MovieLens Project: HarvardX PH125.9x

Authored By: Brad Rossen

Project Duration: May 6th - May 20th

Project Introduction

In this project, we are tasked with developing a machine learning algorithm using the movie ratings data contained in one dataset to predict the movie ratings in our validation dataset. The primary goal here is to achieve the highest possible predictive power of our machine learning algorithm. This algorithm with incorporate a 'regularization' effect to account for the effect of extreme outliers and also aim to account for respective biases in the dataset. This 'movie recommendation system' will be created from the MovieLens data set. Through the course of this report, we will show documented findings, explain key steps as we clean and manipulate this large dataset. We will aim to be concise but thorough throughout the report so that we can maximize understanding from the reader. Due to the reader not having a previous understanding of the data or techniques used, this project will err on the side of being extra thorough when explaining the methodology and steps used to answer the key question outlined for this project. ### Preliminary Steps ### Loading Databases & Setting Digit Options

```
# Loading required databases
library(tidyverse)
library(caret)
library(data.table)
library(knitr)
# Load ggplot2 - allows for better and more easily manipulated plots
library(ggplot2)
# Load lubridate - makes it easier to work with date-time data n R.
library(lubridate)
# Load dplyr - packages used for data manipulation in R (filter, mutate etc)
library(dplyr)
# Report up to 5 significant digits - easy to judge the RMSE scores versus our goal of < 0.8649
options(digits = 5)
```

Create Train and Final Hold-out Test Sets - From Course

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

# Note: this process could take a couple of minutes

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

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

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 4.0.5

## Loading required package: lattice

##
## Attaching package: 'caret'

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

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

```
## Loading required package: data.table

## Warning: package 'data.table' was built under R version 4.0.5

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret)
library(data.table)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
```

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

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


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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

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

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

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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

Exploring the Data - Running functions to describe the type and layout of the chosen dataset.

Our first step here is to have a look into the data and obtain some general, descriptive knowledge regarding our dataset. To do this, we are going to use the class(), glimpse() and summary() functions.

```
class(edx)
```

```
## [1] "data.table" "data.frame"
```

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ genres    <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
```

This glimpse() function tells us the 'type' of data that represents each column. For example, we can see that both the 'titles' and 'genres' columns present character data. The glimpse function also shows us that we have data with 6 columns and 9,000,055 rows. The column headers are as follows: userId, movieId, rating, timestamp, title and genres.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The summary() function gives us a lot of useful information regarding the information contained in each of our columns. This function gives us information such as; minimum and maximum values, mean and median. When initiating a new project, it is important and valuable to the data analyst to run these 3 functions at the beginning to gain value insights on your chosen dataset.

Using functions to answer specific data oriented questions.

How many different moviesand users are in the dataset?

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

Which movies have the greatest number of ratings?

```
edx %>% group_by(movieId, title) %>% summarize(count = n()) %>% arrange(desc(count))
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the `.groups` argument.
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title count
##      <dbl> <chr> <int>
##  1     296 <NA>  31362
##  2     356 <NA>  31079
##  3     593 <NA>  30382
##  4     480 <NA>  29360
##  5     318 <NA>  28015
##  6     110 <NA>  26212
##  7     457 <NA>  25998
##  8     589 <NA>  25984
##  9     260 <NA>  25672
## 10     150 <NA>  24284
## # ... with 10,667 more rows
```

Which ratings have the highest count and lets display those counts

```
edx %>% group_by(rating) %>% summarize(count = n()) %>% arrange(desc(count))
```

```
## # 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
```

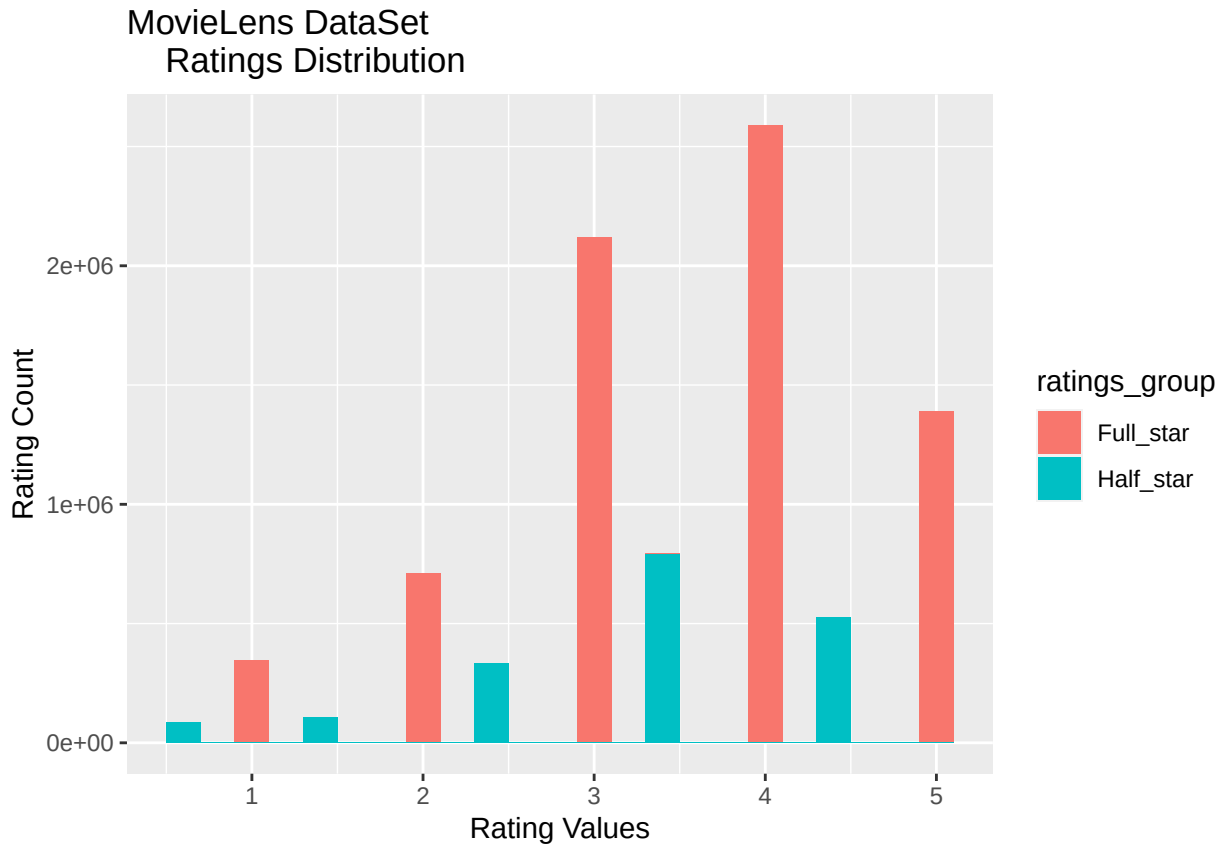
Model Creation: Developing a promising model by accounting for identified data biases.

In this section, the biases that we could identify within the dataset will be showcased and explained. Other avenues for potential biases were explored but this section will just document the biases we were able to

confirm and are able to account for later on through the development of our predictive model. ### Identified Bias I: Whole Star vs Half Star Ratings

```
ratings_group <-  ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                edx$rating == 4 | edx$rating == 5) , "Full_star", "Half_star")
df_ratings <- data.frame(edx$rating, ratings_group)
```

```
plot(ggplot(df_ratings, aes(x= edx.rating, fill = ratings_group))
     + geom_histogram(binwidth = 0.2) + ggtitle("MovieLens DataSet
    Ratings Distribution") + labs(x="Rating Values", y="Rating Count"))
```



Above, we have created a histogram that shows the ratings distribution, grouped by half star and full star ratings.This histogram above gives us some pertinent takeaways regarding the make-up of our dataset, specifically the ratings column.

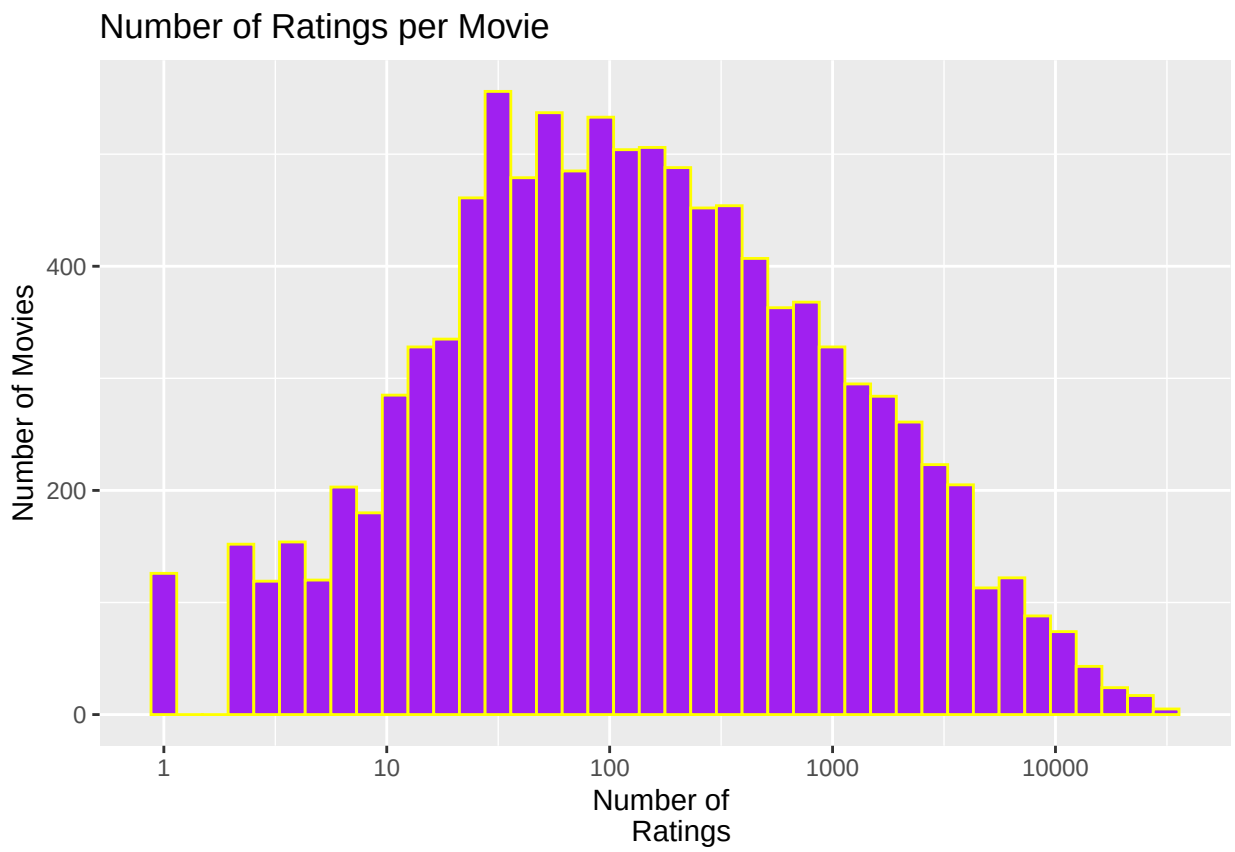
1. There are significantly more full star ratings than half star ratings.
2. There is not a single 0 rating in our dataset.
3. Half star ratings increase in count from 0.5 to 3.5, then drop off for the 4.5 count.
4. Full star ratings increase in count from 0 to 4, then drop off for the 5 count.

Of these takeaways, the primary takeaway is the bias towards full star ratings in our training set. When building out a model to predict ratings in our validation test set, we will need to account for this bias to make our predictions more accurate and improve reliability of the modeling techniques.

Identified Bias II: Movie Ratings Count

Much like we did in the previous section. We need to continue to identify and report upon biases in the dataset.

```
plot(edx %>% count(movieId) %>% ggplot(aes(n)) + geom_histogram(bins = 40,
    color = "yellow", fill = "purple") + scale_x_log10() + labs(x="Number of
    Ratings", y="Number of Movies") + ggtitle("Number of Ratings per Movie"))
```

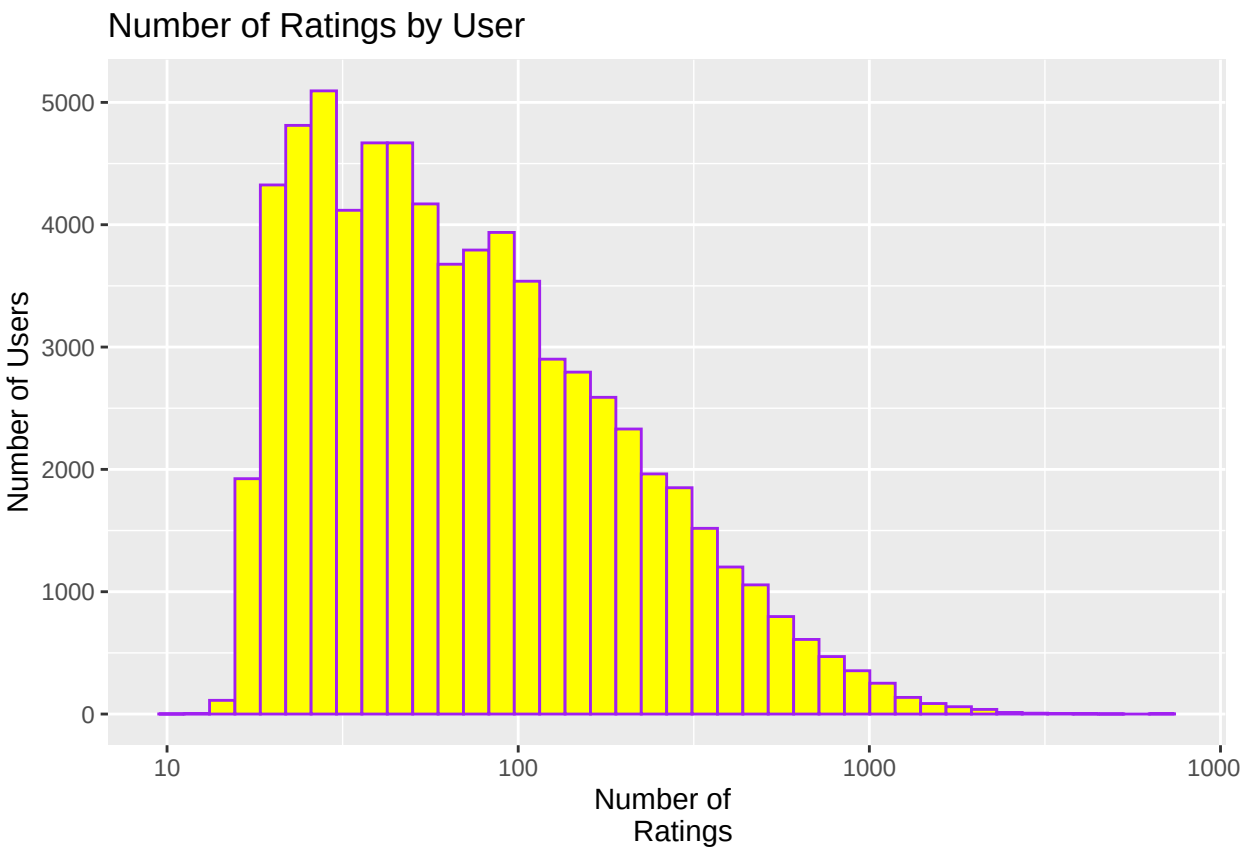


This histogram shows us that there is a bias regarding how many movies have ratings, including a significant number of movies that have been rated just one time. As such, it will make it very difficult to predict the movie ratings for these movies that have been not rated a large number of times which will affect our accuracy. Observing this bias, a model will need to be developed to account for this bias. Regularization will further aid us in achieving this goal.

Identified Bias III: Distribution of Ratings by Users

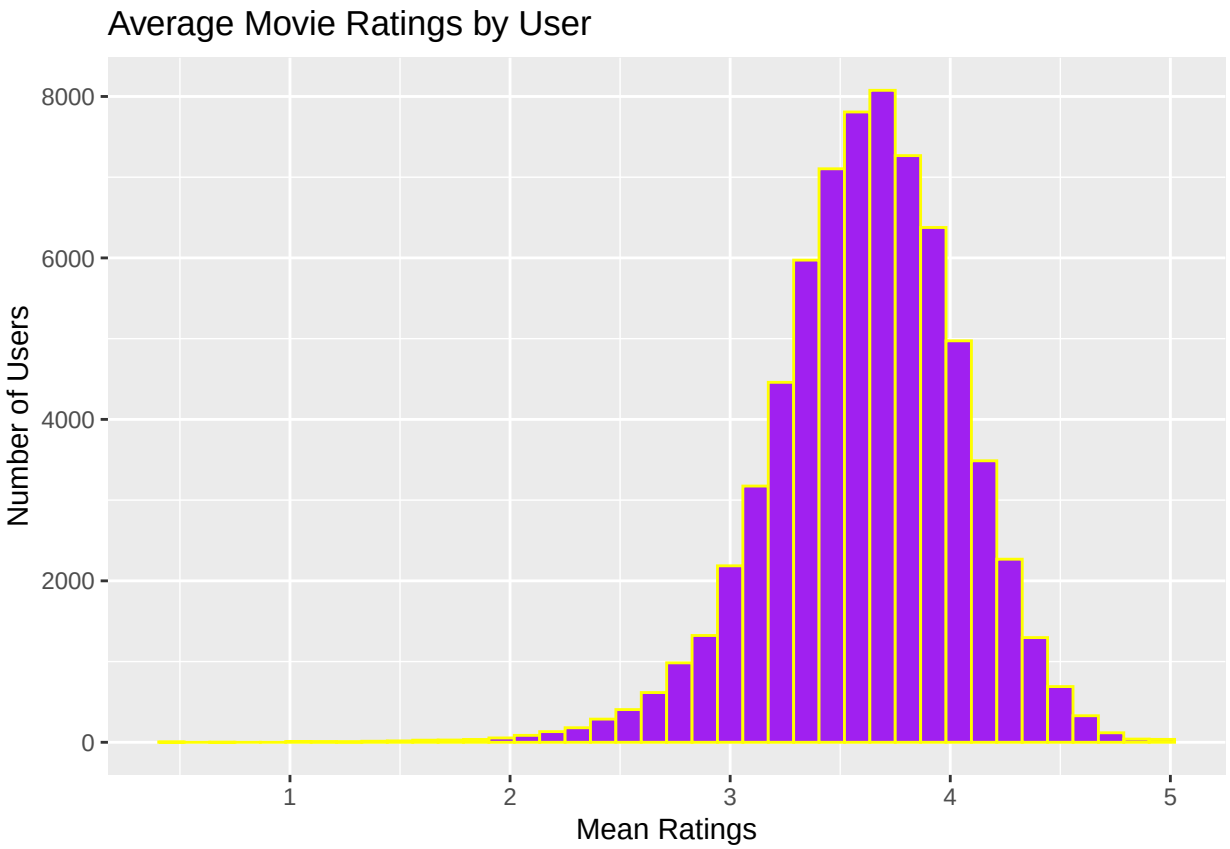
As demonstrated before, we continue with our process of identifying bias / problematic areas in the dataset.

```
plot(edx %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(bins = 40,
    color = "purple", fill = "yellow") + scale_x_log10() + labs(x="Number of
    Ratings", y="Number of Users") + ggtitle("Number of Ratings by User"))
```



This histogram shows most users rate between 50 - 100 movies. So while we have a penalty term for our ratings score bias, we will (through model development) have one for the user term as well. The distribution isn't the only useful part to analyze when looking into users. We can also analyze the distribution of ratings by user. Users who give ratings on the outliers will be more difficult to predict so if that is the case, we will need to account for that. We explore that as follows:

```
plot(edx %>% group_by(userId) %>% summarise(mean_u = mean(rating)) %>%
  ggplot(aes(mean_u)) + geom_histogram(bins = 40, color = "yellow", fill =
  "purple") + labs(x="Mean Ratings", y="Number of Users") +
  ggtitle("Average Movie Ratings by User"))
```



The mean for edx$rating in our dataset is 3.512 and this histogram shows us how many ratings we are getting according to the number of users. We can see that there is an influence here from a high number of users giving certain ratings. Since we have some users who rate a large number of movies, any biases they have (being overly positive, overly negative, genre distastes etc) will be reflected here and make it harder for us to predict ratings in the future.

Model Development & Explanation

Room-mean-square-error(RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. RMSE is simply interpreted as; the lower the number, the better.For this project, we want a RMSE of 0.8649 or less in order to get the maximum number of points available for that section.

The first part of our model development is to define our RMSE function. This will be completed as follows:

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

For this project, we will develop 5 distinct models. These will include our foundation model (Model A) and we will extend off this Model A to achieve results that are better than the results achieved from Model A. These 5 models are broken down and explained as follows:

1. Model A (Average Movie Rating Model)
2. Model B (Median Movie Rating Model)
3. Model C (Adjusted Movie Effect Model)

4. Model D (Adjusted Movie & User Effect Model)
5. Model E (Regularized Movie & User Effect Model)

In the end of this report, we will showcase all the results of the 5 models (the corresponding RMSEs) for comparison purposes.

Model A: Average Movie Rating Model Our first model is among the most basic possible as it predicts the same rating for all movies. We obtain the mean of our rating (previously discussed above) the following way:

```
mu <- print(mean(edx$rating))
```

```
## [1] 3.512465
```

Using this value, 3.51427, we can run our first model (acting as a foundation model) and compute the RMSE.

```
model_a <- print(RMSE(validation$rating, mu))
```

```
## [1] 1.061202
```

This model gives us our first RMSE which is 1.0612 and substantially above our goal RMSE. As such, we can't stop here and need to develop this model further by trying new techniques and accounting for various discovered data biases. #### Model B: Median Movie Rating Model We now look to create our second model;

```
med_u <- print(median(edx$rating))
```

```
## [1] 4
```

Using this value, 4.0, we can run our second model and compare the RMSE with that of our first model.

```
model_b <- print(RMSE(validation$rating, med_u))
```

```
## [1] 1.168016
```

Some key takeaways from Model B:

1. This model gives us our second RMSE which is 1.16802.
2. This RMSE is 10% worse than our first RMSE.

The RMSE differences between a median and mean model were explored to understand/confirm which model to expand upon with extended analyses. This quick exercise confirms that we need to use our first RMSE (using the mean) to develop Model C through Model E.

Model C: Adjusted Movie Effect Model In the earlier section, we identified a biases associated with the fact that some movies will simply be inherently rated higher than other movies. As such, we can add the ability to account for the 'effect of the movie' to create a more involved model, Model C. This model will be an extension of Model A but accounts for the effects of the movie.

```
moviebias_avg <- edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
pred_ratings_bi <- mu + validation %>% left_join(moviebias_avg,
  by='movieId') %>% .$b_i
model_c <- print(RMSE(validation$rating, pred_ratings_bi))
```

```
## [1] 0.9439087
```

Model C produces our lowest RMSE to date which is 0.94391. This represents an approximate 11% improvement over Model A which was previously our top model RMSE prior to integrating Model C. This model represents a significant improvement but further refinement is required to help reach the desired RMSE threshold.

Model D: Adjusted Movie & User Effect Model In the earlier section, we identified a bias associated with the users in our dataset. Some humans are inherently more positive and more negative than others. Since we have a situation where some users rated a high percentage of total movies, this select group of users who rated lots of movies are having an effect on the data and thus our ability to predict movie rating in the validation set. Since we saw that accounting for the movie bias substantially reduced our RMSE, by accounting for the movie and user biases should have an even more significant effect on reducing our RMSE. In a similar fashion to how we developed the model extension for movie bias, we do the same to create a model that accounts for movie and user biases.

```
movieuserbias_avg <- edx %>% left_join(moviebias_avg, by='movieId') %>%
  group_by(userId) %>% summarize(b_u = mean(rating - mu - b_i))
pred_ratings_bu <- validation %>% left_join(moviebias_avg, by='movieId') %>%
  left_join(movieuserbias_avg, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>% .$pred
model_d <- print(RMSE(validation$rating,pred_ratings_bu))
```

```
## [1] 0.8653488
```

Model D produces our lowest RMSE to date which is 0.86535.This represents an approximate 8.3% decrease in RMSE when compared to Model C and 18.5% decrease when compared to Model A. This model represents a significant improvement but is still slightly higher than our desired RMSE threshold so further work is needed. If we can add an extension to this Model D that reduces our RMSE by 0.05% or greater, we will have achieved our goal.

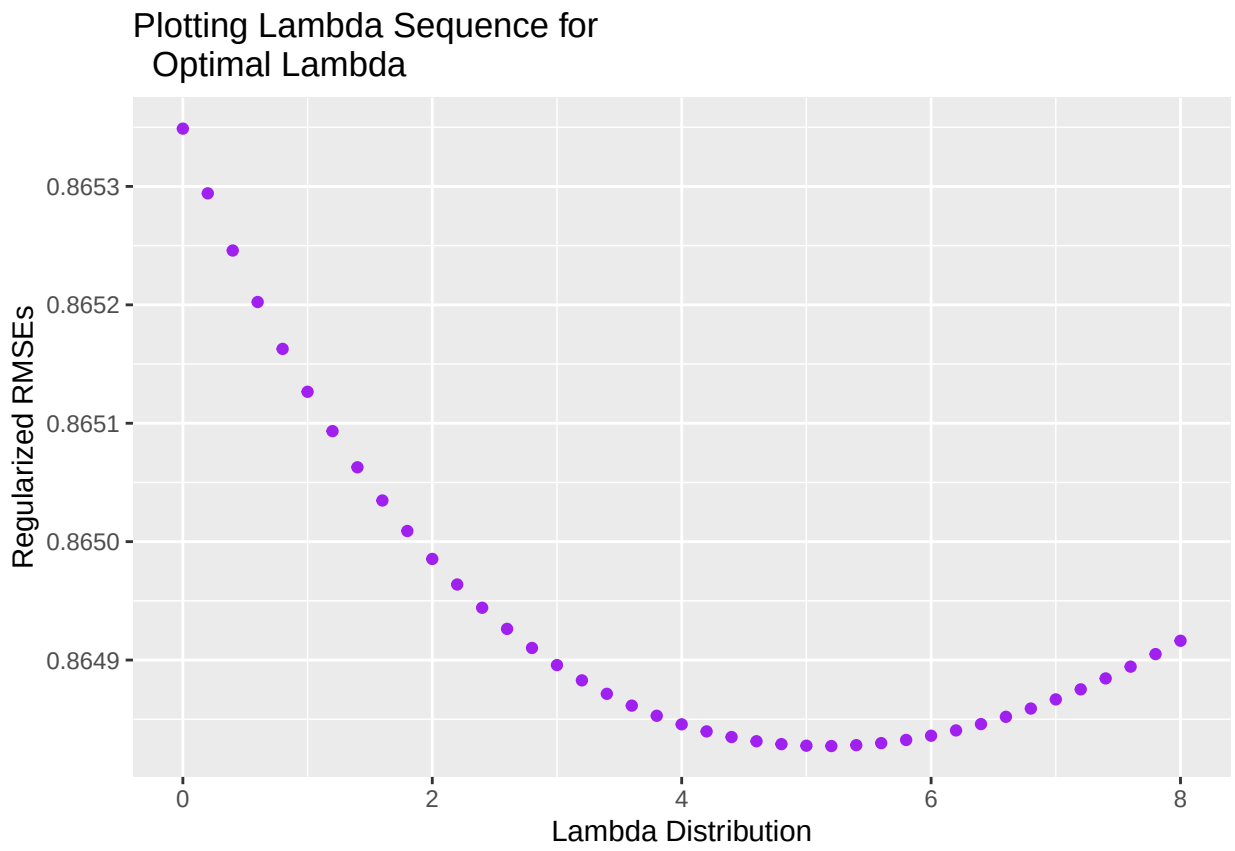
Model E: Regularized Movie & User Effect Model A technique that we went over through the course material will help us lower our RMSE and that technique is known as Regularization. Regularization adds a penalty of the different parameters of the model to reduce the model's freedom. Regularization helps us to add additional information to prevent overfitting and improve model accuracy by punishes values that exist towards the 'tails' (outlier ends) of the dataset values distribution. Regularization is one of the more important prerequisites to improving the accuracy and reliability of convergence and is a very effective method to apply here for our current problem. In regularization, we have to determine what is the optimal 'regularization' effect to inflict upon the model. This 'effect' or 'weighting' in regularization is normally defined as the lambda perameter and can often be referred to as the 'tuning parameter'. The cross validation process helps us determine our optimal tuning parameter, lambda.

That two-step process is developed as follows: 1. Determining the optimal value for the tuning parameter, lambda. This is done using cross validation. 2. Apply that lambda to the regularization equation on Model D in order to determine the RMSE for our regularized model, denoted as Model E.

```
lambdas_seq <- seq(0, 8, 0.2)

reg_rmses <- sapply(lambdas_seq, function(lambda){

moviebias_reg <- edx %>% group_by(movieId) %>%
  summarise(n=n(), bi_reg = sum(rating - mu) / (lambda + n))

movieuserbias_reg <- edx %>% group_by(userId) %>% left_join(moviebias_avg,
  by="movieId") %>% summarise(n=n(), bu_reg = sum(rating - mu - b_i) / (lambda + n))

bi_reg <- validation %>% left_join(moviebias_reg, by="movieId") %>% .$bi_reg

bu_reg <- validation %>% left_join(movieuserbias_reg, by="userId") %>% .$bu_reg

pred_reg <- mu + bi_reg + bu_reg

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

We can plot our lambda distribution if we want to see how it looks.

```
plot_lambdas <- print(data.frame(lambdas_seq, reg_rmses) %>%
  ggplot(aes(lambdas_seq, reg_rmses)) + ggtitle("Plotting Lambda Sequence for
  Optimal Lambda") + labs(x="Lambda Distribution", y="Regularized RMSEs") +
  geom_point(color='purple'))
```



Our optimal number for lambda is computed as follows:

```
opt_lambda <- print(lambdas_seq[which.min(reg_rmses)])
```

```
## [1] 5.2
```

With our optimal lambda determined (5.2), we can create Model E and determine how the corresponding RMSE relates to Model D (our lowest RMSE to date)

```
moviebias_reg <- edx %>% group_by(movieId) %>%
  summarise(n=n(), bi_reg = sum(rating - mu) / (opt_lambda + n))
movieuserbias_reg <- edx %>% group_by(userId) %>%
  left_join(moviebias_avg, by="movieId") %>%
  summarise(n=n(), bu_reg = sum(rating - mu - b_i) / (opt_lambda + n))
bi_reg <- validation %>% left_join(moviebias_reg, by="movieId") %>% .$bi_reg
bu_reg <- validation %>% left_join(movieuserbias_reg, by="userId") %>% .$bu_reg
pred_reg <- mu + bi_reg + bu_reg
model_e <- print(RMSE(validation$rating,pred_reg))
```

```
## [1] 0.8648274
```

After much model development, starting with Model A, we now have an RMSE reported under our required threshold of 0.8649. Success! ### Reported Results: In the Form of an RMSE Table

```
RMSE_Results_Table <- print(data.frame(Letter = c("A","B","C","D","E"),
  Model_Name = c("Average Movie Rating","Median Movie Rating","Adjusted Movie
  Effect","Adjusted Movie & User Effect","Regularized Movie & User Effect"),
  Model_RMSE = c(model_a, model_b, model_c, model_d, model_e),
  RMSE_Success = c("No", "No", "No", "No", "Yes")))
```

```
##   Letter                      Model_Name Model_RMSE RMSE_Success
## 1      A            Average Movie Rating  1.0612018           No
## 2      B             Median Movie Rating  1.1680160           No
## 3      C       Adjusted Movie \n  Effect  0.9439087           No
## 4      D    Adjusted Movie & User Effect  0.8653488           No
## 5      E Regularized Movie & User Effect  0.8648274          Yes
```

Wrap-Up: Conclusions & Project Extensions

From the table reported above (entitled RMSE Results Table), we can see how the RMSE values differ for our 5 modeling attempts. By accounting for movie and user biases and initiating regularization on our modeling, we are able to achieve a RMSE below the required level of 0.8649. Through the duration of this project, we successfully; 1. Uploaded a large dataset into R Studio 2. Cleaned that dataset to specifically meet our needs. 3. Use plots and data tools to identify biases/areas of focus for our project when developing a predictive model. 4. Developed several predictive models to achieve our desired RMSE score. 5. Explained the findings and reported them in a neat, concise table for the reader. In terms of an extension to this project in the future, there are other, more advanced machine learning models that we can experiment with to improve the RMSE. These techniques would be aimed at specifically accounting for genre bias since current limitations with memory (RAM) could not initially be overcome despite some exhaustive efforts. Through the visual data exploration stages, we were able to identify a potential genre bias but initial trials could not develop a proper model to confirm nor account for this added layer. At this stage, that would be the key area of added exploration if this project was to be extended.