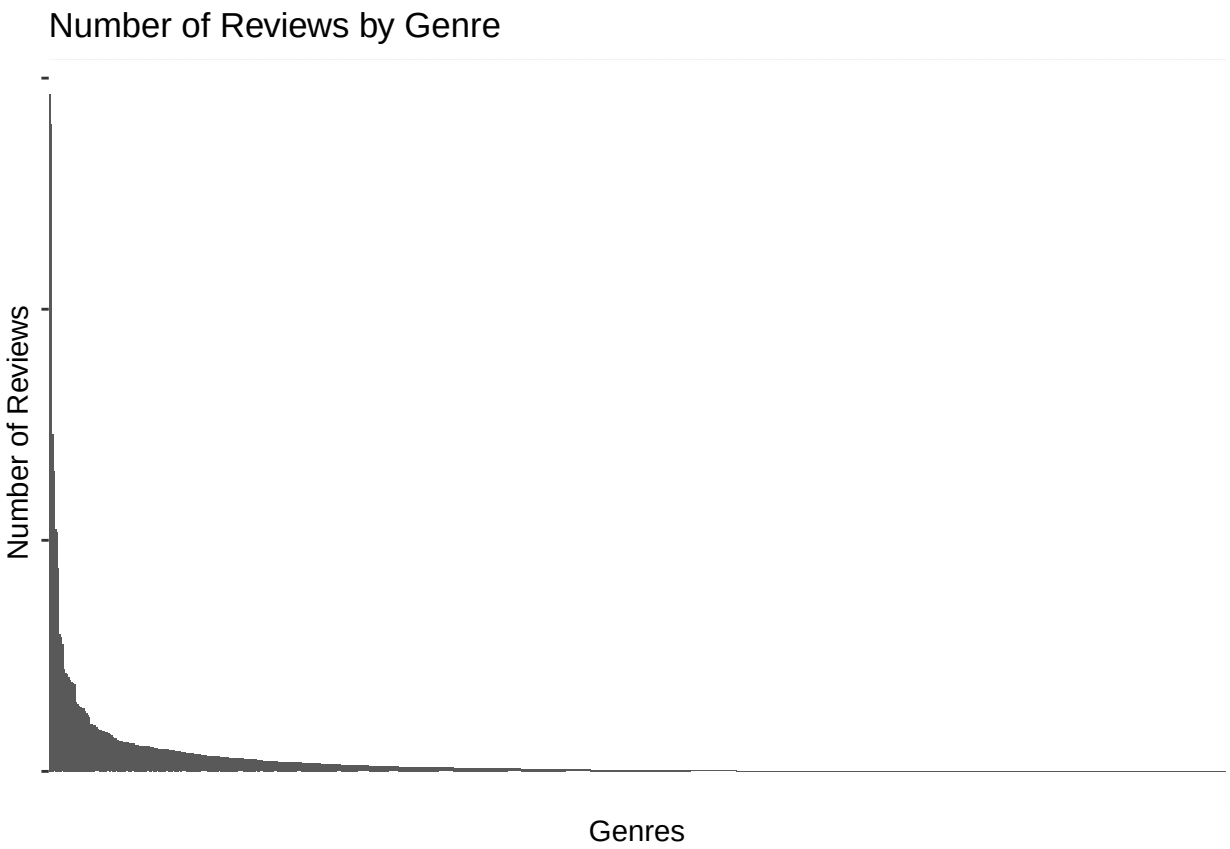
ordered alphabetically. It might be tempting to reduce the number of unique genre to a more manageable number by condensing or summarizing the genre. For example, a (secondary) vector could be added to the dataframe that would classify each movie by the first two genre entries, e.g. "Children|Comedy|Fantasy" and "Children|Comedy|Fantasy|Sci-Fi|Thriller" would be reclassified as "Children|Comedy." If the genre were ordered by importance, this might be a reasonable method to test, however, since the genre are simply alphabetized, it does not seem viable.

The above discussion briefly looked at genres and the number of movies within each genre. The number of movie reviews within each genre can also be examined. An unordered scatter plot of the number of reviews in each of the 797 unique genres is shown below. Those genres with more than 100,000 movie reviews are labeled. Reading down the y-axis: *Drama* is the genre most reviewed, followed by *Comedy* and *Comedy|Romance*, etc.

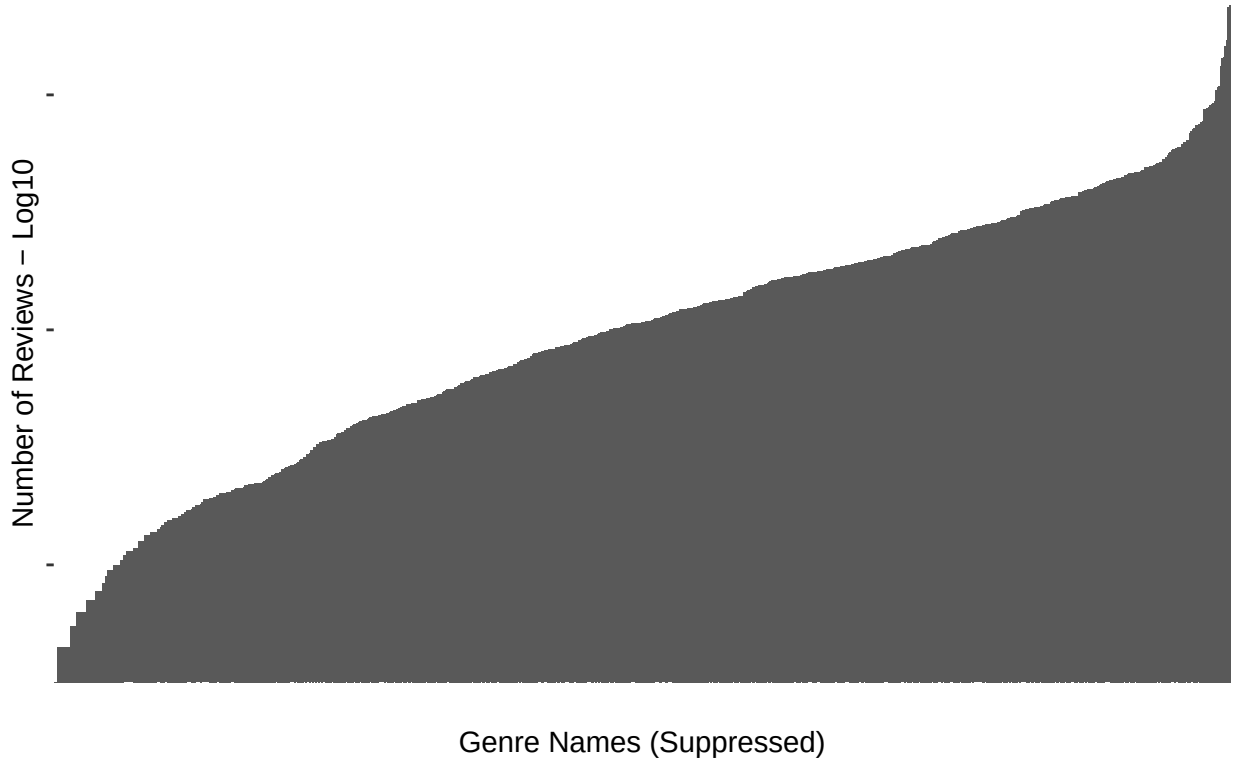


Another possible way to deal with the multiple genres might be to reclassify each movie into one of the more predominant genre (or genres): drama, comedy, comedy|romance, etc., down to some specificity level (number of reviews) on the y-axis, above. This, however, does not seem reasonable. For the moment, we will leave the genre list as it is.

As shown by the bar-graph, figure 1, below, the distribution of reviews by genre-type seem to have a very steep 'exponential' range.



In order to get a better feeling for how skewed the data is, the y values are re-plotted on a log10 scale below.



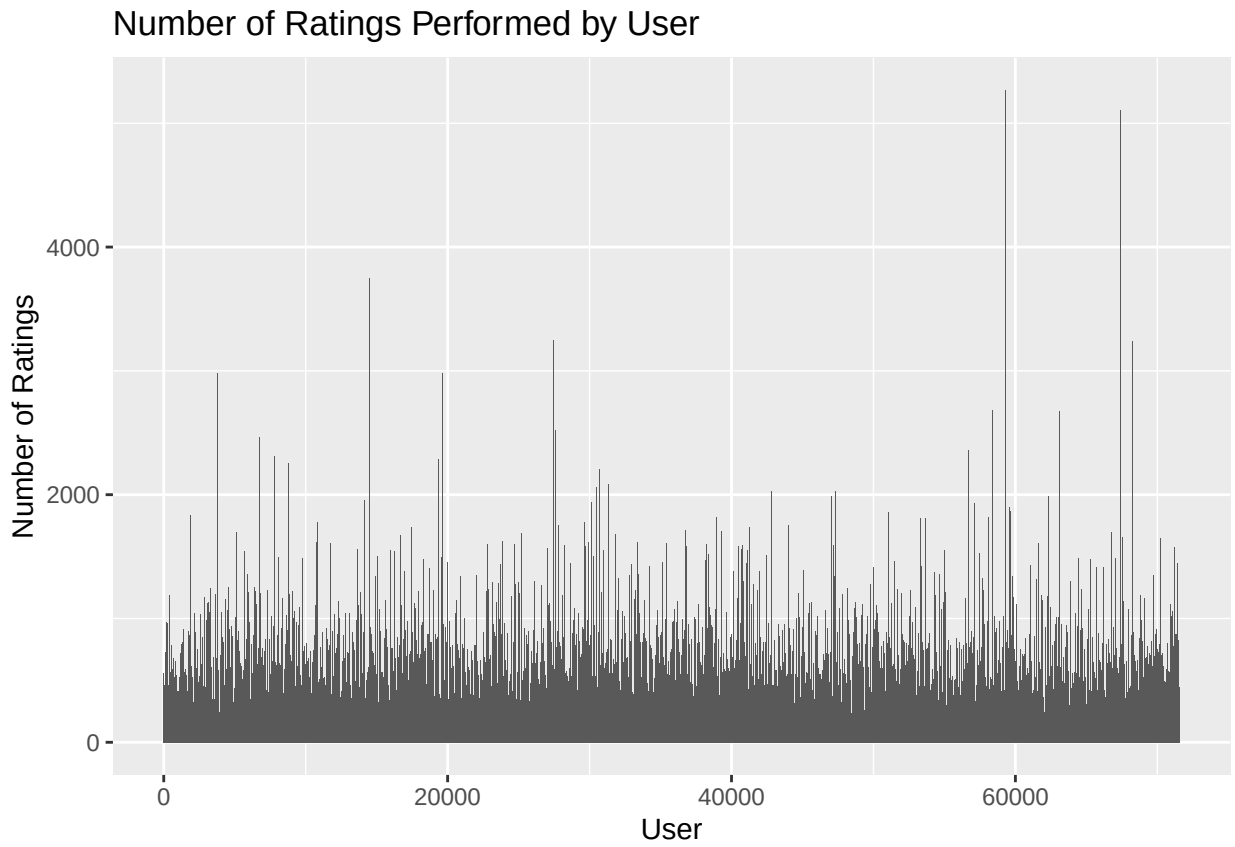
Twenty-five percent of the genres have 147 or less reviews. The median of the reviews by genre is 1,164 which means that 50% of genres have less than 1,164 reviews and 50% of the genres have more. The mean of reviews by genre is 9,033.93 which is significantly greater than the median, showing that the distribution of reviews by genre is heavily skewed to the right – high-values for several specific genre, e.g. *Drama*, *Comedy*, et al. as shown in the figure above, pull the mean much higher than the median. The wide range of the number of reviews by genre may contribute to the problem of over-fitting model to the training data set.

2.2.3 The Movie *title* and *movieId* Fields in the *train* Subset

There are 10,647 unique movies in the *train* subset. There are 69,878 unique reviewers.

2.2.4 The *userId* Field in the *train* Subset

The number of movie ratings by userId is shown in the plot below.

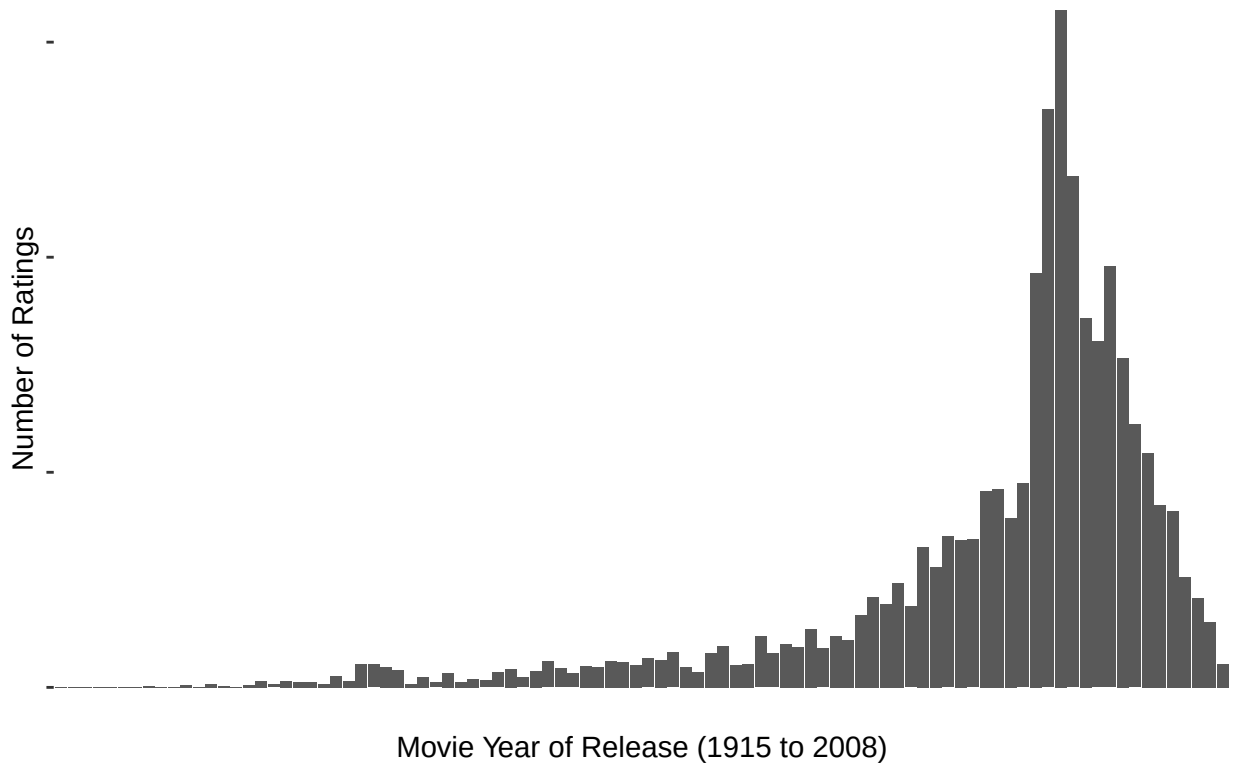


As shown above, the number of reviews by individual reviewer varies widely. This will be addressed in the *Regluarization* section of the report, below.

2.2.5 The *rating* and *timestamp* Fields in the *train* Subset

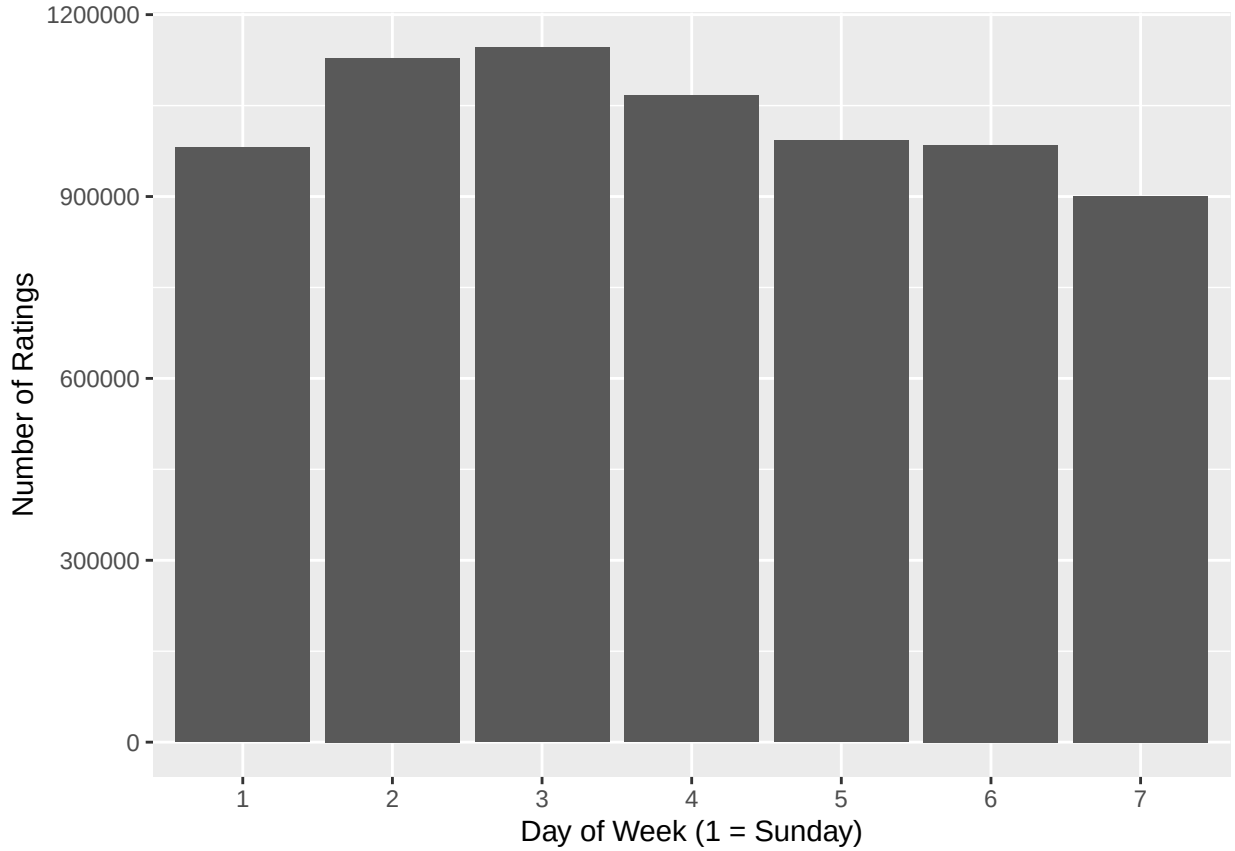
A function was written to strip the movie release year from the title vector and to place it as the *myr* vector in the *train*, *test*, and *validation* subsets. The number of ratings by movie release year in the *train* subset is shown below.

```
#length(years)
library(dplyr)
right <- function (string, char) {
  yr <- substr(string,nchar(string)-(char-1),nchar(string))
  substr(yr,1,4)
}
```



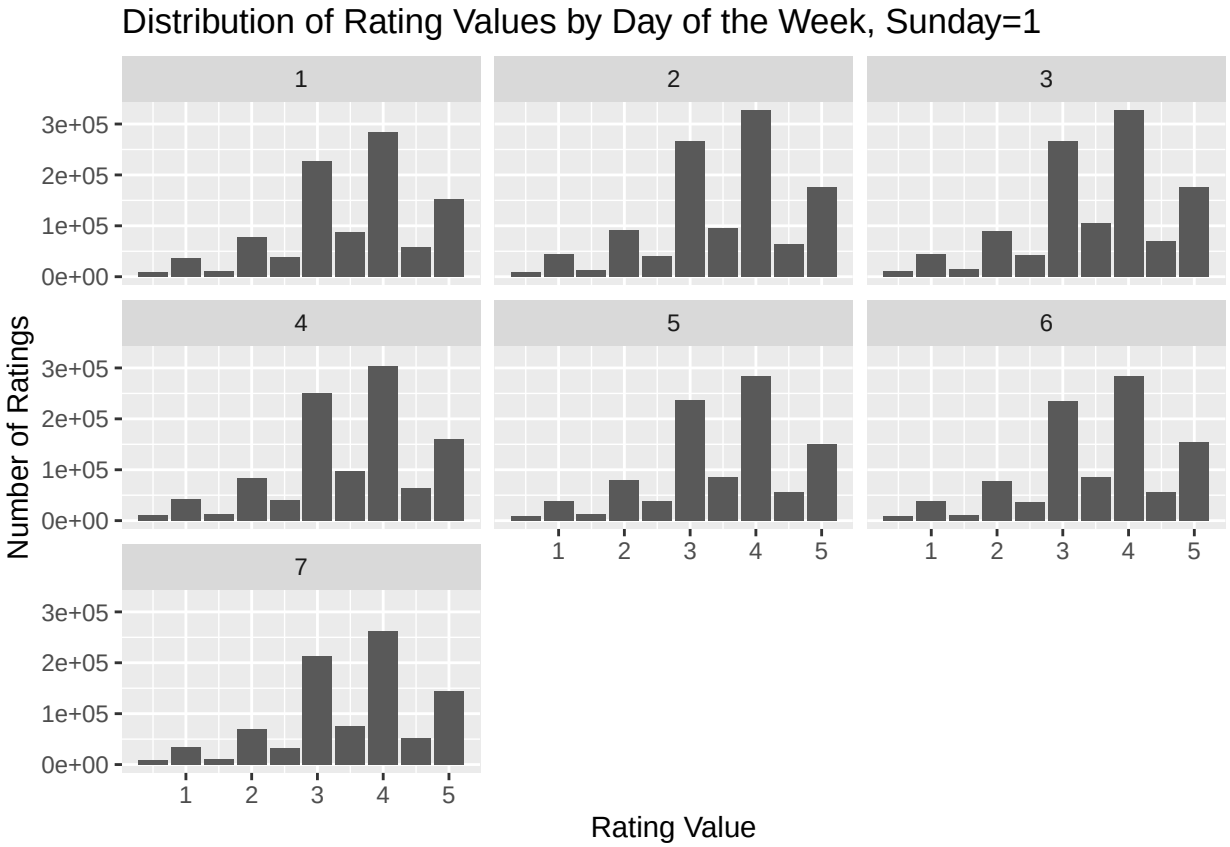
The *timestamp* for each review, e.g. 838,983,392, counts the seconds since midnight Coordinated Universal Time (UTC) since January 1, 1970 (Harper and Konstan 2016). For this example, the date/time is 1996-08-02 10:56:32. The earliest date/time of a movie review in the train subset is 1995-01-09 11:46:49; the date/time of the last movie review in the subset is 2009-01-05 04:55:03. It is assumed that movie reviewers may be located world-wide, so without knowing the geographic location (timezone) of a particular reviewer, not much can be made of the *hour* data; whether a particular review is made late in the day or early in the morning cannot be determined. Using the functions contained in the *lubridate* library, the date/time will be parsed into the following component parts: year(numeric), month(numeric:1-12), and day-of-the-week(numeric:1-7, with 1=Sunday) each of which will be added to the *train* dataframe. Note that any fields added to a training data set and used by a model must also be performed on the testing and validation sets prior to evaluating the performance of the model.

The number of movie ratings by day-of-the-week is shown below.



The number of ratings by day-of-the-week is shown above. I would have thought that a disporportionate number of reviews would have been written early in the week – based on a weekend of movie viewing – but it seems the distribution is fairly constant, so there is probably not much to be gained from including the weekday that the review was written in a model.

The mean rating by day of the week is also fairly constant at 3.5. I did not see a reason to include that plot, but below are seven plots showing the ratings distribution by day from which you can see that the shapes of the distributions are more-or-less constant over the span of days. Again, there seems to be no need to consider the day of the week in a model.



Although using the time of the day (AM/PM) is not possible because we do not know the geographic location, and because the ratings dstirbutions are fairly constant across the seven days of the week as shown above, we can probably only reply on the year of the review, So *ryr*, review year, is extracted from the timestamp and included in the *train*, *test* and *validation* subsets.

3 Results

3.1 Evaluation Metric Defined

ML results may be evaluated in various ways as described in Irizarry (2021a): Confusion Matrix, Sensitivity, Selectivity, F-Score, etc. For this project, Root Mean Square Error (RMSE) is utilized. The RMSE is calculated with:

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

3.2 Preparation of the Subset DataFrames

The *train* and *test* dataframes will now be modified as discussed above. Specifically, fields *myr*, *ryr*, *wday* for movie year (integer), review year (integer), and day of the week (factor) will be added to the dataframes. In additon the rating provided to each move, will be converted to a factor.

Following Irizarry (2021b) a RMSE function will be coded for use in generating results from each of the models tested. The code of the RMSE calculations has been modified to ignore NA values in either vector (*true_ratings* or *predicted_ratings*) by using the *complete.cases* built-in function.

```
RMSE <- function(true_ratings, predicted_ratings){
  tmp <- data.frame(cbind(true_ratings, predicted_ratings))
  tmp <- tmp[complete.cases(tmp),]
  sqrt(mean((tmp$true_ratings - tmp$predicted_ratings)^2))
}
```

3.3 Naive Models

3.3.1 Model 1: Naive Guess

As a first model the naive guess is utilized. Movie ratings are predicted simply based on the average rating, *mu_hat,* calculated from the train subset. For movie = i and user/rater = u, the equation to predict the rating, Y, is a simple linear function:

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

```
mu_hat <- mean(train$rating)
```

Across the train subset, mu_hat = 3.512.

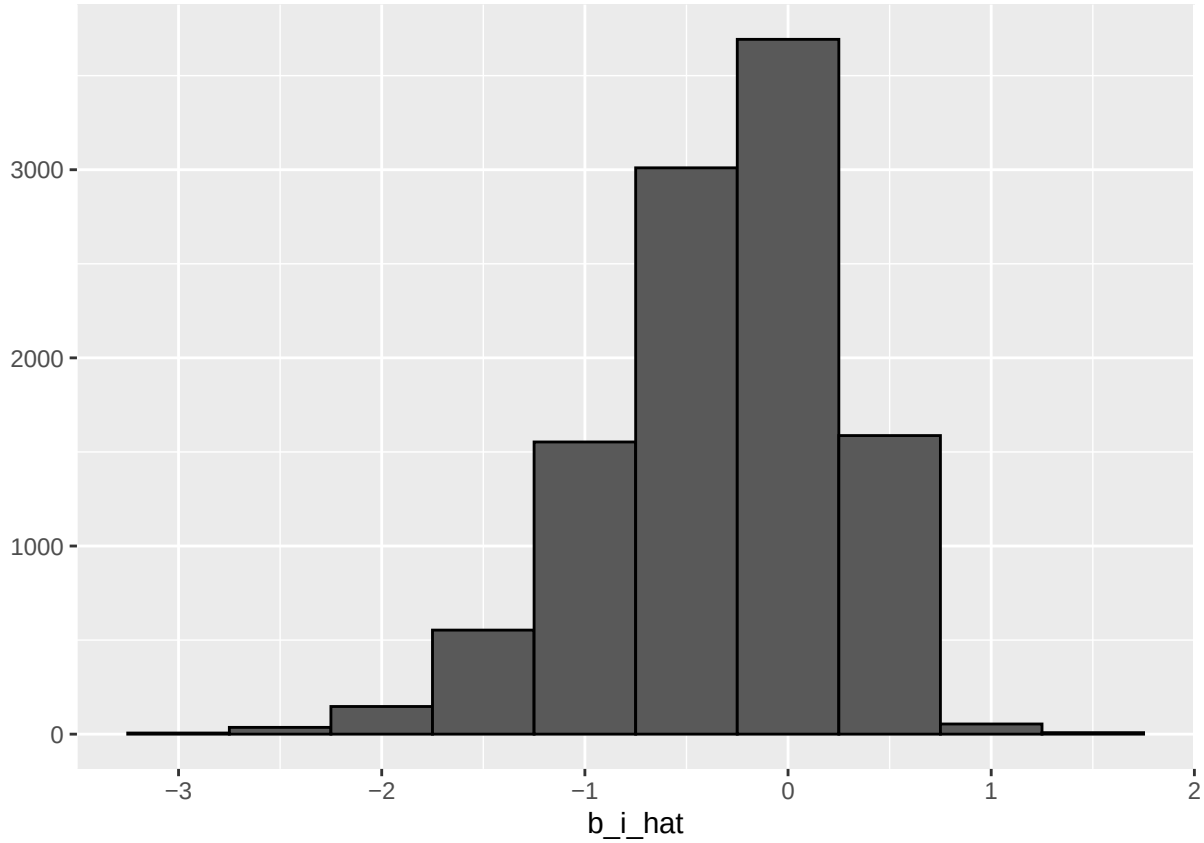
The RMSE for the naive model is 1.06 which is not too good, but the RMSE from this model may serve as an upper limit. Again, following Irizarry (2021b) we will create a results table to hold the results of the various models.

```
rmse_results <- data.frame(Model_Name = "Model 1: Naive Guess", Subset = "Test",RMSE = rmse_model1)
```

3.3.2 Model 2: Naive Guess With *movieId* Bias

Irizarry (2021b) discusses modifying the naive guess model by adding a bias factor. Bias is the average above mu_hat that movies are rated when grouped by movieId. In other words for a given movie (*moveId*) the difference between each actual rating and mu_hat is calculated and these differences are then averaged for each *movieId*.

```
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i_hat = mean(rating-mu_hat))
qplot(b_i_hat, data= movie_avgs, bins=10, color = I("black"))
```



The plot shows that some *movieId*s are above mu_hat (where b_i is greater than zero), but more seem lower than mu_hat (where b_i is less than zero). We can now use the formula

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

and evaluate the performance of this slightly modified model.

```
library(dplyr)
predicted_ratings <- mu_hat + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i_hat)
rmse_model2 <-RMSE(test$rating,predicted_ratings)
```

With this slightly modified model, an RMSE of 0.9432 is achieved which is approximately 11% better than the simple naive model. We will now store these results in the results table.

```
rmse_results <- rbind(rmse_results,data.frame(Model_Name = "Model 2: Naive Guess with movieId Bias", Su
```

3.3.3 Model 3: Naive Guess With *movieId* and *userId* Biases

In a similar way, we can attempt to account for *userId* bias, meaning the average ratings by a specific user above/below the average rating across all movies.

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

```
user_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u_hat = mean(rating - mu_hat - b_i_hat))
```

```
predicted_ratings <- test%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i_hat + b_u_hat) %>%
  pull(pred)
rmse_model3 <- RMSE(test$rating,predicted_ratings)
```

The RMSE for this model is 0.8655 which is 8.236% better than Model 2. predicted_ratings

rmse_results

3.4 Regularization

3.4.1 Model 4: Regularization: Number of Reviews/Movie

Adding the *movieId* and *userId* bias adjustments improved RMSE, but the results are still not very good. If we look at the movies rated best and worse, the sets contain mostly obscure movies which were rated by only a few users. For example, the top 10 rated movies are:

```
#connect movieId to movietitle
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
#
#find 10 best movies
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i_hat)) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "Hellhounds on My Trail (1999)"
##  [2] "Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)"
##  [3] "Shadows of Forgotten Ancestors (1964)"
##  [4] "Fighting Elegy (Kenka erejii) (1966)"
##  [5] "Sun Alley (Sonnenallee) (1999)"
##  [6] "Angus, Thongs and Perfect Snogging (2008)"
##  [7] "Bullfighter and the Lady (1951)"
##  [8] "Human Condition III, The (Ningen no joken III) (1961)"
##  [9] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
## [10] "I'm Starting From Three (Ricomincio da Tre) (1981)"
```

These are fairly obscure – at least to me. The 10 worst rated movies are:

```
##  [1] "30 Years to Life (2001)"
##  [2] "Besotted (2001)"
##  [3] "Hi-Line, The (1999)"
##  [4] "Grief (1993)"
##  [5] "Accused (Anklaget) (2005)"
##  [6] "War of the Worlds 2: The Next Wave (2008)"
##  [7] "SuperBabies: Baby Geniuses 2 (2004)"
##  [8] "Hip Hop Witch, Da (2000)"
##  [9] "Disaster Movie (2008)"
## [10] "From Justin to Kelly (2003)"
```

Again, these movies appear to be obscure, which raises the question: how may reviews did these 'best' and 'worst' movies receive? How often were they rated?

```
##  [1] 1 2 1 1 1 1 1 3 4 2
```

These 'best' movies only received a few ratings each.

```
train %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i_hat) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1]   1   2   1   1   1   1  50  14  27 157
```

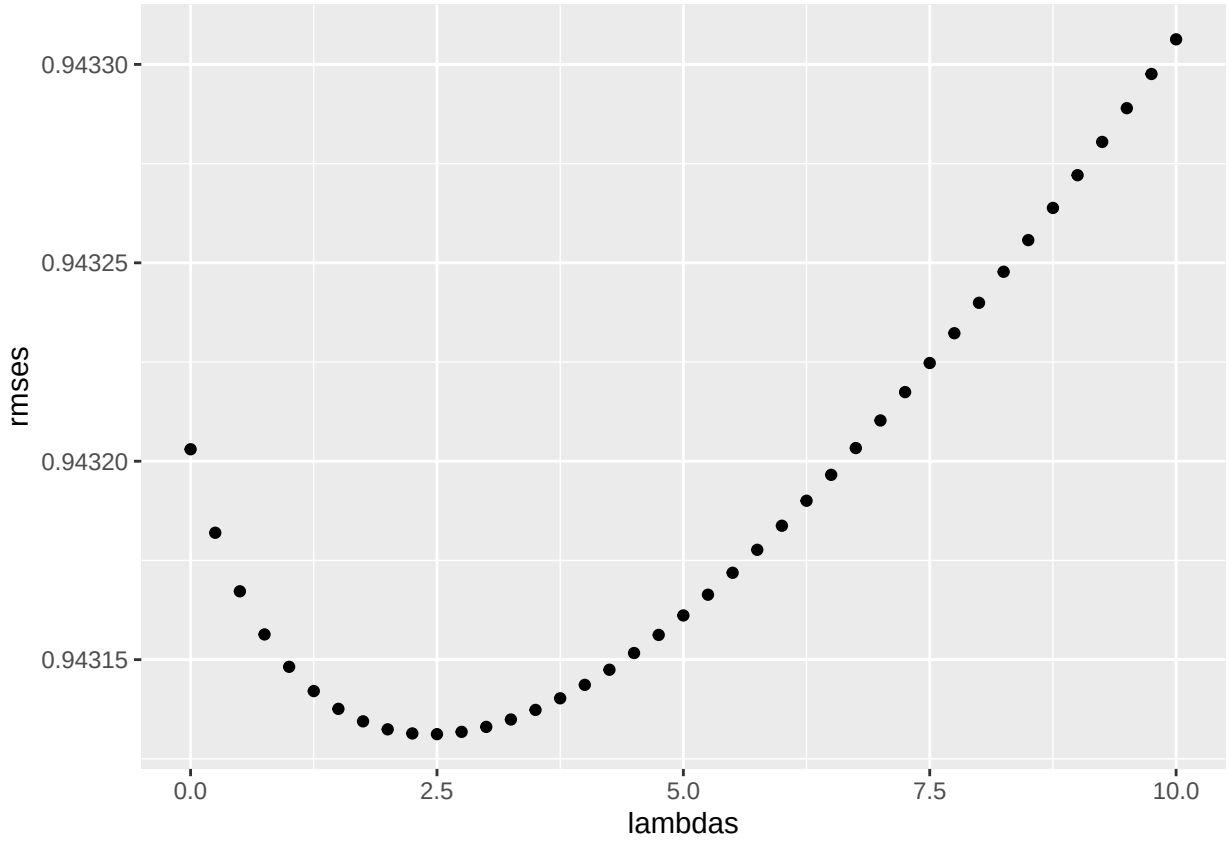
These 'worst' movies only received a few ratings each. In the EDA section above we explored the number of ratings by genre, etc., and saw a wide variation. Here, in both the best and worst movie sets, the number of ratings is very small. How can the number of ratings for a given movie be factored into the over ML model? One way is to add a λ parameter that control a penalty function. The task is to find a λ that minimizes

$$\hat{b_i} = \frac{1}{\lambda + n_i} \sum_{u=1} (Y_{u,i} - \mu)$$

Note that n is the number of reviews for a particular movie (i), and that for large n_i, $\lambda + n_i \approx n_i$, while for movies with a small number of reviews where n_i is small, greater λ will lower $\hat{b_i}(\lambda)$.

Of course, choosing the proper λ without cross-validation would be difficult. Cross-validation for parameter tuning can be utilized to test a sequence of λ parameters, RMSE can be calculated for each and the optimal value can be chosen.

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(train$rating)
just_the_sum <- train %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test$rating))
})
qplot(lambdas, rmses)
```

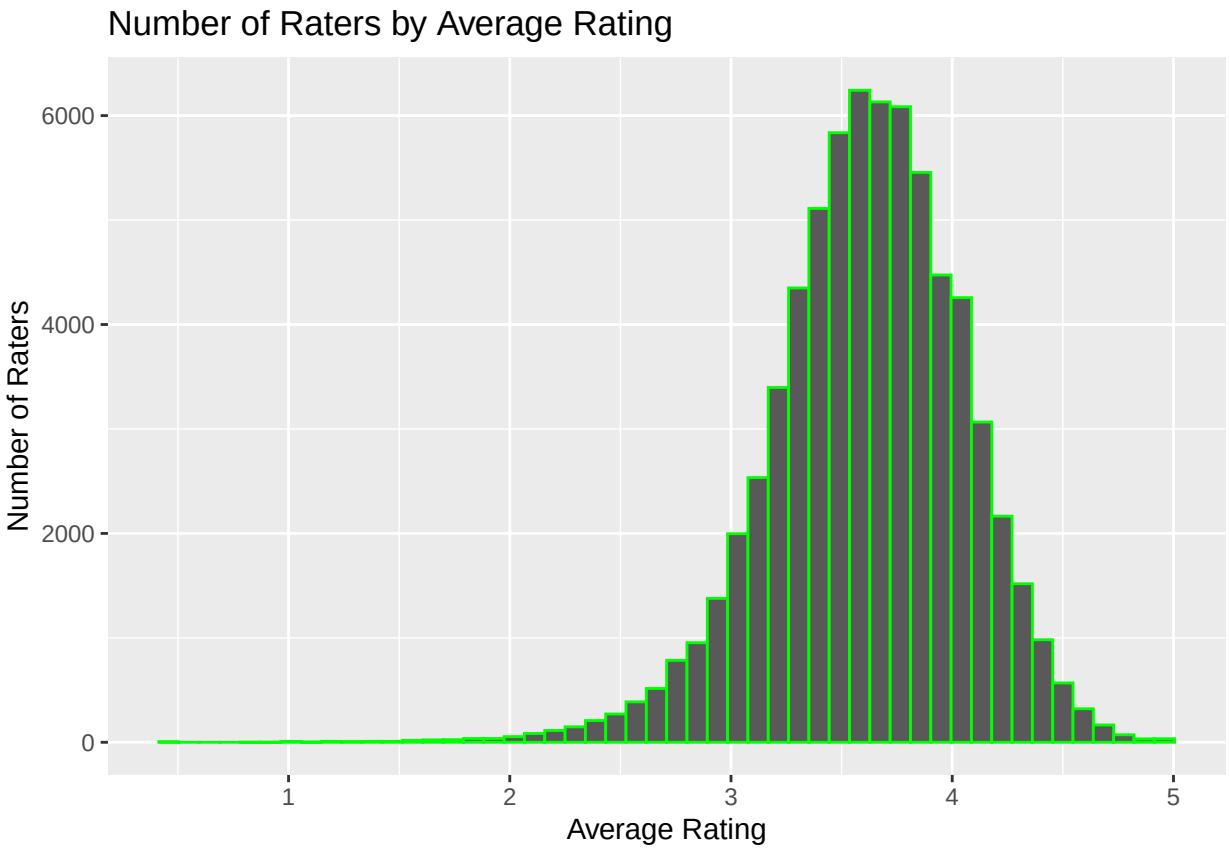



```
min_lambda1 <- lambdas[which.min(rmses)]
min_reg1_rmses <- min(rmses)
```

The plot above shows the relationship between various lambda values and the RMSE which results for each which run against the *test* subset. The lambda associated with the minimum RMSE is $\lambda_1 = 2.50$ which results in an $RMSE = 0.9431$. This is the RMSE when regularizing on the number reviews per movie. We will store this value and try to fine tune this model further by also regularizing the data based on the number of reviewers.

3.4.2 Model 5: Regularization: Reviews/Movie and Reviewers/Movie

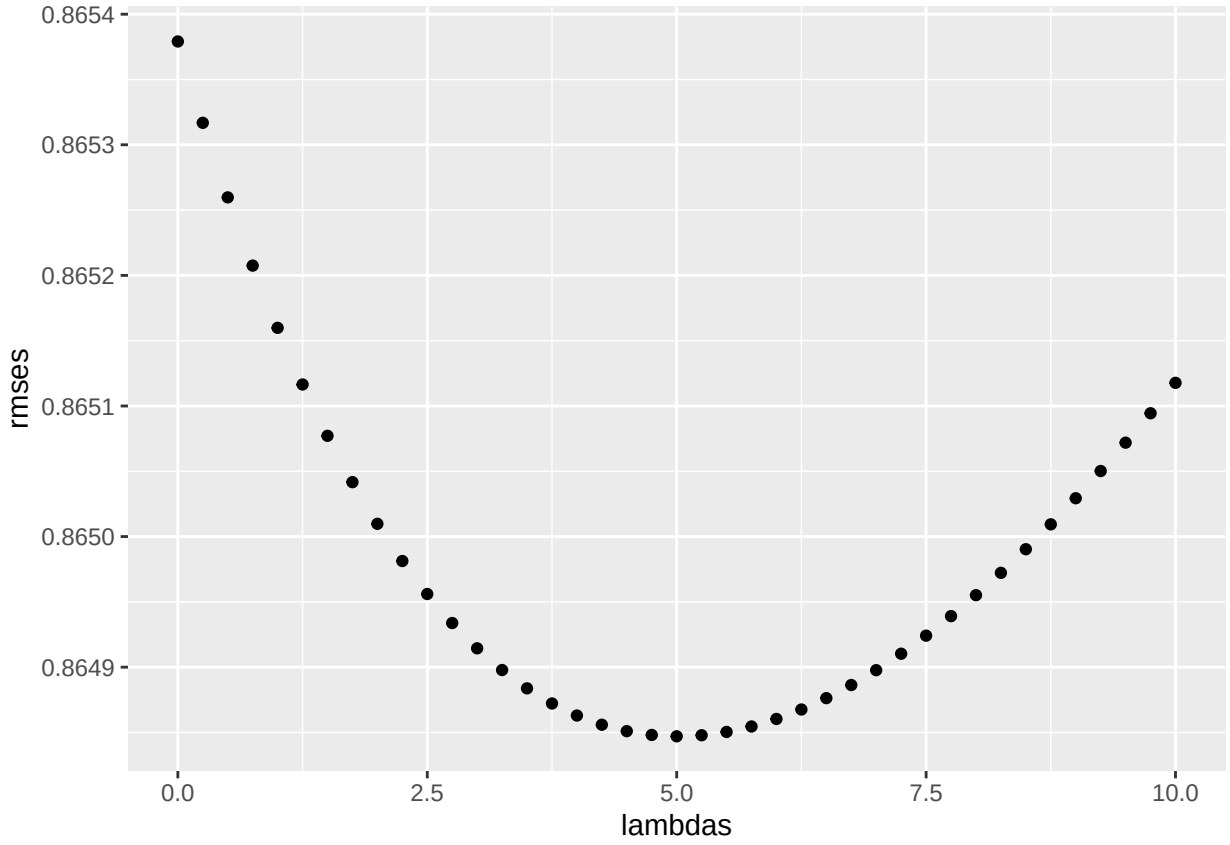
Just as the number of reviews per movie varied widely, the number of reviewers by average rating also varies widely. As shown below only a few raters gave low or high ratings, the majority gave a rating of approximately 3.5. In order to account for this effect, we can again employ the method of regularization to adjust the rating depending on how many raters were involved at that rating level across all movies.



We can use the same process as before to identify the best lambda via cross-validation to adjust for both of these bias-effects.

```
library(ggplot2)
library(dplyr)
lambdas <- seq(0, 10, 0.25)

  rmses <- sapply(lambdas, function(l){

    mu_hat <- mean(train$rating)

    b_i_hat <- train %>%
      group_by(movieId) %>%
      summarize(b_i_hat = sum(rating - mu_hat)/(n()+min_lambda1))

    b_u_hat <- train %>%
      left_join(b_i_hat, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+l))

    predicted_regularization <- test %>%
      left_join(b_i_hat, by = "movieId") %>%
      left_join(b_u_hat, by = "userId") %>%
      mutate(pred = mu_hat + b_i_hat + b_u_hat) %>%
      .$pred

    return(RMSE(test$rating,predicted_regularization))
```

```
})

qplot(lambdas, rmses)
```




```
  min_lambda2 <- lambdas[which.min(rmses)]
min_reg2_rmses <- min(rmses)
```

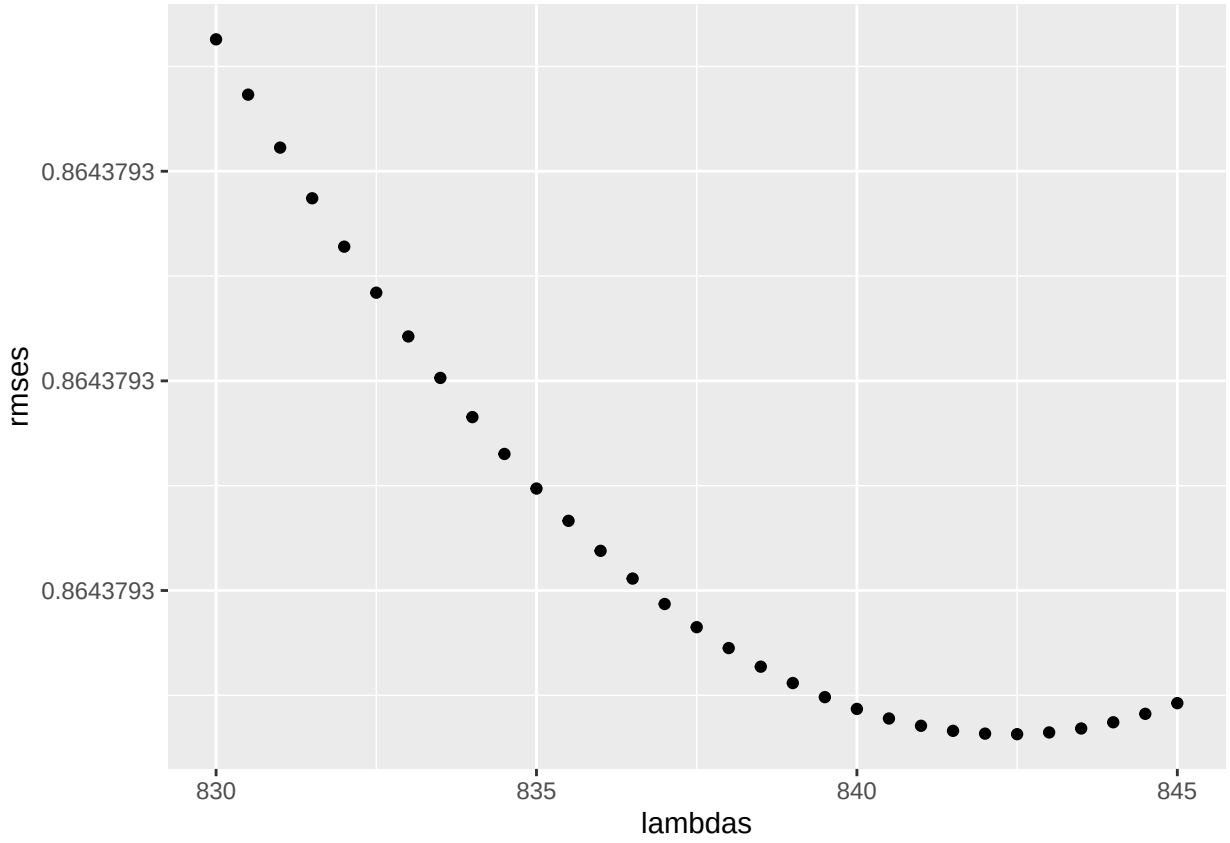
From the plot above, the minimum RMSE of 0.8648 occurs at $\lambda_2 = 5$. Again, remember these results, like the ones above simply test the various models against the *test* set. We will now add this result to the rmse_results table.

3.4.3 Model 6: Regularization: Review/Movie and Reviewers/Movie and Delta Date

In the discussion above, the year of the movie's release and the year of the review were discussed. Many films were released decades prior to being rated by anyone, others (contemporaneous with MovieLens) were rated upon release by some raters and others were rated years after release by others. Vectors of the movie release year, *myr* and the review year, *ryr* were added to the *train* subset. It might be interesting to see what impact the addition of a $\lambda_y r$ parameter to account for the difference between *myr* and *ryr*.To do this we need to add those two vectors to the *test* subset, too; and if we decide to use this model, those vectors must also be added to the *validation* subset, as well. I note that this difference should be non-negative, but in looking at the data in the *train* subset, there are some negative numbers which I conclude originated from missing or corrupted data. I decided to ignore these few problems for the moment, but this should be investigated later.

```
library(ggplot2)
library(dplyr)
lambdas <- seq(830, 845, .5)
```

```
rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(train$rating)

  b_i_hat <- train %>%
    group_by(movieId) %>%
    summarize(b_i_hat = sum(rating - mu_hat)/(n()+min_lambda1))

  b_u_hat <- train %>%
    left_join(b_i_hat, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+min_lambda2))

  b_yr_hat <- train %>%
    left_join(b_i_hat, by="movieId") %>%
    left_join(b_u_hat, by= "userId") %>%
    group_by(delta_yr) %>%
    summarize(b_yr_hat = sum(rating - b_i_hat - b_u_hat - mu_hat)/(n()+l))

  predicted_regularization <- test %>%
    left_join(b_i_hat, by = "movieId") %>%
    left_join(b_u_hat, by = "userId") %>%
    left_join(b_yr_hat, by = "delta_yr") %>%
    mutate(pred = mu_hat + b_i_hat + b_u_hat + b_yr_hat) %>%
    .$pred

  return(RMSE(test$rating,predicted_regularization))
})

qplot(lambdas, rmses)
```




```
min_lambda3 <- lambdas[which.min(rmses)]
min_reg3_rmses <- min(rmses)
```

This model was cross-validated several times with different sequential λ values. This model results in a minimum RMSE of 0.8644 which occurs at $lambda_{yr} = 842.50$. This is not a very large improvement at all, however, a 'large improvement' may simpy result from over fitting. The addition of the *delta_yr* term at least did not decrease RMSE.

At least an inflection point was found, so it can be accepted, still the RSME may not be 'the best' and the RSME may result from overfitting. We are in the dark. It seems strange that that the *delta_yr* value would be divided by such a (relatively) large a number as 842.50. Still this RSME is not much different from the other 'better' group, but how will the model fare with the *validation* subset? Maybe it is over-fitted, who knows. Lurking behind a 'best fit' may be an 'over fit', so choosing simply based on the best RMSE might not be the best method of model selection.

3.5 Regularization with Filling In the Blanks

In processing the *delta_yr* data for Model 6, it was noticed that when the left-Join was completed using *by = "delta_year"* that some *delta_year* entries in the *test* set may not available in the *train* set. This led to considering a method that would 'fill-in' the mean value of *b_yr_hat* to replace each NA that was created by the left_join.

We can refer to this replacement of NAs within the *test* subset with the mean value from the *train* subset as 'filling in the blanks.' Indeed, I have not examined the NA population in the *test* subsets after the b_u_hat left_join, or after the b_i_hat left_join. So maybe we should rerun Model 6 with all the various 'b' NA values 'filled-in' with the respective 'b-means' and evaluate any change in RMSE. We already know the three λ values so cross-validation is not needed, and this model will execute rather quickly. Essentially we calculate the mean for each of *b_u_hat*, *b_i_hat*, and *b_yr_hat* and use them to replace any NA values in those

vectors. With this method, every observation will have a value for these three vectors, even if it is 'only' the mean-value.

3.5.1 Model 7: Model 6 Regularization with Fill In the Blanks

```
    mu_hat <- mean(train$rating)

    b_i_hat <- train %>%
      group_by(movieId) %>%
      summarize(b_i_hat = sum(rating - mu_hat)/(n()+2.5))

    bi_mean <- b_i_hat %>% ungroup()
    bi_mean <- mean(b_i_hat$b_i_hat)

    b_u_hat <- train %>%
      left_join(b_i_hat, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+5.0))

    bu_mean <- b_u_hat %>% ungroup()
    bu_mean <- mean(b_u_hat$b_u_hat)

    b_yr_hat <- train %>%
      left_join(b_i_hat, by="movieId") %>%
      left_join(b_u_hat, by= "userId") %>%
      group_by(delta_yr) %>%
      summarize(b_yr_hat = sum(rating - b_i_hat - b_u_hat - mu_hat)/(n()+842.50))

    byr_mean <- b_yr_hat %>% ungroup()
    byr_mean <- mean(b_yr_hat$b_yr_hat)

    final_test <- test %>%
      left_join(b_i_hat, by = "movieId") %>%
      left_join(b_u_hat, by = "userId") %>%
      left_join(b_yr_hat, by = "delta_yr")

#Here we replace any NA value in the three bias vectors with its mean value
      final_test$b_u_hat[is.na(final_test$b_u_hat)]   <- bu_mean
      final_test$b_i_hat[is.na(final_test$b_i_hat)]   <- bi_mean
      final_test$b_yr_hat[is.na(final_test$b_yr_hat)] <- byr_mean


       final_test <- final_test %>% mutate(pred = mu_hat + b_i_hat + b_u_hat + b_yr_hat)



       model7_results <- RMSE(test$rating,final_test$pred)
```

3.6 Summary of RMSE Results For Each Model

In order to choose among these 7 models, we will review the RMSE of each as measured on the *test* subset.

Model_Name	Subset	RMSE
Model 1: Naive Guess	Test	1.0597347
Model 2: Naive Guess with movieId Bias	Test	0.9432030
Model 3: Naive Guess with movieId and userId Biases	Test	0.8655254
Model 4: Regularize Reviews/Movie	Test	0.9431312
Model 5: Regularize Reviews/Movie and Reviews/Movie	Test	0.8648471
Model 6: Regularize Reviews/Movie and Reviews/Movie and Delta_Year	Test	0.8643793
Model 7: Regularization With Fill In the Blanks	Test	0.8643793

It is hard to determine the best model based on RMSE alone as the minimal RMSE may have resulted not from a robust model, but from an over-fitted one. It looks like the options are between Model 6 and Model 7 with RMSE of 0.864 and 0.865, respectively. I think I will go with Model 7 because I like the use of *delta_yr* AND the 'fill in the blanks' method it utilizes and I am not too sure what to expect with the *validation* subset!

3.7 Final Model Performance on the *validation* Subset

```
mu_hat <- mean(validation$rating)

b_i_hat <- validation %>%
  group_by(movieId) %>%
  summarize(b_i_hat = sum(rating - mu_hat)/(n()+2.5))


bi_i_hat <- b_i_hat %>% ungroup()
bi_mean <- mean(b_i_hat$b_i_hat)

b_u_hat <- validation %>%
  left_join(b_i_hat, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_hat = sum(rating - b_i_hat - mu_hat)/(n()+5.0))

bu_u_hat <- b_u_hat %>% ungroup()
bu_mean <- mean(b_u_hat$b_u_hat)

b_yr_hat <- validation %>%
  left_join(b_i_hat, by="movieId") %>%
  left_join(b_u_hat, by= "userId") %>%
  group_by(delta_yr) %>%
  summarize(b_yr_hat = sum(rating - b_i_hat - b_u_hat - mu_hat)/(n()+842.50))

byr_yr_hat <- b_yr_hat %>% ungroup()
byr_mean <- mean(b_yr_hat$b_yr_hat)


predicted_regularization <- validation %>%
  left_join(b_i_hat, by = "movieId") %>%
  left_join(b_u_hat, by = "userId") %>%
  left_join(b_yr_hat, by = "delta_yr")


#Here we replace any NA value in the three bias vectors with its mean value
```

```
      predicted_regularization$b_u_hat[is.na(predicted_regularization$b_u_hat)]   <- bu_mean
      predicted_regularization$b_i_hat[is.na(predicted_regularization$b_i_hat)]   <- bi_mean
      predicted_regularization$b_yr_hat[is.na(predicted_regularization$b_yr_hat)] <- byr_mean

      predicted_regularization <- predicted_regularization %>% mutate(pred = mu_hat + b_i_hat + b_u_hat

final_results <- RMSE(validation$rating,predicted_regularization$pred)
```

The RMSE result of Model 7 on the *validation* subset is 0.8387.

4 Conclusions

In this ML exercise, I investigated several models for movie recommendations based on the MovieLens data. There are many other models that could have been tested, and should be tested prior to reaching a conclusion, however based on the models examined the ML with regularization seemed to be the best choice. Regularization, to minimize bias in under/over-weighted samples, was performed on movieId and userId. The year the movie was reviewed was unbundled and retreived from the *timestamp* and added to each subset as *ryr*. The year the movie was released was stripped from the *title* and added to each subset as *myr*. A field, *delta_yr* was then added to the three subsets to hold the difference between *ryr* and *myr*. This was also utilized for regularization. Finally, I decided to fill-in any blank bias terms in the regularization model with the mean of that particular bias, e.g. *movieId*, *reviewer*, *delta_yr* which I call Model 7: Model 6 With All Hats! Based on the RMSE achieved, this was was selected as the final model.

When running the final model against the *validation* subset an RMSE =0.8387 was achieved. We can add this to the results table and then list all the models and their results.

Model_Name	Subset	RMSE
Model 1: Naive Guess	Test	1.0597347
Model 2: Naive Guess with movieId Bias	Test	0.9432030
Model 3: Naive Guess with movieId and userId Biases	Test	0.8655254
Model 4: Regularize Reviews/Movie	Test	0.9431312
Model 5: Regularize Reviews/Movie and Reviews/Movie	Test	0.8648471
Model 6: Regularize Reviews/Movie and Reviews/Movie and Delta_Year	Test	0.8643793
Model 7: Regularization With Fill In the Blanks	Test	0.8643793
Model7: Regularization With Fill In the Blanks	Validation	0.8386788

I am not sure what to think of the result of Model 7 being run on the *validation* subset; I think I would want to test my model further. But I am only allowed on run and the deadline for submitting this paper is fast approaching, so this is it!

Also, this is my first time writing an rmarkdown report. I tried to get cross-referencing to work, but was unsuccessful. I did at least get the table of contents and the references processes to work. I need to experiment with kable for better looking tables. In the past most of my reports have been written in Latex.

One concern is the ability of R to actually process large/larger amounts of data. I would have liked to perform a KNN analysis of the data, but R (at least on my system) would not cooperate even on the smaller train subset. I am not sure if that is a limitation of R, or my PC, or both. I will need to look in this question, although at present my processing of (mainly) survey data is on the order of 18K observations of approximately 135 variables which fits easily into my current analytical processes. Another area I need to explore concerns the processing of categorical data and the requirement for converting them to factors for some ML algorithms. When are factors vs. categorical variables required. Another area I want to explore is the normalization of data prior to ML: which algorithms require it, which don't; and should the test subset then be normalized prior to testing the model – I assume the answer is yes.

After this course I plan to collect the end-pages from each of the 3 spiral-bound notebooks I have used during this course where I have noted 'things for further study/investigation.' There are a lot of them! I enjoyed the course very much.
Thank you.

References

Harper, F. M., and J. A. Konstan. 2016. "The Movielens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5 (4): 1–19. https://dl.acm.org/doi/10.1145/2827872.

Irizarry, R. A. 2009. "HarvardX Ph125.9x: Capstone Project." 2009. https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+1T2021/home/.

———. 2021a. "Introduction to Data Science: Data Analysis and Prediction Algorithms with R." 2021. https://rafalab.github.io/dsbook/introduction-to-machine-learning.html.

———. 2021b. "Introduction to Data Science: Data Analysis and Prediction Algorithms with R, Section 33." 2021. https://rafalab.github.io/dsbook/introduction-to-machine-learning.html.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

R Studio. 2020. "R Markdown." 2020. https://rmarkdown.rstudio.com/.

SA Ignite. 2021. "MovieLens 10M Dataset." 2021. https://grouplens.org/datasets/movielens/10m/.

Stevens, S. S. 1946. "On the Theory of Scales of Measurement." *Science* 103 (2684): 677–80.