Data Science Capstone Movielens Murray Levitt

```r
# --------------------------------------------------------------------------------
# Data Science Capstone : Movielens Assignment
# Student: Murray Levitt
# Date: January 7, 2021
# --------------------------------------------------------------------------------
#
# ---------------
# 1. INTRODUCTION
# ---------------
#
# The objective of this project is to develop a movie recommendation system
# using the MovieLens dataset. A machine learning algorithm will use the
# inputs in one subset to predict movie ratings in the validation set.
# RMSE will be used to evaluate how close the predictions are to the
# true values in the validation set.
#
# -----------------------
# 2. METHODS AND ANALYSIS
# -----------------------
#
# -----------------------
# 2.1 Create Train and Validation Sets
# -----------------------
#
# ##########################################################
# Create edx set, validation set (final hold-out test set)
# ##########################################################

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

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

```
## -- Attaching packages ------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

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

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

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

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

```
##
```

```
## Attaching package: 'caret'

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

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

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

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

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

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

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

```
## Loading required package: dslabs
```

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

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(dslabs)
library(lubridate)

# 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)
```

```r
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
```

```r
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")
```

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

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

```r
# -----------------------
# 2.2 A bit of data wrangling for later model building. Extract
# the movie release date (year) from the title column into a separate column.
# -----------------------

# Extract movie release year from title field into separate column
pattern <- "\\([1-2][0-2,9][0-9][0-9]\\)"
pattern2 <- "[1-2][0-2,9][0-9][0-9]"

# edx set
edx <- edx %>% mutate(temp = str_extract(title, pattern)) %>%
   mutate(rlse_year = str_extract(temp, pattern2))
edx <- select(edx,-temp)

# repeat for validation set
validation <- validation %>% mutate(temp = str_extract(title, pattern)) %>%
   mutate(rlse_year = str_extract(temp, pattern2))
```

```r
validation <- select(validation,-temp)

rm(pattern, pattern2)
```

2.3 Create Additional Test and Training Sets for Model Development (keep separate from Validation set)

```r
# -----------------------
# 2.3 Create Additional Test and Training Sets for Model Development
# (keep separate from Validation set)
# -----------------------
#
# Test set will be 20% of edx data
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- test_set %>%
     semi_join(train_set, by = "movieId") %>%
     semi_join(train_set, by = "userId")
```

```r
# -----------------------
# 2.4 Data Exploration
# -----------------------
#
# -----------------------
# 2.4.1 Genres
# -----------------------
#
# First step was to count the number of distinct combinations of genres.
#
# Count the distinct genres
n_distinct(train_set$genres)
```

```
## [1] 797
```

```r
# 797 genres combos are too many to plot. Next, look at the top and bottom 25 genres
# in terms of mean ratings to see if there is high variability.
```

```r
# Look at the top and bottom 25 genres in terms of mean ratings to
# see if there is high variability.
genres_summary <- train_set %>%
   group_by(genres) %>%
   summarise(count = n(), mean_rating = mean(rating))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
genres_summary %>% top_n(25, mean_rating)
```

```
## # A tibble: 25 x 3
##    genres                                             count mean_rating
##    <chr>                                              <int>       <dbl>
##  1 Action|Adventure|Animation|Comedy|Sci-Fi               2        4.5
##  2 Action|Adventure|Comedy|Fantasy|Romance            11800        4.20
##  3 Action|Crime|Drama|Film-Noir|Mystery                 899        4.11
```

```
##   4 Action|Crime|Drama|IMAX                              1900         4.31
##   5 Action|Drama|Thriller|War                             377         4.10
##   6 Adventure|Animation|Children|Comedy|Romance|Sci-Fi  1010         4.12
##   7 Adventure|Animation|Children|Comedy|Sci-Fi          2838         4.14
##   8 Adventure|Comedy|Romance|War                        4159         4.12
##   9 Adventure|Documentary                                643         4.11
##  10 Adventure|Drama|Film-Noir|Sci-Fi|Thriller          11178         4.15
## # ... with 15 more rows
```

```
genres_summary %>% top_n(-25, mean_rating)
```

```
## # A tibble: 25 x 3
##    genres                                          count mean_rating
##    <chr>                                           <int>       <dbl>
##  1 Action|Adventure|Children                         654        1.88
##  2 Action|Adventure|Children|Comedy|Fantasy|Sci-Fi  2239        2.06
##  3 Action|Adventure|Children|Comedy|Mystery          234        2.10
##  4 Action|Adventure|Comedy|Fantasy|Sci-Fi|Western   4190        2.27
##  5 Action|Adventure|Drama|Fantasy|Sci-Fi              44        1.86
##  6 Action|Adventure|Fantasy|Thriller                3565        2.20
##  7 Action|Animation|Comedy|Horror                      1        2
##  8 Action|Children                                  3149        2.04
##  9 Action|Children|Comedy                            408        1.93
## 10 Action|Children|Fantasy                          1858        2.27
## # ... with 15 more rows
```

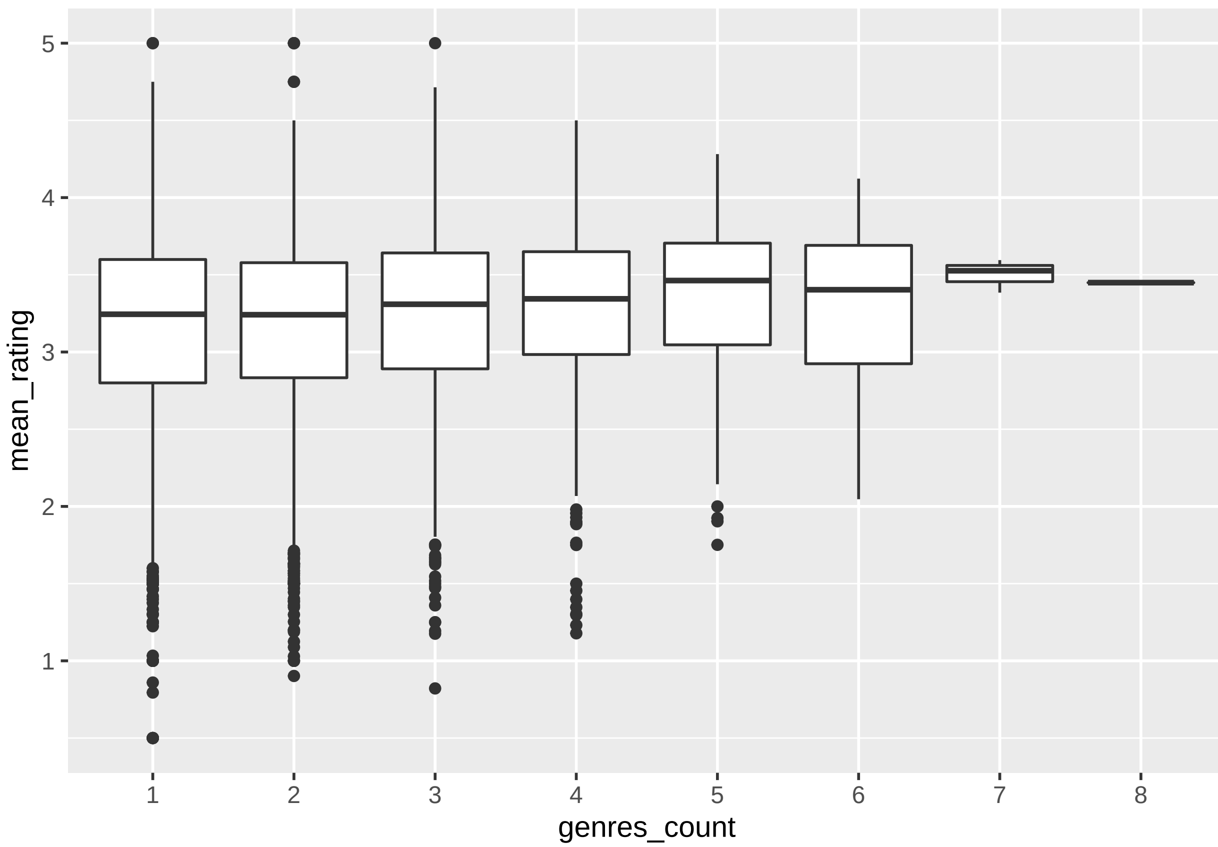
```
#
# Crude analysis but there would appear to be enough variability to factor
# genres into model.Two questions came to mind: Do movies with more genres
# associated have more appeal (i.e higher ratings)?
# Is the variability driven more by individual genres vs. the combo of genres?
#
# To explore this further, we need to parse out the genres column.
#
```

```
library(ggplot2)
# Expand the genres column to rows
# WARNING: This code takes several minutes to run

genres_separated <- edx %>% separate_rows(genres, sep = "\\|")
```

```
#
# First we'll look at the multiple genres question.
#

# group row by movieId, sum number of distinct genres by movie, calculate mean rating
summary_set <- genres_separated %>% group_by(movieId) %>%
   summarize(num_genres = n_distinct(genres),mean_rating = mean(rating))
```

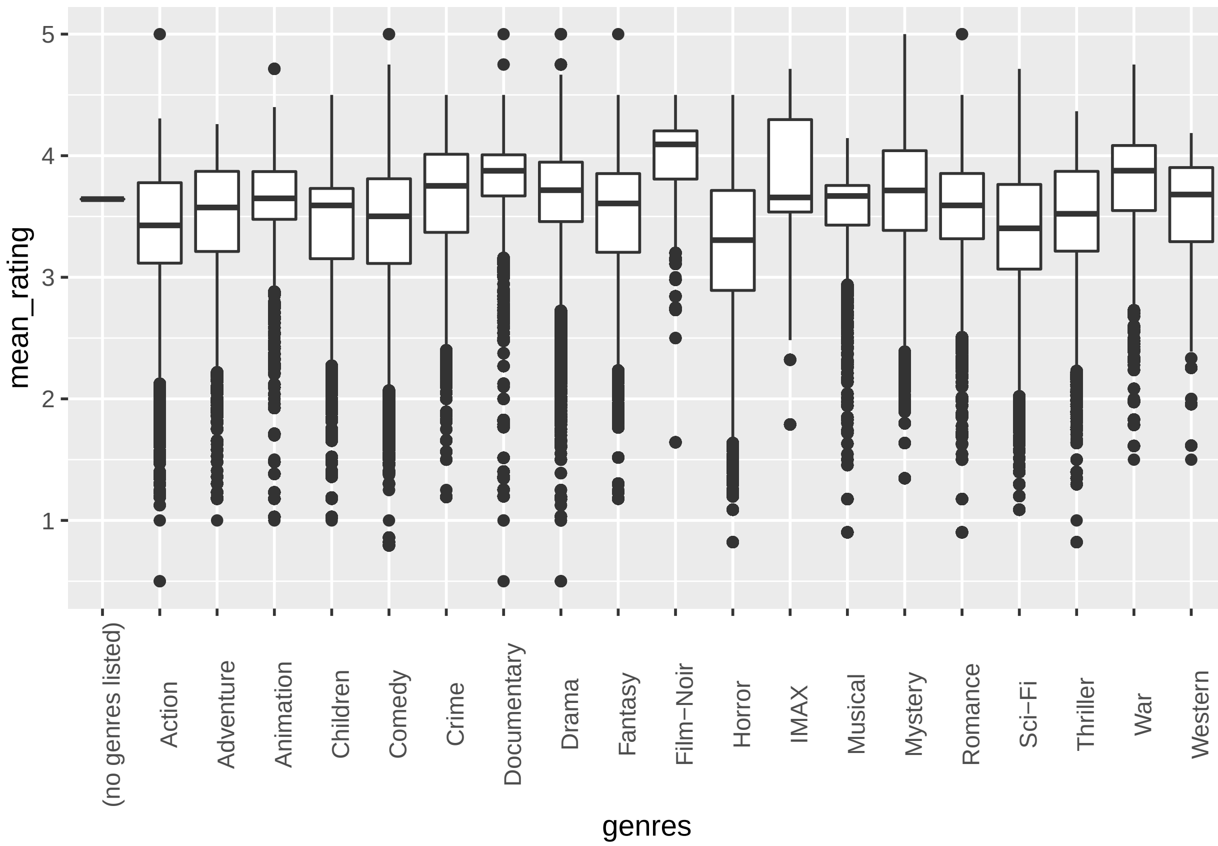
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# convert genres count to character for boxplot
summary_set <- summary_set %>% mutate(genres_count = as.character(num_genres))

# create box plot
summary_set %>% ggplot(aes(genres_count,mean_rating)) +
   geom_boxplot()
```



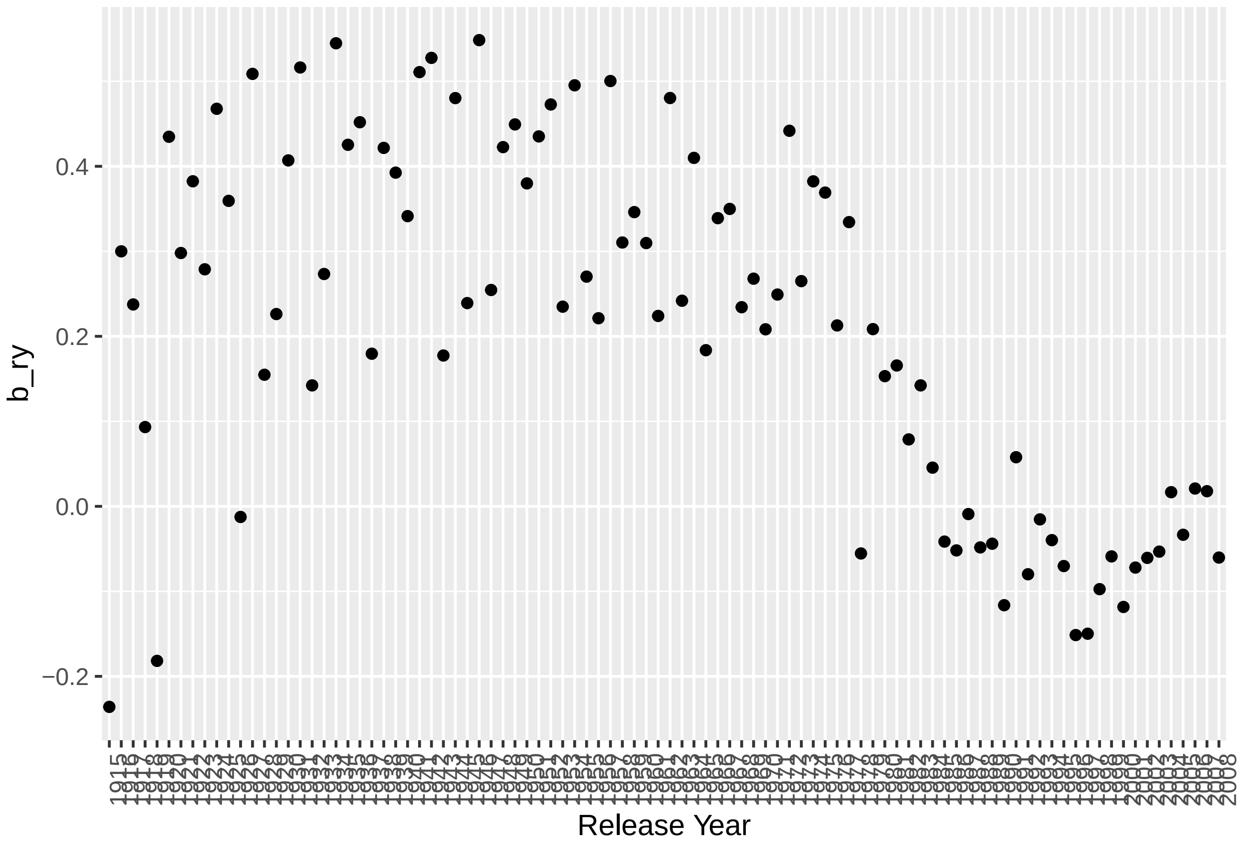

```
#
# Slight increase in rating with number of genres per movie but effect appears small.
# Not worth further consideration.
#
# Next we'll look at the individual genres.
#
# Create boxplot of movie ratings by genre
genres_separated %>% group_by(movieId) %>%
  summarize(genres, mean_rating = mean(rating)) %>%
  ggplot(aes(genres, mean_rating)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90))
```

```
## `summarise()` regrouping output by 'movieId' (override with `.groups` argument)
```



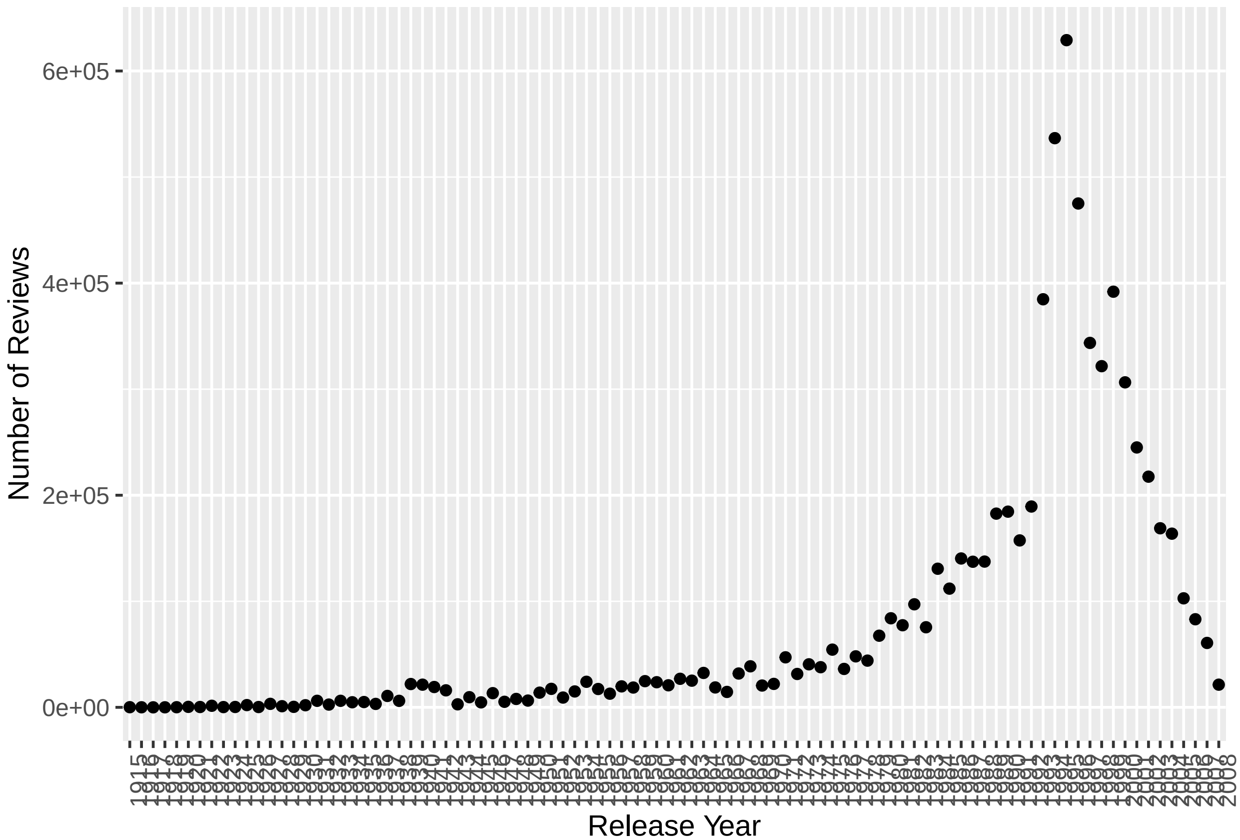

```
# Not a lot variability between most genres. Film-Noir is rated
# most positively; horror most negatively.  But it would appear that
# there is more variability between the combinations of genres.
# So for the purposes of model building, we'll use the combined genres.
#
# -----------------------
# 2.4.2 Release Year
# -----------------------
#
# First we'll look at rating effect (mean difference from average) by release
# year to see if there is pattern.
#

mu <- mean(train_set$rating)
train_set %>%
   group_by(rlse_year) %>%
   summarize(b_ry = mean(rating - mu)) %>%
   ggplot(aes(rlse_year, b_ry)) +
   geom_point() +
   theme(axis.text.x = element_text(angle = 90)) +
   xlab("Release Year") +
   ylab("b_ry")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
# Interestingly, movies release up until the mid to late '70's tended to
# be rated higher than average and then the pattern trended downward rather
# sharply in the early '80's. Movies released since about 1984 tend to have,
# in aggregate, average ratings at or below the overall average.
#
# Since the averages for release dates from 1984-2008 have been at or below
# the overall mean, it would suggest that there have been many more reviews
# for movies released in the those years.
# Next, we'll plot a count of ratings by release year.
#
train_set %>%
   group_by(rlse_year) %>%
   summarize(count = n()) %>%
   ggplot(aes(rlse_year, count)) +
   geom_point() +
#   scale_y_log10() +
   theme(axis.text.x = element_text(angle = 90)) +
   xlab("Release Year") +
   ylab("Number of Reviews")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```r
# This plot confirms the hypothesis.  This exploration suggests release year may
# have a significant effect and will be included in our model.
#
# -----------------------
# 2.5 Establish RMSE function
# -----------------------
#
RMSE <- function(true_ratings, predicted_ratings){
     sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

```r
# -----------------------
# 2.6 Modeling Approach
# -----------------------
#
# As noted in the Data Exploration section, both genres and release year appear
# to have a strong effect on the variability of movie ratings.  So the objective
# of the exercise is to develop and compare a series of models that build upon
# the class examples by adding genres and release year effects.
# The remainder of this section will build the following models:
#
# -Naive ()
# -Movie effect
# -Movie & user effect
# -Movie, user, & genre effect (New)
# -Movie, user, genre & release year effect (a.k.a 'Combo' model) (New)
# -Regularization + Combo effect model (New)
```

```r
#
# The first three models are from the class exercises and are provided for
# comparison only.  The last three are new to this exercise.
#
# Note: A few attempts at linear regression models were made using small
# subsets of the training data but were generally unsuccessful due to the size
# of the datasets involved and memory issues.
#
# -----------------------
# 2.6.1 'Just the Average' Model (For Comparison Only).
# -----------------------
#
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.512482
```

```r
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse
```

```
## [1] 1.059904
```

```r
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
# -----------------------
# 2.6.2 Movie Effect Model (For Comparison Only)
# -----------------------
movie_avgs <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- mu + test_set %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i
model_m_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_m_rmse ))
```

```r
# -----------------------
# 2.6.3 Movie + User Effects Model (For Comparison Only)
# -----------------------
user_avgs <- train_set %>%
   left_join(movie_avgs, by='movieId') %>%
   group_by(userId) %>%
   summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- test_set %>%
   left_join(movie_avgs, by='movieId') %>%
```

```r
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_u_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effect Model",
                                     RMSE = model_u_rmse ))
```

```r
# -----------------------
# 2.6.4  Movie + User + Genres Effect Model (New)
# -----------------------
genres_avgs <- train_set %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     group_by(genres) %>%
     summarize(b_g = mean(rating - mu - b_i - b_u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- test_set %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by="userId") %>%
     left_join(genres_avgs, by='genres') %>%
     mutate(pred = mu + b_i + b_u + b_g) %>%
     .$pred
model_g_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Genres Effect Model",
                                     RMSE = model_g_rmse ))
```

```r
# -----------------------
# 2.6.5 Movie + User + Genres + Release Year (a.k.a 'Combo') Effect Model
# -----------------------
movie_avgs <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
user_avgs <- train_set %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
genres_avgs <- train_set %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     group_by(genres) %>%
     summarize(b_g = mean(rating - mu - b_i - b_u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
rlse_year_avgs <- train_set %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
```

```r
    left_join(genres_avgs, by='genres') %>%
    group_by(rlse_year) %>%
    summarize(b_ry = mean(rating - mu - b_i - b_u - b_g))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(genres_avgs, by='genres') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(rlse_year_avgs, by='rlse_year') %>%
    mutate(pred = mu + b_i + b_g + b_u + b_ry) %>%
    .$pred
model_combo_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Combo Effect Model",
                                     RMSE = model_combo_rmse ))
```

```r
# Comparing the models before regularization, we see that the 'Combo' model,
# Movie + User + Genres + Release Year, has the lowest RMSE.

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effect Model	0.8659320
Movie + User + Genres Effect Model	0.8655941
Combo Effect Model	0.8654189

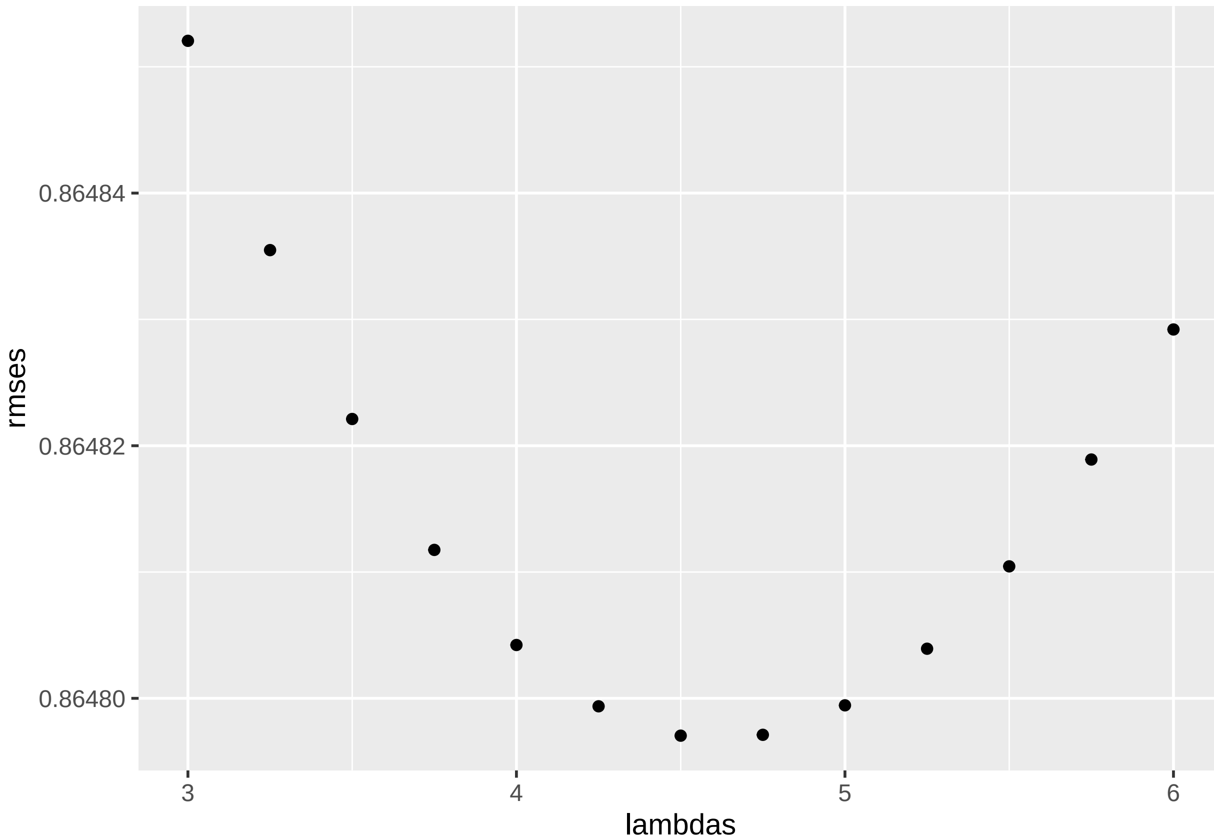
```r
# -----------------------
# 2.6.6 Regularizing the Combo Model
# -----------------------
# Test to find out lambda that optimizes RMSE for the 'Combo' Model
# WARNING: This code takes several minutes to run
lambdas <- seq(3, 6, 0.25)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  b_ry <- train_set %>%
    left_join(b_i, by="movieId") %>%
```

```
      left_join(b_u, by="userId") %>%
      left_join(b_g, b="genres") %>%
      group_by(rlse_year) %>%
      summarize(b_ry = sum(rating - b_i - b_u - b_g - mu)/(n()+l))
   predicted_ratings <-
      test_set %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_g, by = "genres") %>%
      left_join(b_ry, by = "rlse_year") %>%
      mutate(pred = mu + b_i + b_u + b_g + b_ry) %>%
      pull(pred)
   return(RMSE(predicted_ratings, test_set$rating))
})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(lambdas, rmses)
```




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

```
## [1] 4.5
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularization + Combo Effect Model",
                                     RMSE = min(rmses)))
```

```
# Lambda of 4.5 optimizes the RMSE of the 'Combo' model.
#
# Results of all the models using the training and test sets:
#
```

```r
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effect Model	0.8659320
Movie + User + Genres Effect Model	0.8655941
Combo Effect Model	0.8654189
Regularization + Combo Effect Model	0.8647970

```r
# As we see from the table, adding each effect improves the RMSE.
# Regularizing the 'Combo' effect model - movie + user + genres + release year,
# produced the best RMSE of 0.8647970.
```

```r
# -----------------
# 3. RESULTS
# -----------------
#
# To generate the final results, the Regularization + Combo Effect Model will
# be run against the validation set.
#
l <- lambda
mu_hat <- mean(edx$rating)
b_i <- edx %>%
   group_by(movieId) %>%
   summarize(b_i = sum(rating - mu_hat)/(n()+l))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_u <- edx %>%
   left_join(b_i, by="movieId") %>%
   group_by(userId) %>%
   summarize(b_u = sum(rating - b_i - mu_hat)/(n()+l))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_g <- edx %>%
   left_join(b_i, by="movieId") %>%
   left_join(b_u, by="userId") %>%
   group_by(genres) %>%
   summarize(b_g = sum(rating - b_i - b_u - mu_hat)/(n()+l))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_ry <- edx %>%
   left_join(b_i, by="movieId") %>%
   left_join(b_u, by="userId") %>%
   left_join(b_g, b="genres") %>%
   group_by(rlse_year) %>%
   summarize(b_ry = sum(rating - b_i - b_u - b_g - mu_hat)/(n()+l))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- validation %>%
   left_join(b_i, by = "movieId") %>%
```

```r
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_ry, by = "rlse_year") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_ry) %>%
  pull(pred)

#Run the RMSE
rmse_final <- RMSE(predicted_ratings, validation$rating)

#Print the result
print("Final RMSE using Regularized Combo Model is:")
```

```
## [1] "Final RMSE using Regularized Combo Model is:"
```

```r
print(rmse_final)
```

```
## [1] 0.8642948
```

```r
# --------------------
# 4.  CONCLUSION
# --------------------
#
# Using machine learning techniques, an algorithm was constructed that applied
# regularization to a model that combined movie, user, genre and release date
# effects to predict movie ratings with a root mean squared error (RMSE) of
# 0.8642948 for a lambda of 4.5 against the validation set.
#
# While this is a good value relative to other the other models explored,
# there are many limitations of this approach and opportunities for further study.
# For example, other effects, such as movie review time, were not explored or modeled.
# Moreover, other machine learning techniques, such as matrix factorization, clustering
# and PCS were not considered for this assignment.
```