HarvardX PH125.9x Movie Recommendation Model

Henry Andrew Wong

03/09/2021

Executive Summary

Purpose & Process

The goal of this project is to build a predictive model for user ratings of movies, using the publicly available "MovieLens 10M Dataset" from https://grouplens.org/datasets/movielens/. Data acquisition code and wrangling to arrive at a starting state was provided by edX. Further data wrangling, exploration, visualization, analysis, model building, testing and validation followed.

Model Development

Many ratings prediction models were attempted. Some long-running experiments breached the limits of practicality while others presented too little predictive value to report. Candidate models with viable predictive value, measured by their RMSE against a validation dataset, are reported herein:

1. A built-up linear model, incrementally adding bias effects.
2. A standalone regression tree model.
3. An ensemble model using linear predictions as a feature in a regression tree model.

The built-up linear model presented the best results (lowest RMSE).

Results

1. The built-up linear model presented the strongest results, yielding RMSE scores between 0.820 to 0.860 over different parameter variations. Additional time features were derived from the base data to increase predictive power. Feature engineering proved to be more effective than alternative model selection.

2. The standalone regression tree model resulted in RMSE scores that approached, but did not beat the linear model. Adding features did not improve the model.

3. The ensemble approach of combining predictions from the linear model as inputs to a regression tree improved upon the standalone regression trees, but not the linear model that used derived features.

4. Some experiements were conducted with standalone random forest and ensembles with a random forest component. These were ultimately abandoned due to hardward limits.

Methods & Analysis

Data Acquisition

Steps Followed by edX Code

Code provided in the course "HarvardX PH125.9x Data Science: Capstone" (on the edX educational platform) acquires and formats movie ratings data into a starting state. From there, additional columns of data are derived and wrangled for visualization, analysis, model creation, testing and validation.

The script provided in the course does the following:

1. Checks for and installs the following libraries, if necessary: tidyverse, caret, data.table.
2. Invokes the same libraries for use in the current script.
3. Creates a "dl" variable to store the compressed .zip file from the grouplens.org site.
4. Downloads the contents of the file into the "dl" object.
5. Creates a "ratings" variable and fills it with contents from ratings.dat, and adds column names.
6. Creates a "movies" variable and fills it with contents from movies.dat, and adds column names.
7. Overwrites the "movies" variable with a data frame version of the contents.
8. Creates a "movielens" variable by joining "ratings" to "movies" (with "ratings" on the left to keep each row as an individual user's rating of an individual movie). The join is necessary to add the human-readable name to "movielens" variable ("movies" identifies the movie by a numerical ID only).
9. Creates an index using a sampled 0.1 proportion of all records in "movielens."
10. Applies the index to fill a variable "edx" containing the records NOT sampled (to use as a training set), and applies the index of sampled record identifiers to fill a "temp" variable to contain the basis for a validation set (some records removed in the next step).
11. Finalizes the validation set in a variable "validation" by keeping only the records where the "movieId" and "userId" values occur in both "edx" and "temp." Technically, all the rows in "temp" are kept where there is matching value in "edx," without duplicates in "temp."
12. Finalizes the test set "edx" by putting records that were rejected for "validation" back into "edx."
13. Removes intermediary variables to keep the R environment clean.

Data Acquisition Code Provided by edX

```
### Beginning of edX Code ###

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

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

```r
# 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")

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) # if using R 3.5 or earlier, use `set.seed(1)`
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)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)

### End of edX code ###
```

Data Description

Inspection revealed the starting state of the data acquired.

```r
# What type of structure is it?  A list (as opposed to a data frame or tibble).
typeof(edx)
```

```
## [1] "list"
```

```r
# Inspect structure of resulting data.
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Bird of Prey (1996)" "Bad Moon (1996)" "Wings of Desire (Der Himmel Ã¼ber Berlin
##  $ genres   : chr  "Action" "Action|Adventure|Horror" "Comedy|Drama|Fantasy|Romance" "Comedy|Mystery
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# See sample rows.
head(edx, 5)
```

```
##    userId movieId rating timestamp
## 1:      1     122      5 838985046
## 2:      1     185      5 838983525
## 3:      1     231      5 838983392
## 4:      1     292      5 838983421
## 5:      1     316      5 838983392
##                                                 title
## 1:                                Bird of Prey (1996)
## 2:                                    Bad Moon (1996)
## 3: Wings of Desire (Der Himmel Ã¼ber Berlin) (1987)
## 4:                          Arsenic and Old Lace (1944)
## 5:                       Some Kind of Wonderful (1987)
##                          genres
## 1:                       Action
## 2:      Action|Adventure|Horror
## 3: Comedy|Drama|Fantasy|Romance
## 4:      Comedy|Mystery|Thriller
## 5:                 Drama|Romance
```

The resulting data set is technically contained in a variable of type "list" in the R environment. The field names (ie. column names) and contents are as follows:

1. *userId*: A unique identifier for users who provide ratings, expressed as an integer. Because each record (ie. row) in the data set represents one rating of one specific movie provided by one specific users, any individual userID may have multiple records associated with it in the database, but the identifier itself always represents the unique individual user.

2. *movieId*: A unique identifier for a movie, expressed as an integer. The integer in this field corresponds with a specific movie title in the "title" column. A movieId never refers more than one unique title.

3. *rating*: This is the movie rating provided by the specific userId for the specific movieId in the row. It is a numerical value, ranging from 1.0 to 5.0, in steps of 0.5. The possible values are 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5 and 5.0.

4. *timestamp*: This is the time that the movie rating was submitted by the user, expressed as an integer representing seconds, from a specific starting point. It is assumed that this follows the well-known convention that starts the first second of January 1, 1970 in UTC time, beginning one second after the end of December 31, 1969.

5. *title*: The title of the movie, expressed as a character string. This is unique for each movie, but matches with a specific movieId.

6. *genres*: A character string showing all of the genres that a particular movie belongs to. For any given movie, the applicable genres appear to be listed in alphabetical order, separated by a pipe character, "|". This means every movie in the exact same list of genres should contain the exact same string (as opposed to having the same genres, but listed in a separate order).

Data Exploration & Visualization

The required libraries (pre-installed), were invoked.

```
# For data frames and manipulation:
library(dplyr)

# To process strings:
library(stringr)

# For plots and charts:
library(ggplot2)

# Tools to help with quick evaluation of data:
if(!require(tidyverse)) install.packages("cgwtools", repos = "http://cran.us.r-project.org")
library(cgwtools)
```

Rating Frequencies

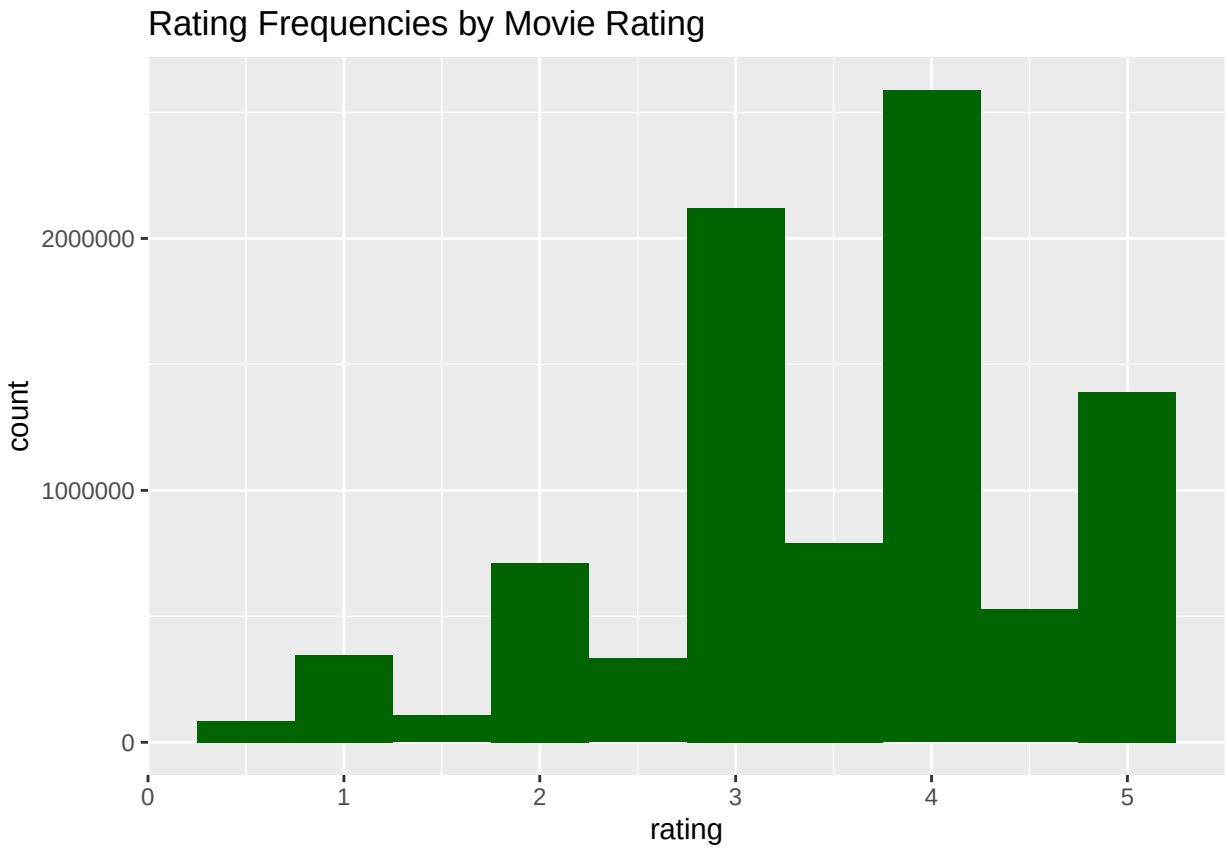
The frequency of ratings was tabled and plotted in a histogram, showing:

1. A user preference for whole number ratings over half-step ratings (ie. there are more 1's than 1.5's, more 2's than 2.5's, etc.).
2. The frequencies of half step ratings of 1.5, 2.5 and 3.5 rise roughly in proportion to the whole number ratings of 1, 2 and 3.
3. The most frequent rating is 4, followed by 3.

```
# Turn off scientific notation.
options(scipen=999)

# Show the frequencies of each rating in a table.
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85420  345935  106379  710998  332783 2121638  792037 2588021  526309 1390541
```

```
# Plot rating frequencies in a histogram. Fit it into 10 bins (1 per rating value).
ggplot(data=edx, aes(x=rating)) +
  geom_histogram(fill="dark green", bins=10) +
  labs(title="Rating Frequencies by Movie Rating")
```

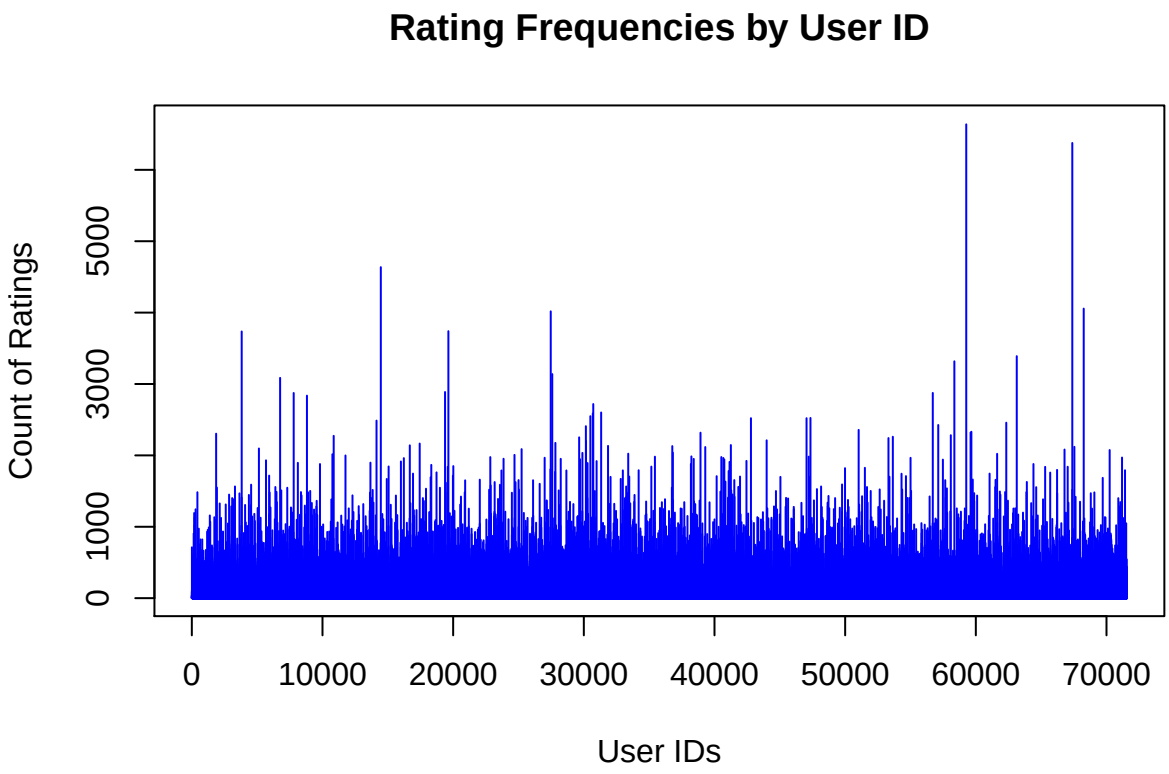


Ratings Grouped by User

First, the number of ratings registered by individual users was examined. Some users submitted many more ratings than others. A few users provided more than 2000 ratings.

Intuitively, enough ratings from an individual user may show central tendency in the user's ratings. If mean ratings between users appear signficantly different, individual user rating means would be a good predictor to include in a model.

```
# Call binit() function from the cgwtools library.
# This creates a table of frequencies.
userIds <- binit(edx$userId)

# Graph it.
plot(userIds$values, userIds$lengths, type="h"
     , col="blue"
     , main="Rating Frequencies by User ID"
     , xlab="User IDs"
     , ylab="Count of Ratings")
```



Next, ratings were grouped by users and filtered to those showing 100+ ratings (to ensure robust averages).

```
# Group ratings by user, and calculate each user's average, min and max ratings.
ratings_by_user <- edx %>% group_by(userId) %>%
  summarize(userRatingsCount = n()
            , userMeanRating = mean(rating)
            , userMinRating = min(rating)
            , userMaxRating=max(rating)) %>%
              filter(userRatingsCount >= 100)

# Show example rows.
ratings_by_user %>% arrange(desc(userRatingsCount))
```

```
## # A tibble: 24,085 x 5
##    userId userRatingsCount userMeanRating userMinRating userMaxRating
##     <int>            <int>          <dbl>         <dbl>         <dbl>
##  1  59269             6637           3.26           0.5             5
##  2  67385             6376           3.20           1               5
##  3  14463             4637           2.41           1               5
##  4  68259             4056           3.56           0.5             5
##  5  27468             4018           3.83           1               5
##  6  19635             3740           3.50           1               5
##  7   3817             3736           3.11           1               5
##  8  63134             3390           3.27           0.5             5
##  9  58357             3318           3.00           0.5             5
## 10  27584             3139           3.00           1               5
```

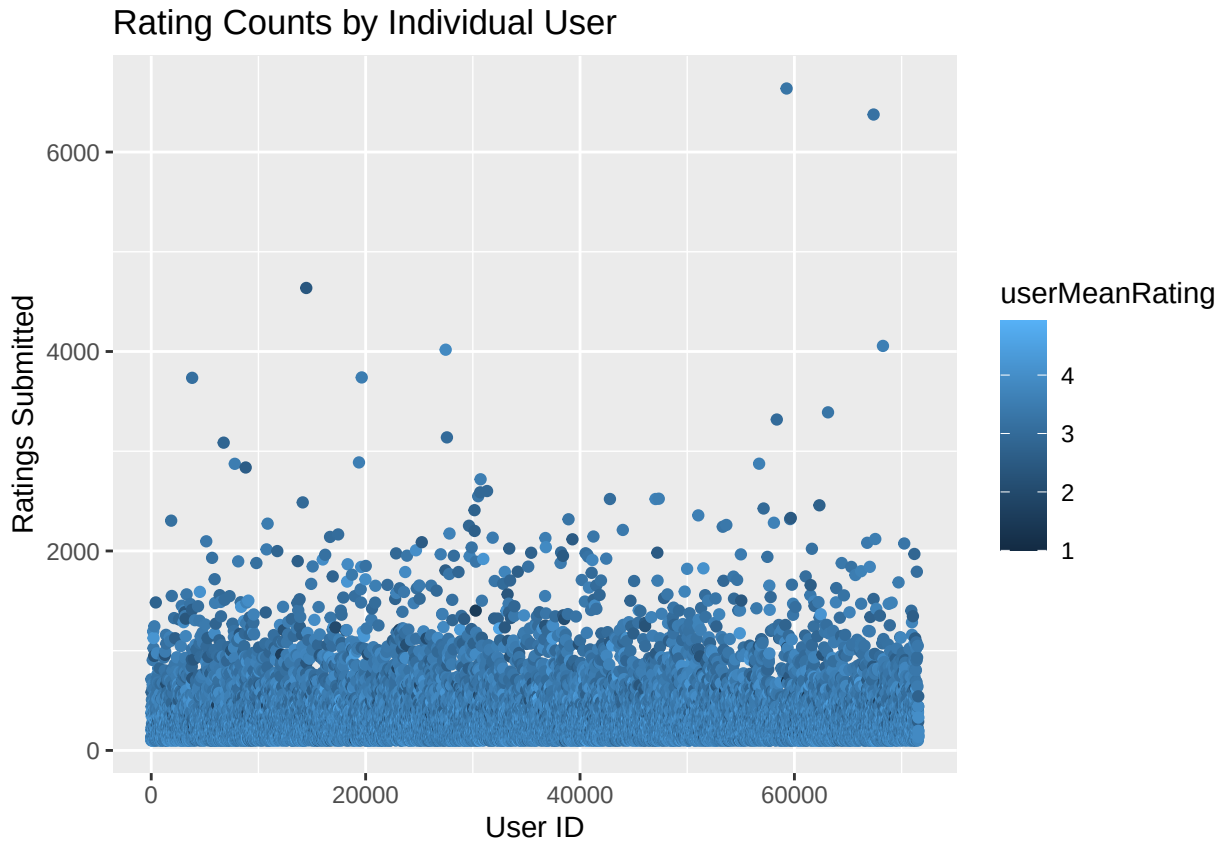
```
## # ... with 24,075 more rows
```

A scatterplot was built to show, for *each user*:

1. The count of ratings submitted by each user (height of the point on the y-axis).
2. The average rating given by each user (color indicator).

The scatterplot indicates different users give different average ratings (at all count levels of ratings submitted). This is a good predictor to include in a model.

```
# Graph rating counts by user.
ratings_by_user %>%
  data.frame() %>%
  ggplot(aes(y=userRatingsCount, x=userId)) +
    geom_point(aes(color=userMeanRating)) +
    labs(title="Rating Counts by Individual User"
    , y="Ratings Submitted"
    , x="User ID")
```



A quick check shows weak negative correlations between the number of ratings from a user and associated mean and minimum ratings.

In general, when a user submits more ratings, their *average* and *maximum* ratings tend to fall slightly. A falling mean comes from diluting existing ratings with lower ones. A falling minimum could result from a user adding a lower minimum, supplanting the previous mimimum.

Submitting more ratings is not correlated with either higher or lower scores.

```r
cor(ratings_by_user$userRatingsCount, ratings_by_user$userMeanRating)
```

```
## [1] -0.2186968
```

```r
cor(ratings_by_user$userRatingsCount, ratings_by_user$userMinRating)
```

```
## [1] -0.2343683
```

```r
cor(ratings_by_user$userRatingsCount, ratings_by_user$userMaxRating)
```

```
## [1] 0.0286438
```

A sorted list of users that submitted a high count of ratings shows two users with more than 6000 ratings, 3 with more than 4000, and more with 3000+ ratings.

The variation between mean ratings given by these users is based on relatively more observations than for other users and shows significant differences between users in just this short sampling.

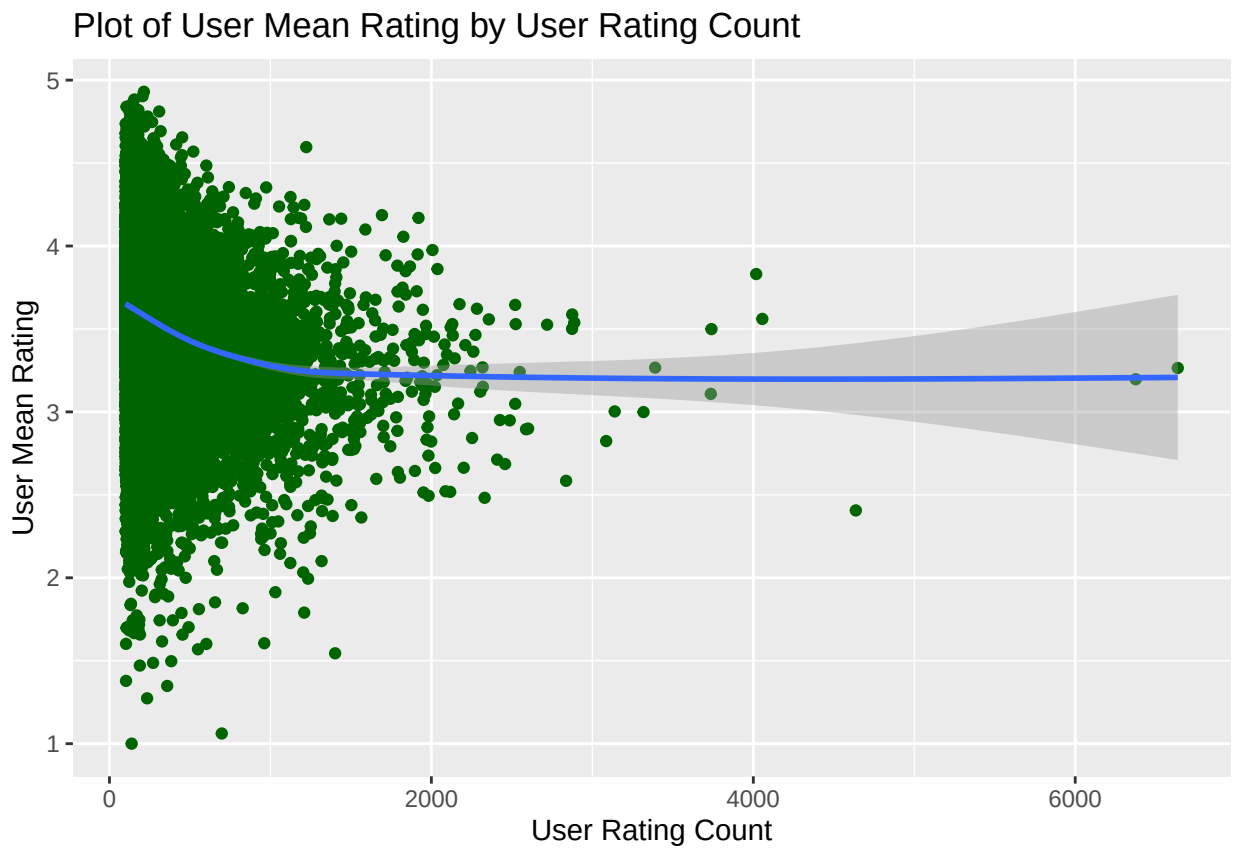
```r
# Show number of ratings by the top 10 most frequent raters.
ratings_by_user %>% arrange(desc(userRatingsCount)) %>% head(.,10)
```

```
## # A tibble: 10 x 5
##    userId userRatingsCount userMeanRating userMinRating userMaxRating
##     <int>            <int>          <dbl>         <dbl>         <dbl>
##  1  59269             6637           3.26           0.5             5
##  2  67385             6376           3.20           1               5
##  3  14463             4637           2.41           1               5
##  4  68259             4056           3.56           0.5             5
##  5  27468             4018           3.83           1               5
##  6  19635             3740           3.50           1               5
##  7   3817             3736           3.11           1               5
##  8  63134             3390           3.27           0.5             5
##  9  58357             3318           3.00           0.5             5
## 10  27584             3139           3.00           1               5
```

For users who registered 100 or more ratings:

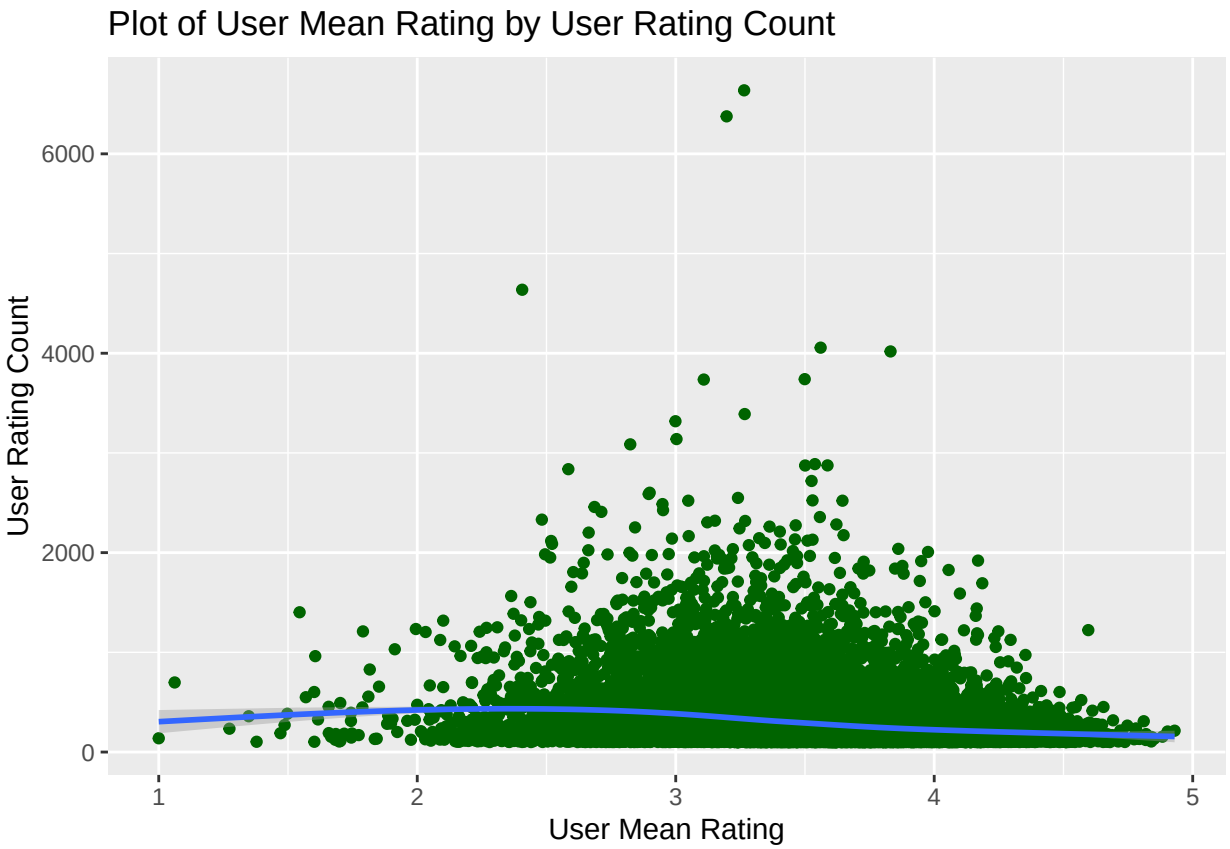
1. The average rating given by users is more varied among users who gave fewer ratings.
2. By inspection, the mean of user mean ratings starts high (left of the graph) among users who gave fewer ratings, then drops as the number of ratings increases. This could be due to an early "halo effect" when a user first experiences rating movies or simply picking movies they already like from a wide selection of unwatched movies, then having to pick from riskier choices as they watch more.
3. As individual users register more ratings, they appear to converge around a steady mean of user mean ratings, roughly 3.2 by inspection, among users registering 1000 or more ratings.
4. It is noted that a few "disgruntled raters" who, although they register few ratings, average very low (in the 1 to 2 range).
5. Point (4) is in contrast to the rating range of 4 to 5, where those registering fewer ratings average high enough to pull the overall average higher than the convergence tendency in (3).

```
# Graph the distribution of mean ratings for users with 100+ ratings.
ratings_by_user %>% arrange(desc(userRatingsCount)) %>%
  data.frame() %>%
  ggplot(aes(x=userRatingsCount, y=userMeanRating)) +
  geom_point(color="dark green") +
  geom_smooth() +
  ggtitle("Plot of User Mean Rating by User Rating Count") +
  labs(x="User Rating Count", y="User Mean Rating")
```



By flipping the axis to show mean ratings given by individual users as the X-axis, a relatively normal distribution appears, starting just before the rating of 2 on the x-axis.

```
# Fip the axis.
ratings_by_user %>% arrange(desc(userRatingsCount)) %>%
  data.frame() %>%
  ggplot(aes(y=userRatingsCount, x=userMeanRating)) +
  geom_point(color="dark green") +
  geom_smooth() +
  ggtitle("Plot of User Mean Rating by User Rating Count") +
  labs(y="User Rating Count",x="User Mean Rating")
```



Next, users will filtered to those with 1000+ ratings. Their averages are even lower. This reinforces using user rating means as a bias in models.

```
# Filter user to only those with 1000 or more ratings.
ratings_by_user1000 <- edx %>% group_by(userId) %>%
  summarize(userRatings = n(), userMeanRating = mean(rating)
  , userMinRating = min(rating), userMaxRating=max(rating)) %>%
  filter(userRatings > 1000)

# Show example data.
head(ratings_by_user1000,5)
```

```
## # A tibble: 5 x 5
##   userId userRatings userMeanRating userMinRating userMaxRating
##    <int>       <int>          <dbl>         <dbl>         <dbl>
## 1    182        1193           3.71           1               5
## 2    212        1025           3.36           0.5             5
## 3    215        1127           4.16           1               5
## 4    289        1245           3.45           0.5             5
## 5    426        1484           2.77           0.5             5
```

Users with 1000+ ratings had a lower mean rating of 3.23 as compared to the overall mean of 3.51. Variation around the mean produced a slightly higher standard deviation for users with 1000+ ratings compared to the overall SD.

This is more evidence that individual user mean ratings are a strong predictor to use in a model.

```r
# Calculate mean and standard deviation for users with 1000 or more ratings.
edx %>% semi_join(ratings_by_user1000, by="userId") %>%
  summarize(mean1000=mean(rating), sd1000=sd(rating)) %>%
  select(mean1000, sd1000)
```

```
##   mean1000   sd1000
## 1  3.23319 1.066251
```

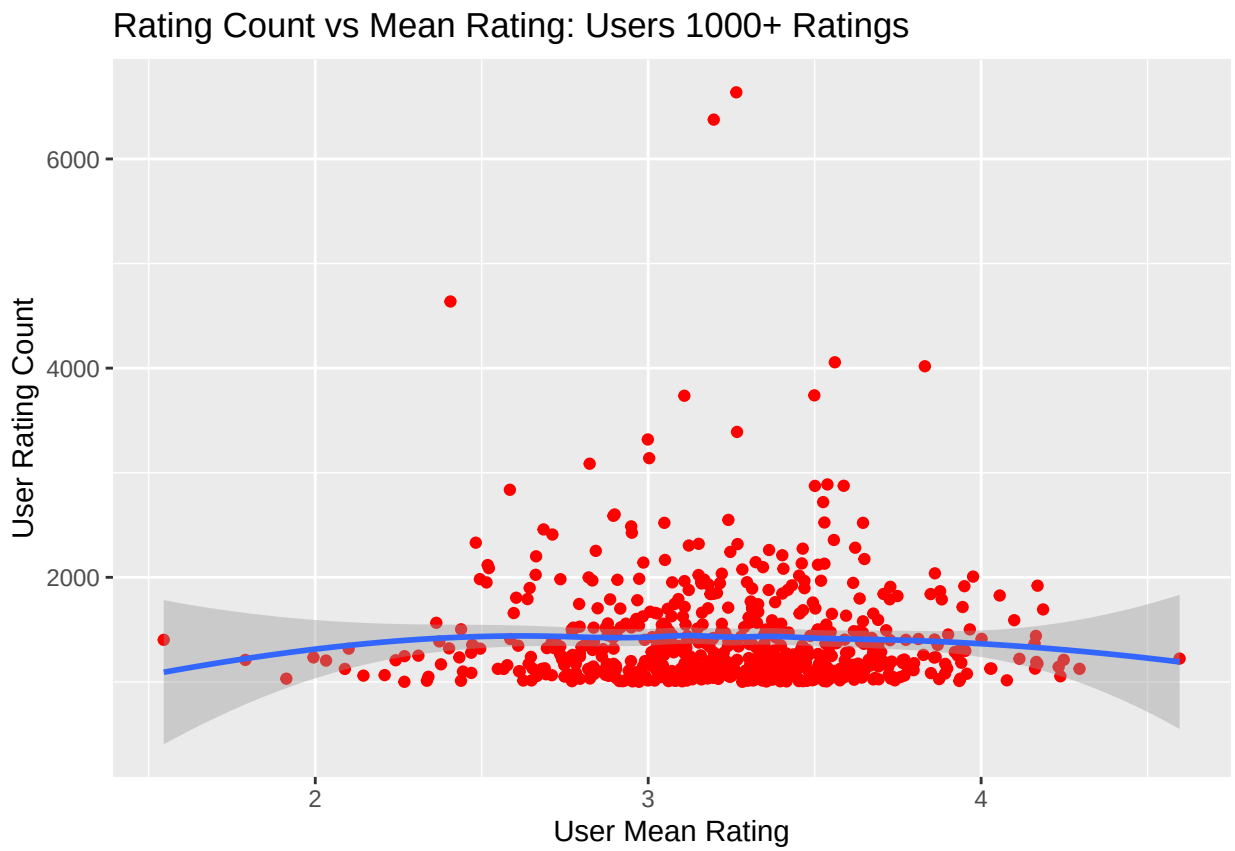
```r
# Compare to users with 100+ ratings.  This includes those with 1000+ ratings.
edx %>% semi_join(ratings_by_user, by="userId") %>%
  summarize(mean100=mean(rating), sd100=sd(rating)) %>%
  select(mean100, sd100)
```

```
##    mean100    sd100
## 1 3.472059 1.055842
```

```r
# Compare to overall mean and SD, without filters.
c(mean(edx$rating), sd(edx$rating))
```

```
## [1] 3.512464 1.060393
```

```r
# Graph filtered set of users who gave at least 1000 ratings.
ratings_by_user1000 %>% arrange(desc(userRatings)) %>%
  data.frame() %>%
  ggplot(aes(y=userRatings, x=userMeanRating)) +
  geom_point(color="red") +
  geom_smooth() +
  ggtitle("Rating Count vs Mean Rating: Users 1000+ Ratings") +
  labs(y="User Rating Count",x="User Mean Rating")
```



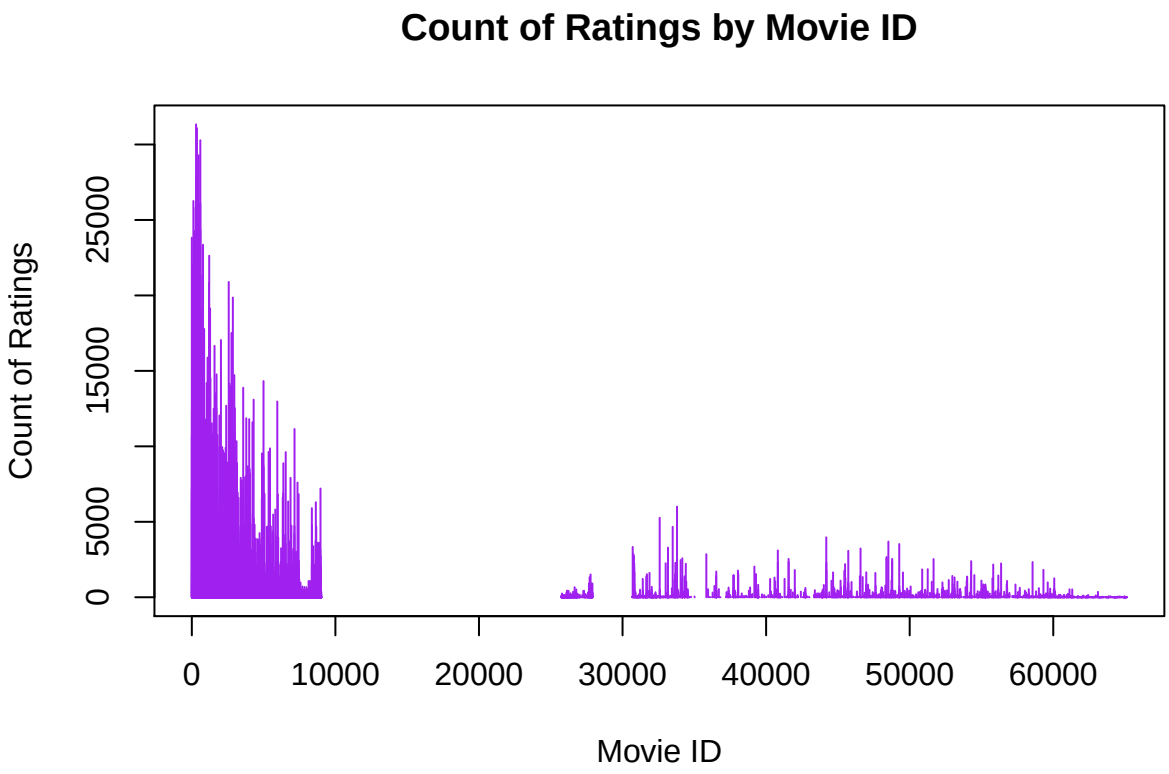
Overall, there was sufficient evidence to include user rating averages as a predictor in models.

Ratings Grouped by Movie

The movie itself is highly likely to be an influential predictor for user ratings. After all, rating movies is fundamentally about scoring movie entertainment value relative to other movies.

A simple histogram shows some movies are rated much more frequently than others.

```
# Create an aggregation object.
movieIds <- binit(edx$movieId)
# Plot frequency of movies.
plot(movieIds$values, movieIds$lengths
     , type="h"
     , col="purple"
     , xlab="Movie ID"
     , ylab="Count of Ratings"
     , main="Count of Ratings by Movie ID")
```



Next, ratings were grouped by movie. The counts, rating minimums and maximums, and rating means were calculated for individual movies. There are 10,677 titles ready for analysis. The *mean of the movie rating means* is 3.19 with a standard deviation of 0.571 when including all titles.

```
# Group ratings by movie, and calculate each movie's average, min, and max ratings.
ratings_by_movie <- edx %>% group_by(movieId) %>%
  summarize(movieRatingsCount = n(), movieMeanRating = mean(rating)
  ,movieMinRating = min(rating), movieMaxRating=max(rating))

# Preview the summary measures.
head(ratings_by_movie,5)
```

```
## # A tibble: 5 x 5
##   movieId movieRatingsCount movieMeanRating movieMinRating movieMaxRating
##     <dbl>             <int>           <dbl>          <dbl>          <dbl>
## 1       1             23826            3.93            0.5              5
## 2       2             10717            3.20            0.5              5
## 3       3              7053            3.15            0.5              5
## 4       4              1579            2.88            0.5              5
## 5       5              6415            3.08            0.5              5
```

```
# Count the number of unique movies (one per grouped row).
nrow(ratings_by_movie)
```

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

```
# Compute the mean and standard deviation of mean ratings by movie.
c(mean(ratings_by_movie$movieMeanRating)
  ,sd(ratings_by_movie$movieMeanRating))
```

```
## [1] 3.1920666 0.5711631
```

Theoretically, mean ratings from movies with more ratings are a higher quality measure of a movie effect than mean ratings from movies that have only been rated a few times.

A movie with only one rating of 5 would average the rating of that movie to the highest possible value of 5, even if only one person was ever interested in watching it. While this example 5 rating is not completely irrelevant, one rating is not a reliable predictor compared to movies with 6000+ ratings, for example. A threshold of relevance must be selected.

Arbitrarily, movies with fewer than 100 ratings were filtered out according to this thinking.

```
# Count the movies with fewer than 100 ratings.
ratings_by_movie %>% filter(movieRatingsCount < 100) %>% nrow()
```

```
## [1] 4962
```

```
# Show 10 examples of movies with fewer than 100 ratings.
ratings_by_movie %>% filter(movieRatingsCount < 100) %>% head(.,10)
```

```
## # A tibble: 10 x 5
##    movieId movieRatingsCount movieMeanRating movieMinRating movieMaxRating
##      <dbl>             <int>           <dbl>          <dbl>          <dbl>
##  1      33                40            3.1            0.5              5
##  2      37                50            2.83           0.5              5
##  3      51                14            2.75           0.5              5
##  4      53                84            3.73           1                5
##  5      56                42            1.90           0.5              5
##  6      84                86            3.37           1                5
##  7      98                47            2.55           1                5
##  8     106                68            3.47           1                5
##  9     108                76            3.01           1                5
## 10     109                 6            1.67           0.5              3
```

```
# Create a variable for movies with 100+ ratings.
ratings_by_movie <- ratings_by_movie %>% filter(movieRatingsCount >= 100)

# Count the rows to use for analysis.
nrow(ratings_by_movie)
```

```
## [1] 5715
```

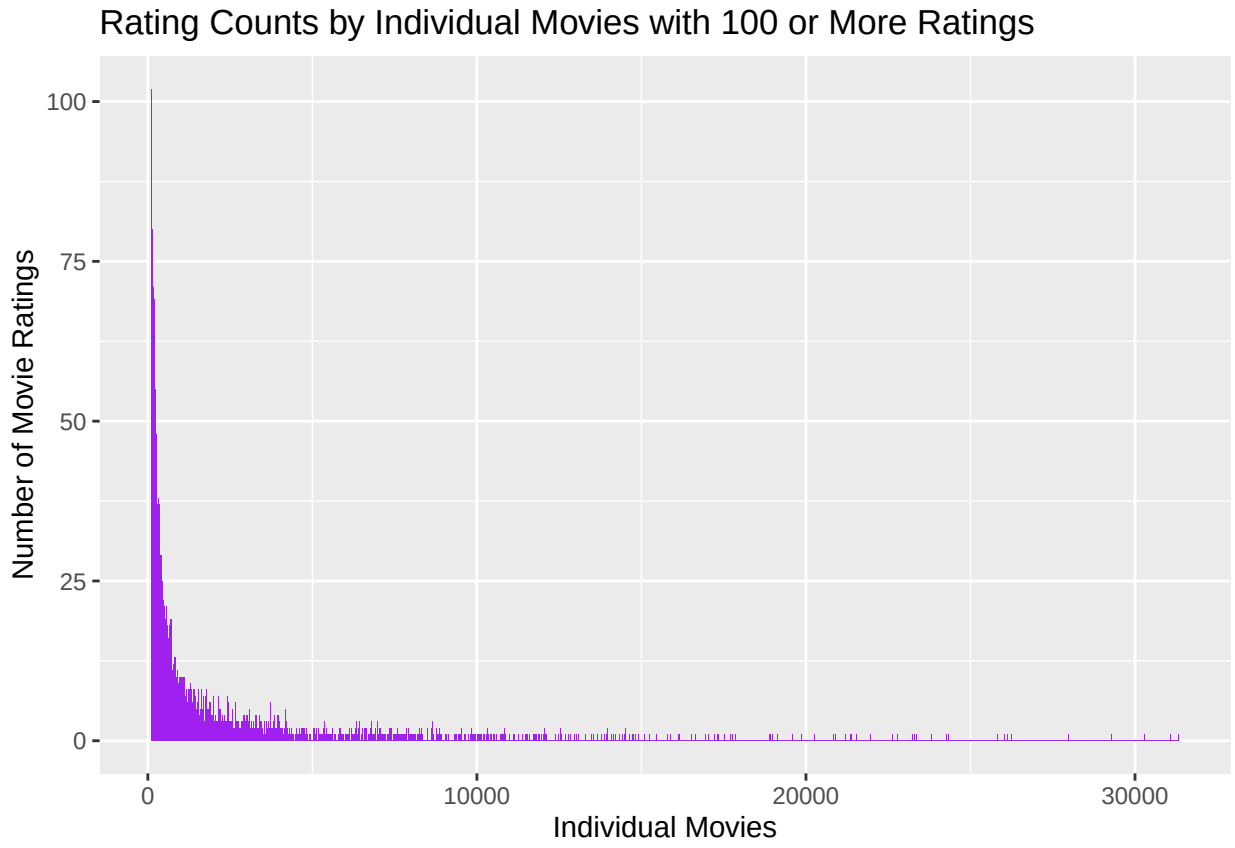
Movies with 100+ ratings have a group mean (that is the *mean of mean movie ratings* for the group) slightly higher than for all titles.

```
# Compute the mean and standard deviation of mean ratings using all titles.
c(mean(ratings_by_movie$movieMeanRating)
  ,sd(ratings_by_movie$movieMeanRating))
```

```
## [1] 3.2952160 0.5468734
```

Counts of ratings from movies with 100 or more ratings were graphed. There is a large range of variability in numbers of ratings.

```
# Graph the rating counts for the remainig movies (one vetical line per movie).
ratings_by_movie %>%
  data.frame() %>%
  ggplot(aes(x=movieRatingsCount)) +
    geom_histogram(bins=nrow(ratings_by_movie), fill="purple") +
    labs(title="Rating Counts by Individual Movies with 100 or More Ratings"
    , x="Individual Movies", y="Number of Movie Ratings")
```



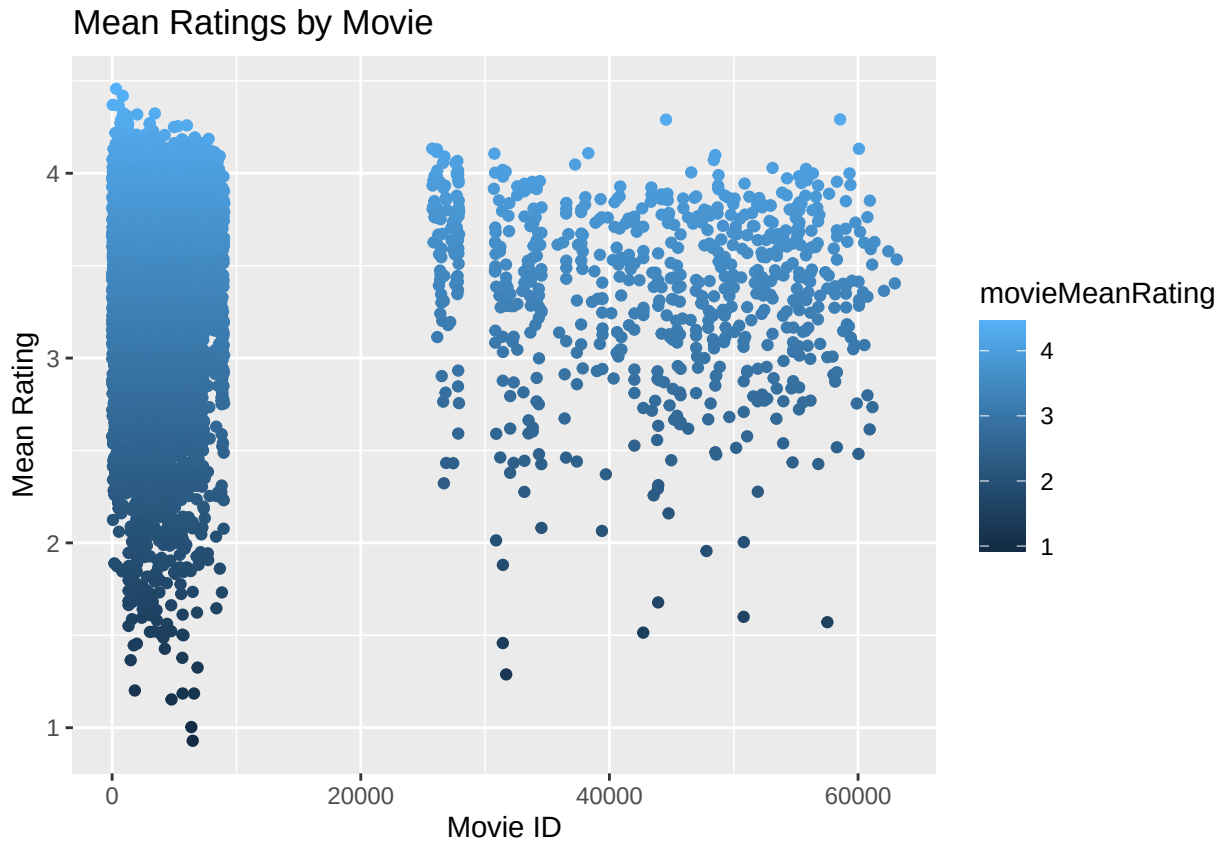
Next, the movie rating counts were plotted by movie, and shaded according to the ratings given on a scale of 0.5 to 5.0.

1. The rating counts are significantly different between movie titles.
2. The mean ratings are only weakly correlated with the number of ratings (positive correlation of 0.221).
3. Movies with high rating counts are not necessarily rated better or worse than those with lower counts.

```
# Show sample movie rating means.
head(ratings_by_movie,5)
```

```
## # A tibble: 5 x 5
##   movieId movieRatingsCount movieMeanRating movieMinRating movieMaxRating
##     <dbl>             <int>           <dbl>          <dbl>          <dbl>
## 1       1             23826            3.93            0.5              5
## 2       2             10717            3.20            0.5              5
## 3       3              7053            3.15            0.5              5
## 4       4              1579            2.88            0.5              5
## 5       5              6415            3.08            0.5              5
```

```
# Plot movie rating means by movie.
  na.exclude(ratings_by_movie) %>%
  data.frame() %>%
  ggplot(aes(x=movieId, y=movieMeanRating )) +
    geom_point(aes(color=movieMeanRating)) +
    labs(title="Mean Ratings by Movie"
    , y="Mean Rating"
    , x="Movie ID")
```



The correlation was calculated between *mean movie ratings* and *count of movie ratings* per movie. There is a weak association between movies with more ratings and those ratings being higher. This agrees with the findings above.

```
# Compute correlation of mean ratings to counts.
cor(ratings_by_movie$movieMeanRating, ratings_by_movie$movieRatingsCount)
```

```
## [1] 0.2204093
```

The mean ratings for individual movies were built into the predictive models in a later section. Because of the observed difference in mean ratings from movie to movie, a movie effect is useful for any effective predictive ratings model.

Ratings Grouped by Movie Genre Combinations

In this dataset, movie genres are shown as composite strings that include every genre that applies to a movie. Pipe-separated genre strings (the "|" pipe character is the separator) are ordered alphabetically, making the genre combinations unique. For instance, a movie in both Drama and Suspense has an entry of "Drama | Suspense" and never "Suspense | Drama".

The data was first grouped by the "genres" column.

```
# Group ratings by genres and calculate average, min and max ratings.
ratings_by_genres <- edx %>% group_by(genres) %>%
  summarize(genresRatingsCount = n(), genresMeanRating = mean(rating)
  , genresMinRating = min(rating), genresMaxRating=max(rating))

#  Preview the summary measures.
head(ratings_by_genres,5)
```

```
## # A tibble: 5 x 5
##   genres       genresRatingsCo~ genresMeanRating genresMinRating genresMaxRating
##   <chr>                   <int>            <dbl>           <dbl>           <dbl>
## 1 Action                  72327             3.65             0.5               5
## 2 Action|Adve~            53408             3.71             0.5               5
## 3 Action|Adve~               62             1.90             0.5               4
## 4 Action|Adve~               73             3.47             0.5               5
## 5 Action|Adve~              697             3.48             0.5               5
```

Next, a table with all of the unique genre combinations and their counts was created, ordered from highest to lowest count.

```
# Count the frequencies of the unique genre combinations.
genre_counts <- table(edx$genres) %>%
  data.frame %>%
  arrange(desc(Freq))

# Show the top 10 genre combinations by count.
head(genre_counts,10)
```

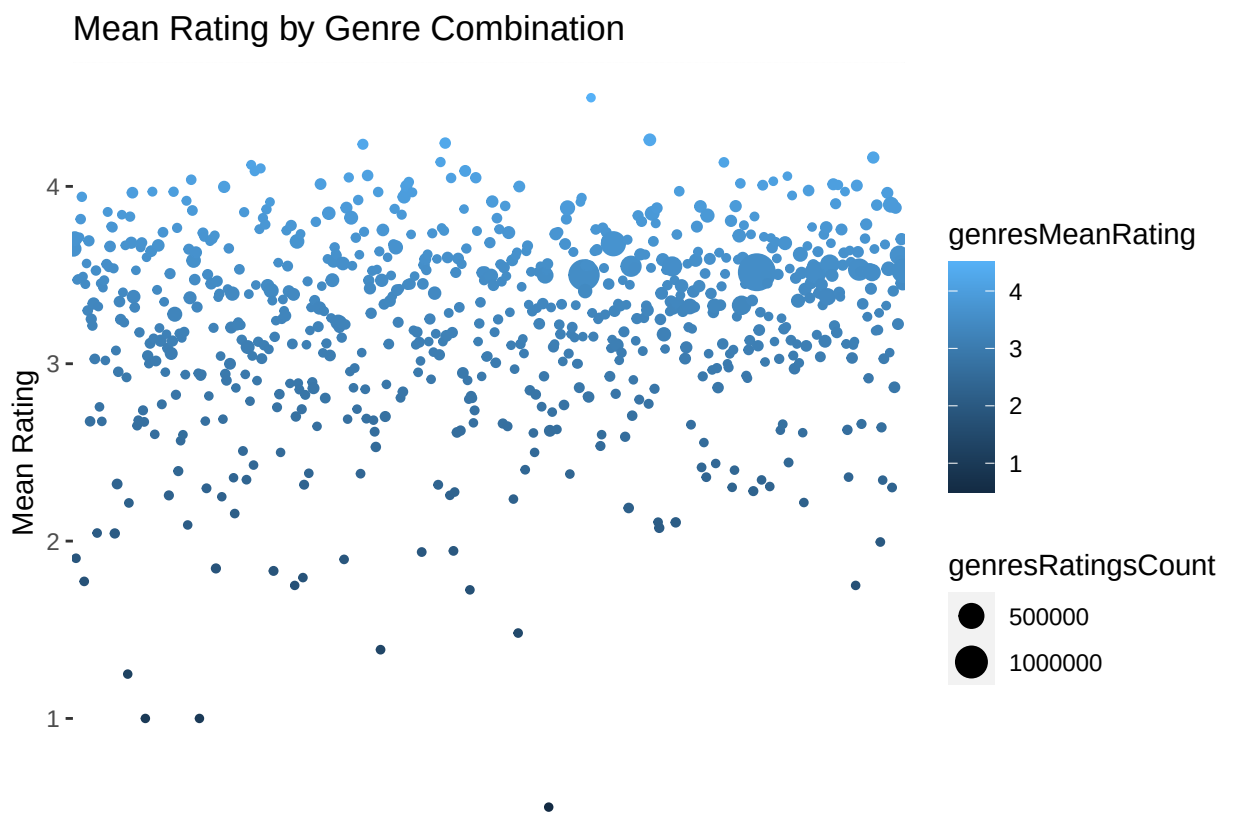
```
##                    Var1    Freq
## 1                 Drama 1467890
## 2                Comedy  878025
## 3          Comedy|Drama  451746
## 4                Horror  299812
## 5         Drama|Romance  286899
## 6  Comedy|Drama|Romance  253178
## 7        Comedy|Romance  206897
## 8           Documentary  182416
## 9             Drama|War  177632
## 10             Thriller  158417
```

```r
# Count the number of unique movies (one per grouped row).
nrow(ratings_by_genres)
```

```
## [1] 707
```

The rating means were graphed against genre combinations, with point size showing the relative number of ratings.

```r
# Graph averages for composite genres.
ratings_by_genres %>%
  data.frame() %>%
  ggplot(aes(x=genres, y=genresMeanRating)) +
    geom_point(aes(col=genresMeanRating, size=genresRatingsCount)) +
    labs(title="Mean Rating by Genre Combination"
         , x="Genre Combination", y="Mean Rating") +
     theme(axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```



Decomposition of Genre Strings

While genre combination strings confer the advantages of uniqueness and uniformity to analysis, one drawback is the inability to consider individual genres without further processing.

The string content of the "genres" column was decomposed into individual genre columns below, where each genre is either TRUE or FALSE for a rated movie row.

```
# Identify all unique genre combinations covered by genres field.
genre_names <- unique(unlist(edx$genres))

# Invoke stringr library to manipulate strings.
if(!require(corrplot)) install.packages("stringr", repos = "http://cran.us.r-project.org")
library(stringr)

# Extract all individual genres from composite strings, and show them.
(unique_genres <- genre_names %>% strsplit("\\|") %>% unlist() %>% unique())
```

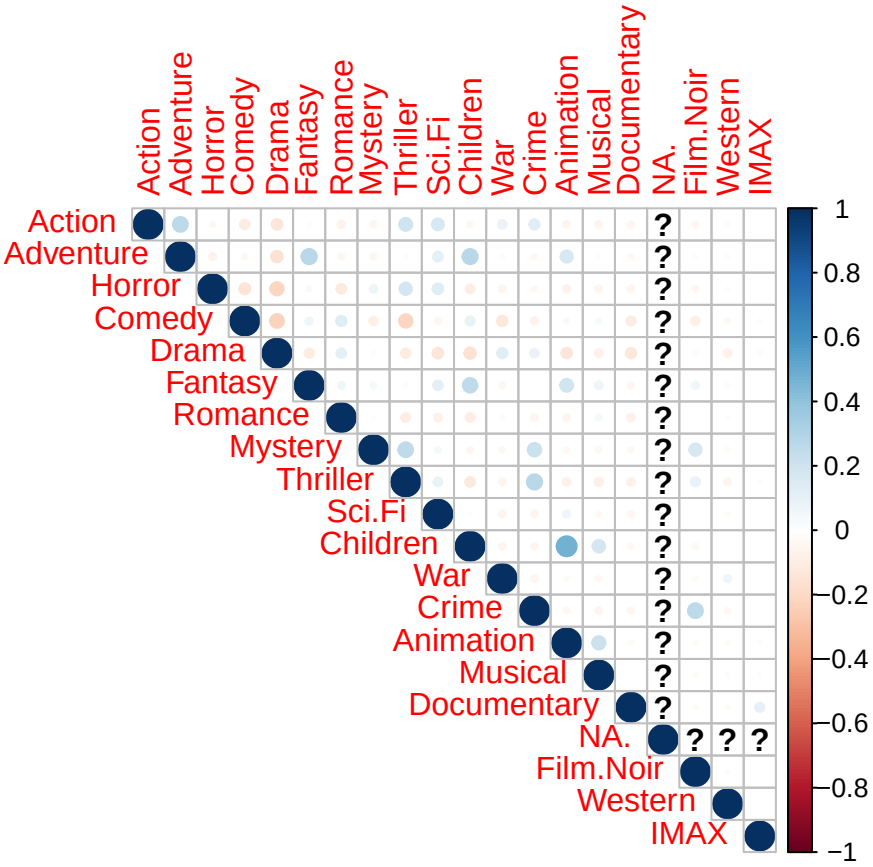
```
##  [1] "Action"      "Adventure"   "Horror"      "Comedy"      "Drama"
##  [6] "Fantasy"     "Romance"     "Mystery"     "Thriller"    "Sci-Fi"
## [11] "Children"    "War"         "Crime"       "Animation"   "Musical"
## [16] "Documentary" NA            "Film-Noir"   "Western"     "IMAX"
```

```
# Create a function to determine if "genres" string includes a specified genre.
is_this_genre <- function(input_genre){
  x <- grepl(input_genre, edx$genres, fixed = TRUE)
  return(x)
}

# Create a table containing each genre that applies to a movie.
genres_table <- sapply(unique_genres, is_this_genre) %>% data.frame()

# Preview select genres.
head(genres_table,5) %>% select(Drama, Comedy, Action, Adventure, Thriller)
```

```
##   Drama Comedy Action Adventure Thriller
## 1 FALSE  FALSE   TRUE     FALSE    FALSE
## 2 FALSE  FALSE   TRUE      TRUE    FALSE
## 3  TRUE   TRUE  FALSE     FALSE    FALSE
## 4 FALSE   TRUE  FALSE     FALSE     TRUE
## 5  TRUE  FALSE  FALSE     FALSE    FALSE
```

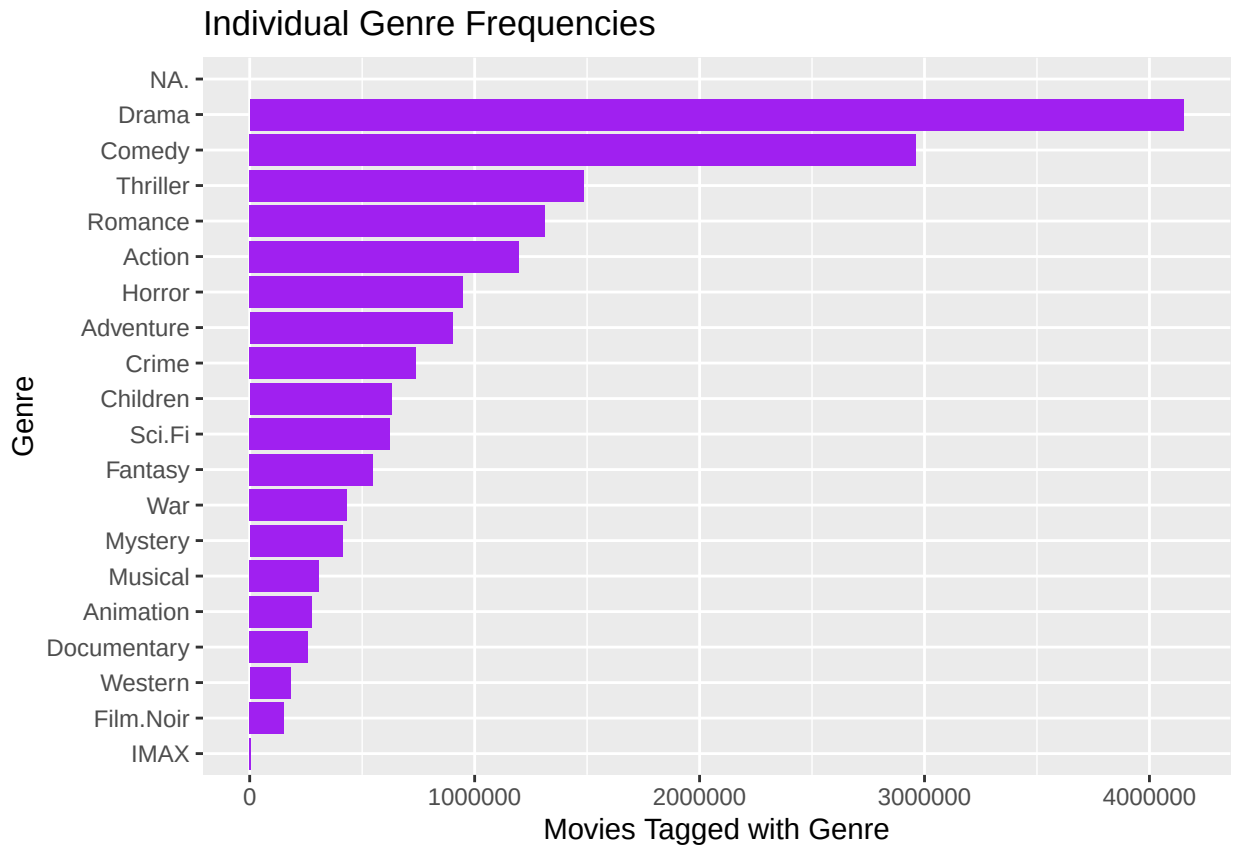
Some genres correlate with each other, meaning movies are likely to combine their genres.

```
# Build a correlation matrix of genres with each other.
genres_matrix <- cor(genres_table, method="pearson") %>% round(digits=4)

# Install (if needed) and invoke correlation plotting library.
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
library(corrplot)
corrplot(genres_matrix, method="circle", type="upper")
```



The genres of Drama, Comedy, Action, Adventure Thriller and Adventure are the most frequent tags among rated movies.

```
# Total the genre columns.
genre_totals <-  cbind(names(genres_table)
                , colSums(genres_table)  )%>%
                  data.frame()

# Since all fields are factors, make a quantity of the totals column.
genre_totals <- genre_totals %>%
  mutate (total=as.numeric(as.character(genre_totals$X2)))

# Graph the individual genre tag frequencies.
genre_totals %>%
  mutate(genre = reorder(X1, total)) %>%
  ggplot(aes(x=genre, y=total)) +
    geom_bar(stat="identity", fill="purple") +
    labs(x="Genre", y="Movies Tagged with Genre",
      title="Individual Genre Frequencies") +
      coord_flip()
```



Rating means were calculated for the most abundantly rated genres. These did not indicate strong differences in mean ratings, wether a movie reported TRUE or FALSE for any single genre. All the rating means rounded to the 3.5 rating available for users to select. Therefore, the composite genre dimension appears to be a stronger predictor than individual genre tags.

```
# Bind individual genres table to original table.
edx_genres <- cbind(edx, genres_table)

# Calculate means for each high count genre.
edx_genres %>% group_by(Thriller) %>% summarize(averageRating=mean(rating))
```

```
## # A tibble: 2 x 2
##   Thriller averageRating
##   <lgl>            <dbl>
## 1 FALSE             3.52
## 2 TRUE              3.48
```

```
edx_genres %>% group_by(Drama) %>% summarize(averageRating=mean(rating))
```

```
## # A tibble: 2 x 2
##   Drama averageRating
##   <lgl>         <dbl>
## 1 FALSE          3.51
## 2 TRUE           3.51
```

```
edx_genres %>% group_by(Comedy) %>% summarize(averageRating=mean(rating))
```

```
## # A tibble: 2 x 2
##   Comedy averageRating
##   <lgl>          <dbl>
## 1 FALSE           3.50
## 2 TRUE            3.53
```

```
edx_genres %>% group_by(Adventure) %>% summarize(averageRating=mean(rating))
```

```
## # A tibble: 2 x 2
##   Adventure averageRating
##   <lgl>             <dbl>
## 1 FALSE              3.51
## 2 TRUE               3.50
```

```
edx_genres %>% group_by(Action) %>% summarize(averageRating=mean(rating))
```

```
## # A tibble: 2 x 2
##   Action averageRating
##   <lgl>          <dbl>
## 1 FALSE           3.53
## 2 TRUE            3.43
```

Though tried in experiments, individual genre tags were not used in the reported models. Instead, the composite "genres" field was used.

Time Variables

The time variables available in the starting data set include a timestamp for when a rating was given, and a year of movie release (appended to the end of the movie title string).

For analytical and modeling purposes, the following further columns were derived:

1. *ratingdate*: The timestamp of the rating provided by the user, converted to a POSIX format. It is assumed the original timestamp has the standard UTC start date of the first second on January 1, 1970.
2. *ratingyear*: The year a movie rating was provided, derived from the timestamp of the rating. It was hypothesized that trends during the year a rating was given could influence the user's rating. The *rating year effect* is distinguished from the *lag between the time the movie was released and the time when it was rated.*
3. *ratinglag*: This is the lag between the year of release of a movie and the year a rating was given. For simplicity, both the year of release and year of rating were reduced to a four-digit number. While this is not ideal for fractional years between release and rating, the use of this field (as opposed to its absence) recognizes a potential "lag effect" that is distinct from simply the year of rating. It is hypothesized that the same lag between release and rating has a distinct impact on ratings.

```
# Derive fields for year of release, year of rating and whole year
#  lag between release and rating (as described above).
edx_derived <- edx %>%   mutate(year=str_sub(title,-5,-2)) %>%
                         mutate(ratingdate=as.POSIXct(`timestamp`
                       , origin = "1970-01-01")
                       , ratingyear = format(ratingdate,'%Y')
                       , ratinglag = as.numeric(ratingyear)-as.numeric(year)) %>% ungroup()

# Preview rows, showing new fields.
head(edx_derived,5) %>% select(movieId, ratingdate, ratingyear, ratinglag)
```

```
##    movieId          ratingdate ratingyear ratinglag
## 1:     122 1996-08-02 04:24:06       1996         0
## 2:     185 1996-08-02 03:58:45       1996         0
## 3:     231 1996-08-02 03:56:32       1996         9
## 4:     292 1996-08-02 03:57:01       1996        52
## 5:     316 1996-08-02 03:56:32       1996         9
```

```
# For analysis and modeling, remove movies with fewer than 100 ratings,
#  ratings from users who have rated fewer than 100 movies,
#  and genre combinations that appear in fewer than 100 movies.
edx_derived <- edx_derived %>%
  group_by(movieId) %>% filter(n()>=100) %>%
  group_by(userId) %>% filter(n()>=100) %>%
  group_by(genres) %>% filter(n()>=100) %>%
  ungroup() %>%
  select(userId, movieId, rating, timestamp
  , title, genres, year, ratingdate
  , ratingyear, ratinglag)

# Check for malformed years, using regular expression, count
# rows where they are 4 digit years.
grepl("^[0-9]{4}$", edx_derived$year) %>% sum()
```

```
## [1] 6440346
```

```
# Compare with all rows (should be equal as all years should be 4 digits).
nrow(edx_derived)
```

```
## [1] 6762226
```

Movie Release Year

Movies release years in the dataset range freom 1915 to 2008.

```
# Earliest year of release.
min(as.numeric(edx_derived$year))
```

```
## [1] NA
```

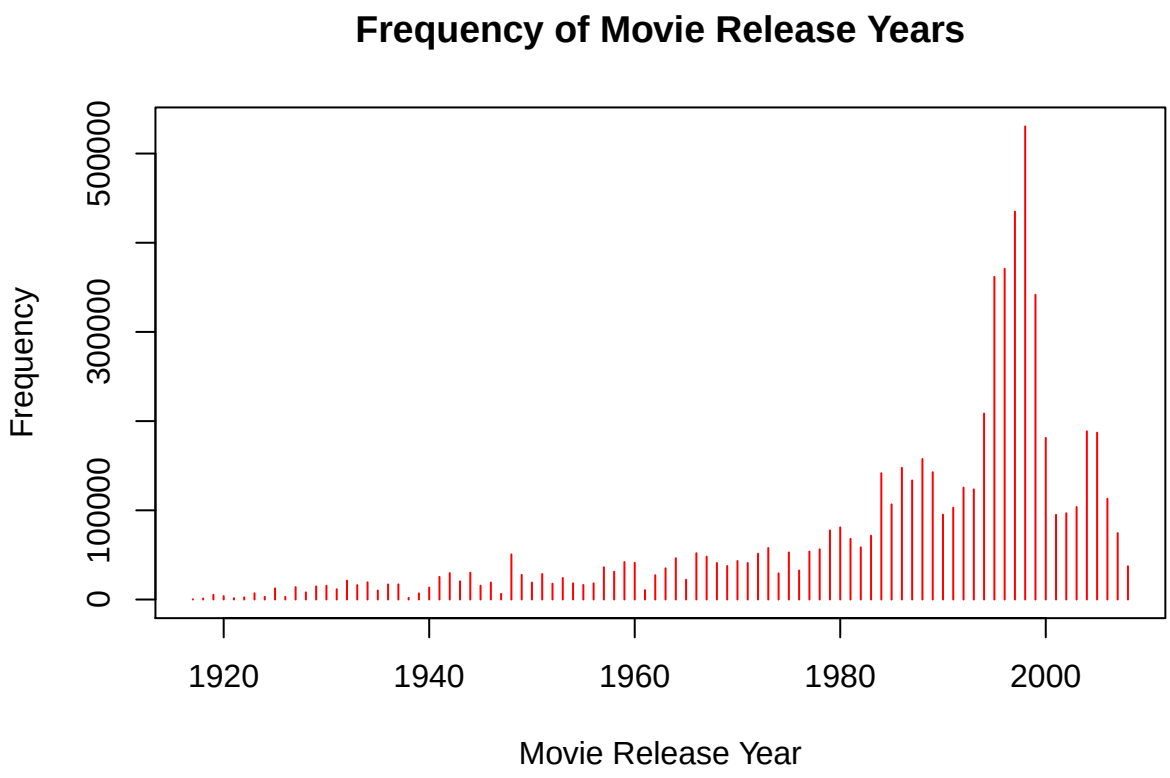
```
# Latest year of release.
max(as.numeric(edx_derived$year))
```

```
## [1] NA
```

Of the movies rated, the frequency of ratings by release years increased from about 1980 to a peak around 1995, then declines steeply towards 2008.

Years with more ratings data could be related to the onset of digital streaming, lifting the tide of movies and the abundance of ratings in general, or to other hypothetical factors. More ratings in a given year may contribute to robustness of the mean ratings for those years.

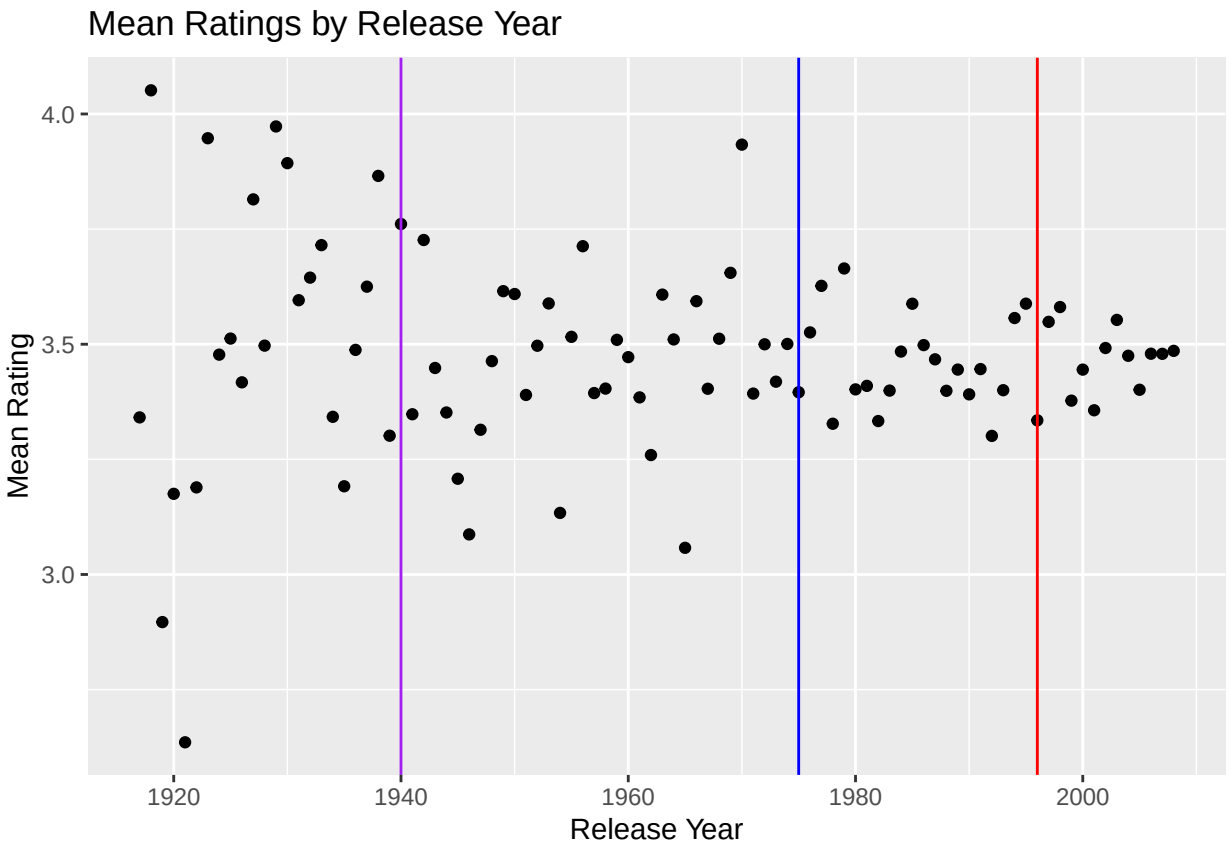
```
# Plot the frequencies of movie release years.
#  Use binit from "cgwtools" library to create
#  an object with "values" (ex. 1985)
#  and "lengths" (the count of occurrences for each "value").
plot_year <- binit(edx_derived$year)

# Turn off scientific notation.
options(scipen=999)

# Plot the object to show a histogram of movie release year frequencies.
# Add labels and color.
plot(plot_year$values, plot_year$lengths, type="h", col="red",
     ylab = "Frequency", xlab = "Movie Release Year"
     , main = "Frequency of Movie Release Years")
```



Average ratings by movie release years were plotted next:

1. Ratings for movies released before 1940 were slightly more distributed than other time frames.
2. Besides outlier means, those released in 1975 and before invited higher average ratings, in general.
3. Average ratings of movies released after 1975 dropped off severely, descending towards a lower mean of around 3.3 until about 1996.
4. Rating means started to rise again in 1996, climbing towards 2008, but never reaching the heights pre-1975 movies.

```
# Group ratings by release year, and calculate each movie's
#  count, average and SD.  Add a numericYear field to convert
#  the data type of the year label into a number, for graphing.
ratings_by_year <- edx_derived %>% group_by(year) %>%
  summarize(yearRatingsCount = n(), yearMeanRating=mean(rating)
            , yearSD=sd(rating), numericYear=as.numeric(year))

# Use distinct entries in the grouped table for faster graphing.
plot_summary <- ratings_by_year %>% distinct() %>% data.frame

# Graph mean ratings by release year.
ggplot(plot_summary, aes(x=numericYear, y=yearMeanRating)) +
  geom_point() +
  geom_vline(xintercept=1975, colour="blue") +
  geom_vline(xintercept=1996, colour="red") +
  geom_vline(xintercept=1940, colour="purple") +
  labs(title="Mean Ratings by Release Year"
       , y="Mean Rating"
       , x="Release Year")
```



These differentiated means by release year were convincing enough to use it as a predictor in the models.

Movie Rating Year

The starting data set covers ratings submitted between 1996 and 2009.

```
# Earliest and latest year of ratings.
c(min(edx_derived$ratingyear), max(edx_derived$ratingyear))
```

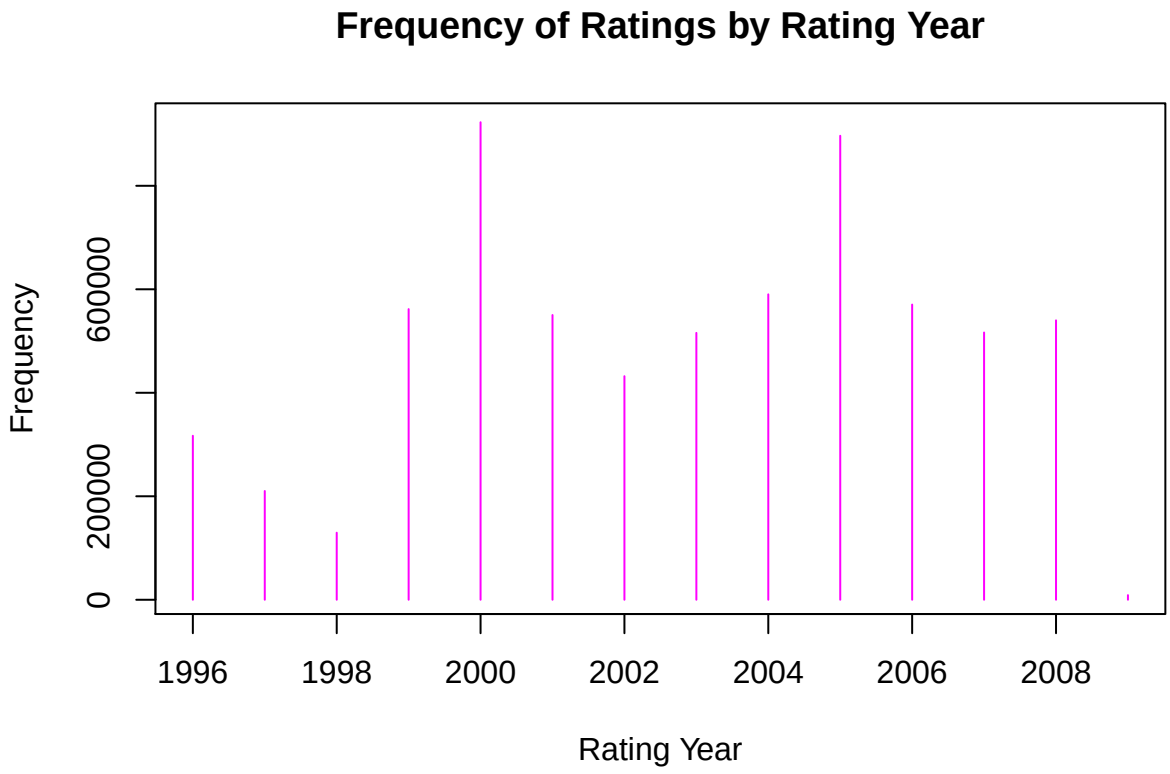
```
## [1] "1996" "2009"
```

Rating frequencies were plotted by rating year:

1. Some years invited many more ratings than others.
2. In particular the years 2000 and 2005 saw many more ratings than other years.
3. There are too many potential causal factors to speculate on here, ranging from when movies became available to rate (in batches) to what alternative activities became popular, to changes in work schedules.

```
# Plot the frequencies of movie release rating years.
plot_ratingyear <- binit(edx_derived$ratingyear)

# Plot the object to show a histogram of movie release ratingyear frequencies.
```

```
# Add labels and color.
plot(plot_ratingyear$values, plot_ratingyear$lengths, type="h", col="magenta",
     ylab = "Frequency", xlab = "Rating Year"
     , main = "Frequency of Ratings by Rating Year")
```



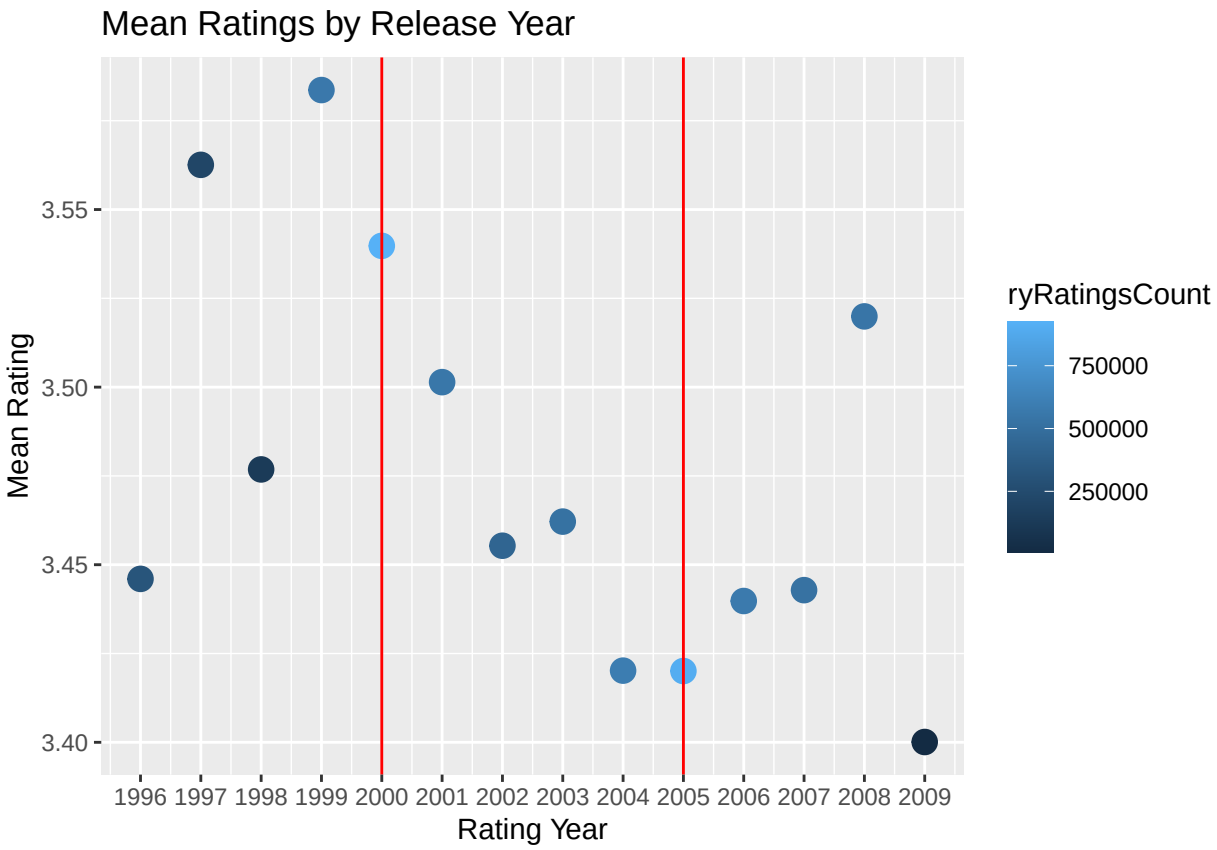
Next, mean ratings were plotted for each rating year:

1. The graph was sectioned for the two high-frequency years of 2000 and 2005.
2. From 1996 to 2000, ratings averages were sporadic relative to the database mean.
3. 2000 to 2005 saw a secular decrease in mean ratings.
4. Beginning in 2005, average ratings become sporadic again, but around a lower mean.

```
# Group ratings by rating year,
ratings_by_ratingyear <- edx_derived %>% group_by(ratingyear) %>%
  summarize(ryRatingsCount = n(), ryRating=mean(rating)
            , ryNumericYear=as.numeric(ratingyear))

# Use distinct entries in the grouped table for faster graphing.
plot_summary <- ratings_by_ratingyear %>% distinct() %>% data.frame

# Fewer rows to plot.
nrow(plot_summary)
```

```
## [1] 14
```

```
# Graph mean ratings by release year.
#  Add color as a dimension for rating counts (color of the dot).
#  Increase dot size to 4.  Add titles.
#  Add lines for 2000 and 2005 inflections.
#  Show every rating year on the x-axis by specifying the range.
ggplot(plot_summary, aes(x=ryNumericYear, y=ryRating
       , color=ryRatingsCount)) +
  geom_point(size=4) +
  labs(title="Mean Ratings by Release Year"
       , y="Mean Rating"
       , x="Rating Year") +
        geom_vline(xintercept=2000, colour="red")  +
        geom_vline(xintercept=2005, colour="red")  +
        scale_x_continuous(breaks=seq(1995,2009,1))
```



The insights from rating years were convincing enough to encapsulate them into a "rating year" predictor in models.

Rating Lag Indicator

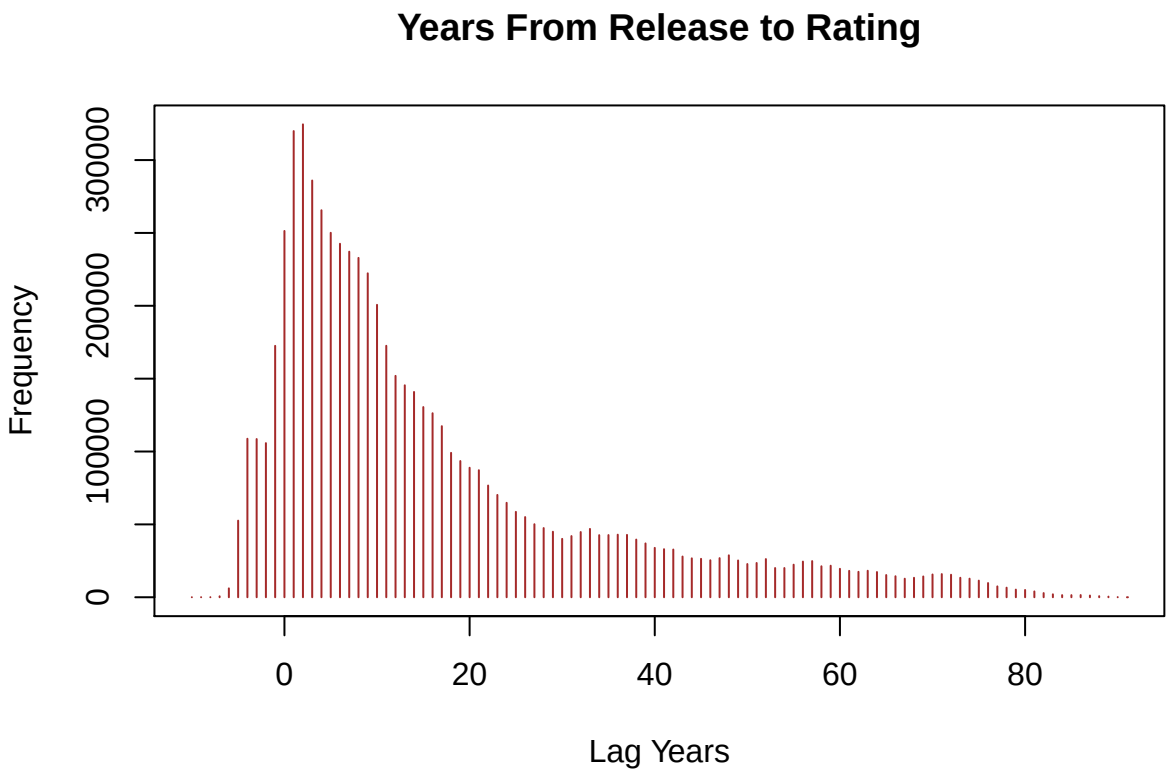
A rating lag variable represents the time between a movie's release rating by a user, measured by the difference in numerical years. Trends at the time of rating can impact movies based their age.

Lag frequencies were plotted first.

```r
# Create an object to facilitate plotting the lag variable
# (years between movie release and user's rating for the row).
plot_lag <- binit(edx_derived$ratinglag)

# Preview object to plot.
head(plot_lag,5)
```

```
## $lengths
##   [1]     39     52     54    661   6124  52572 108776 108570 105768 172525
##  [11] 251350 319891 324505 285982 265572 250092 242570 237132 232878 222289
##  [21] 200669 172566 151925 145460 140881 130513 126275 117416  99085  93465
##  [31]  88888  87205  76670  70220  64817  58592  55022  50120  47495  44968
##  [41]  40078  41974  44758  46897  42625  42669  42953  42798  39576  36942
##  [51]  33892  32988  32767  27951  26727  26368  25409  26850  28785  25255
##  [61]  22807  23517  26216  20068  20062  22363  24479  24816  21264  21649
##  [71]  19532  18128  17464  18192  17311  15242  14429  12735  13357  14275
##  [81]  15624  15908  15450  13385  12762  11425   9762   7466   6594   5258
##  [91]   4994   4013   2870   2027   1471   1454   1525   1200    758    451
## [101]    117     10
##
## $values
##   [1] -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6   7
##  [19]   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25
##  [37]  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43
##  [55]  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61
##  [73]  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79
##  [91]  80  81  82  83  84  85  86  87  88  89  90  91
```

```r
# Plot the lag frequencies.
plot(plot_lag$values, plot_lag$lengths, type="h", col="brown",
     ylab = "Frequency", xlab = "Lag Years", main = "Years From Release to Rating")
```



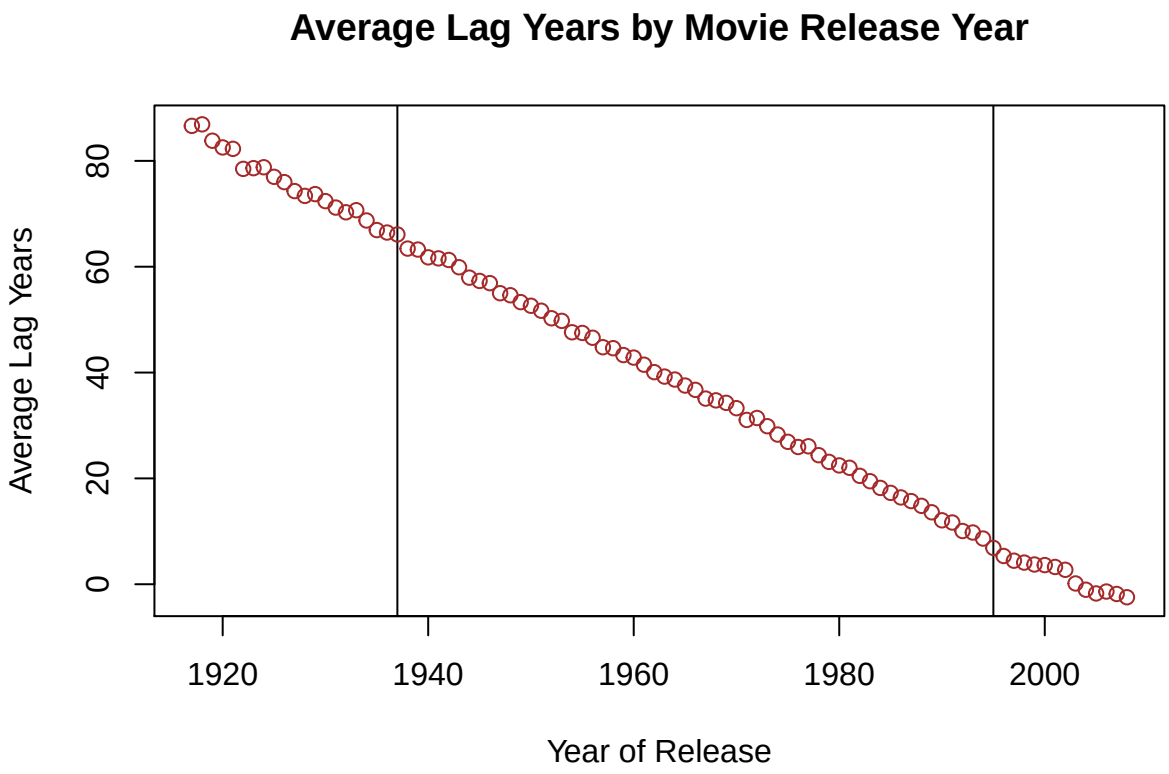
The average lags were also plotted by year of release to clarify the lag effect. For instance, a 5-year lag for a movie released in 2000 means it is rated in the year 2005. A 5-year lag for a movie released in 1991 means it was rated in 1996. These are both 5-year lags, but could mean very different impacts on a predicted rating.

Plotting the average lag by movie release year shows the following:

1. Very old movies, released from 1915 up to the late 1930's (around 1937) show slightly more variation in average lag than for later years.
2. Movies released from the 1940's through to the 1990's show a roughly linear decline in lag. This is expected because more movies were released in this period compared to earlier, and the earliest possible ratings were in 1996.
3. There is a flattening of lag in the late 1990's that continues to the latest movie release year of 2008. This could reflect movies viewed and rated very soon after release, a preference for watching more recently released movies, or both and more.

The above observations made lag between release and rating an interesting, but likely not a strongly influential predictor to add in the modeling section.

```
# Plot average lags by year of release.
edx_derived %>% group_by(year) %>%
  summarize(avg_lag=mean(ratinglag)) %>%
  plot(main="Average Lag Years by Movie Release Year"
  ,xlab="Year of Release"
  ,ylab="Average Lag Years"
  ,col="brown"
  ,abline(v=c(1995,1937)))
```



Models

Measure of Effectiveness

The Root Mean Square Error of predictions (versus actual ratings) was used to evaluate the effectiveness of models. This function encapsulates how close a prediction is to actual ratings by using Euclidian distances.

1. The calculation first finds the difference (the error) between the true rating and the predicted rating.
2. It then squares these differences, putting negative differences on proportional footing with positive differences. For example, -0.5 squared is the same as 0.5 squared. This operation is a step in recognizing absolute differences as distances.
3. Next, the squared distances are summed to capture a numerator representing all the distances.
4. The sum is then averaged to represent the average of all errors. (The summing in the R code is implicit because the true_ratings and predicted_ratings are entire vectors, not individual numbers).
5. Taking the square root of the average returns the number to its initial scale as an error term (by reversing the effect of the squaring done earlier).

A function is created to calculate RMSE from input estimates and actuals. Lower scores are better because they represent less error in the predictions.

```
# Build the RMSE function to evaluate models.
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Data Preparation for Linear Model

To reflect the discussion of features in the above sections, derived time fields were added to the edx data set.

```
# Derive new columns and filter training data.
edx_extended <- edx %>%
    mutate(year=str_sub(title,-5,-2)
         , ratingdate = as.POSIXct(`timestamp`, origin = "1970-01-01")
         , ratingyear = format(ratingdate,'%Y')
         , ratinglag = as.numeric(ratingyear)-as.numeric(year))
```

The set is further filtered to reflect prior insights.

The following rows have been removed:

1. Ratings for movies with under 100 ratings.
2. Ratings from users who have rated fewer than 100 movies.
3. Ratings in composite movie genres that have fewer than 100 ratings.

The edx_extended set was split into:

1. *edx_train*: A training set, consisting of 80% of the original edx data.
2. *edx_test*: A testing set, consisting of 20% of the original edx data.

The validation set is left alone.

```
# Filter out extraneous records
edx_extended <- edx_extended %>%
           group_by(movieId) %>% filter(n()>=100) %>%
            group_by(userId) %>% filter(n()>=100) %>%
            group_by(genres) %>% filter(n()>=100) %>%
            ungroup() %>%
            select(userId, movieId, rating, timestamp
            , title, genres, year, ratingdate
            , ratingyear, ratinglag)

# Count rows
nrow(edx_extended)
```

```
## [1] 6762226
```

```r
# Set seed tp make randomization repeatable.
set.seed(1)

# Test set taken as 20% of edx dataset.
test_index <- createDataPartition(y = edx_extended$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx_extended[-test_index,]
hold_for_test <- edx_extended[test_index,]

# Ensure userId and movieId in test set are also in training set
edx_test <- hold_for_test %>%
  semi_join(edx_extended, by = "movieId") %>%
  semi_join(edx_extended, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(hold_for_test, edx_test)
edx_train <- rbind(edx_train, removed)
rm(test_index, hold_for_test, removed, edx_extended)
```

Built-up Linear Model

A linear model was built up, starting with using the bare mean, then adding *bias* or *effect* terms to represent influential predictors identified in the above sections of this report.

Mean as Model

The mean for test set is slightly different than from the original set.

```r
# Compute mean rating from test data set.
(mu <- mean(edx_train$rating))
```

```
## [1] 3.480144
```

```r
# Compare to mean rating of original set.
(mu_original <- mean(edx$rating))
```

```
## [1] 3.512464
```

The mean from the training set was used in the models.

```r
# Use the training set mean as the prediction.
#  Test against the test data set.  Store this for later comparison.
(rmse_mu <- RMSE(edx_test$rating, mu))
```

```
## [1] 1.052902
```

Movie Effect: b_i

The first effect for the linear model is added to represent the effect of each movie. That is, each movie theoretically attracts its own mean rating that is different from other movies' mean ratings, all other factors held constant. Average ratings by movie were analyzed in the *Ratings Grouped by Movie* section above.

The effect b_i is added to the model, to explain the portion of the rating that isn't explained by the mean (mu). This is done by subtracting the overall mean rating from the grouped movie mean rating, for each movie.

```
# Calculate movie effect.
movie_avgs <- edx_train %>% group_by(movieId) %>%
  summarize(b_i = mean(rating-mu))

# View a sample of the movie biases.
head(movie_avgs,5)
```

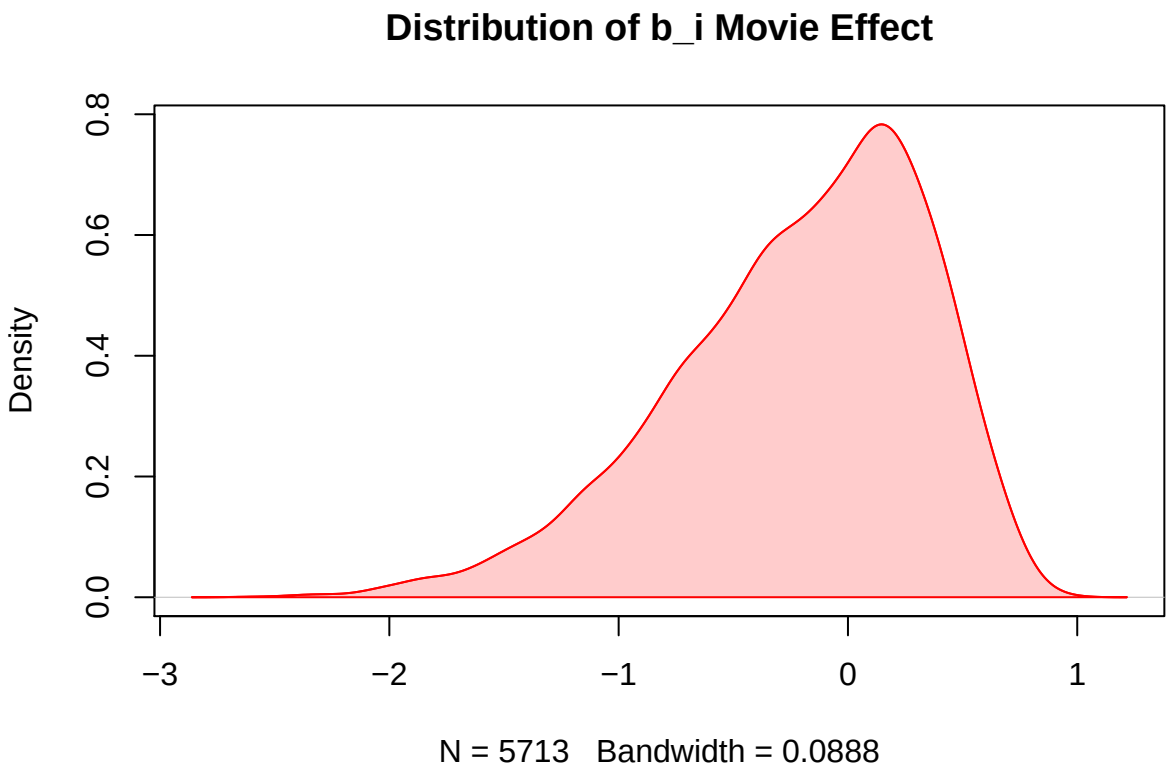
```
## # A tibble: 5 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.419
## 2       2 -0.421
## 3       3 -0.449
## 4       4 -0.690
## 5       5 -0.567
```

The distribution of b_i is skewed left, so the mean is to the left of the peak (most highly occurring b_i). Inspection of the density plot shows there are many movies that drag the prediction up by up to a full rating point and many others that drag the rating down by up to 2 rating points, and still some that pull the rating down by almost 3 points.

```
# Compute the average b_i.
mean(movie_avgs$b_i)
```

```
## [1] -0.2096744
```

```
# Plot the distribution of b_i as a density plot with a red line.
plot(density(movie_avgs$b_i)
    ,main="Distribution of b_i Movie Effect"
    ,col="red")

# Color under the curve using a polygon.
# Use RGB codes transparency.  Later graphs will overlap.
polygon(density(movie_avgs$b_i), col=rgb(1,0,0,0.2), border="red")
```



Applying the movie averages improves the RMSE significantly. The RMSE produced using the improved model is consistently better, whether testing against the training set or the test set. To evaluate the model against the test set, predictions must be calculated on the test data using the b_i averages from the training set.

```
# Join movie_avgs to main data to get b_i and compute predictions.
predicted_ratings <- edx_train %>%
  left_join(movie_avgs, by= 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull (pred)

# Evaluate against training set to compare against results in
#  test set.
RMSE(edx_train$rating, predicted_ratings)
```

```
## [1] 0.9271508
```

```
# Make predictions using training averages on test data calculateions.
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by= 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull (pred)

# Compute and store RMSE.
(rmse_mu_bi <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 0.9285891
```

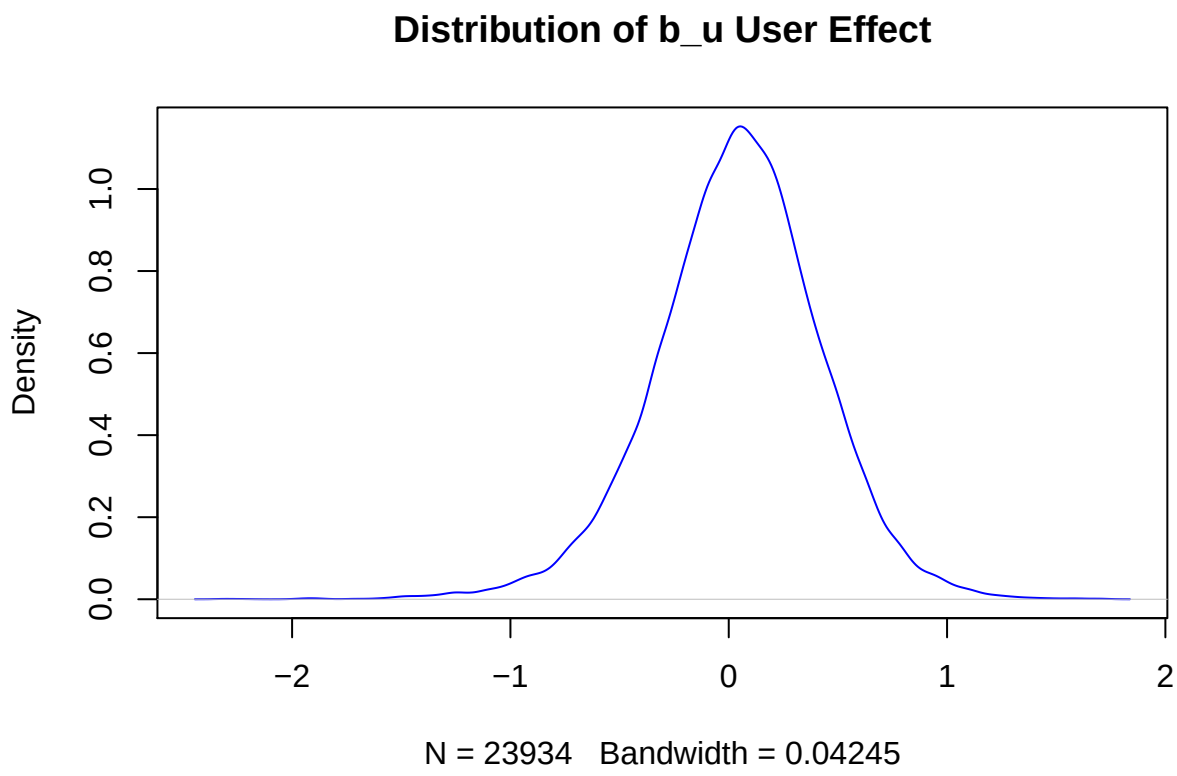
User Effect: b_u

A user effect b_u was computed using the same technique as for b_i and evaluated using RMSE. User effects were analyzed in detail in the *Ratings Grouped by User* section above.

```
# Add user effect.
# Join back to movie_avgs to get b_i.
# Must subtract both mu and b_i from actual rating to compute b_u.
user_avgs <- edx_train  %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-mu-b_i))

# Inspect some user averages.
head(user_avgs,6)
```

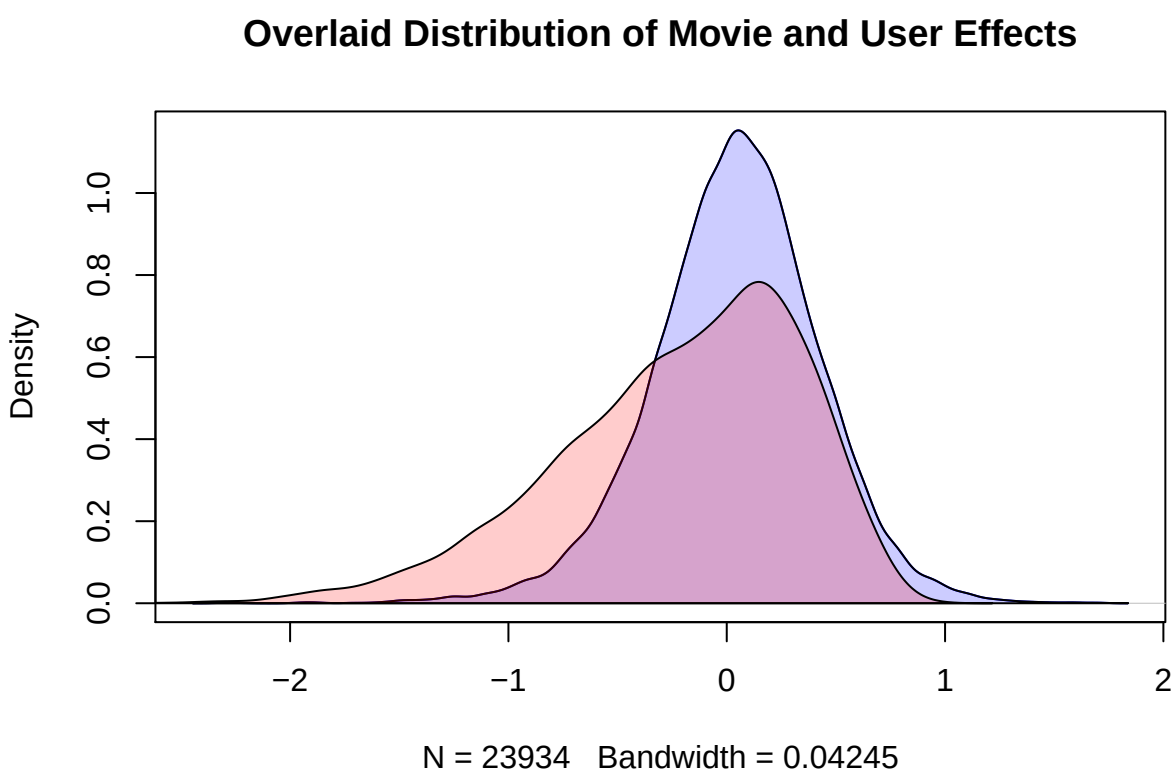
```
## # A tibble: 6 x 2
##   userId     b_u
##    <int>   <dbl>
## 1      8 0.237
## 2     10 0.124
## 3     13 0.00836
## 4     18 0.177
## 5     19 0.0792
## 6     30 0.911
```

```
# Plot the distribution of b_u with a blue line.
plot(density(user_avgs$b_u)
    ,main="Distribution of b_u User Effect"
    ,col="blue")
```



The distribution of b_u is closer to normal than the left skewed distribution of b_i. The two distributions are displayed together here:

```
# Plot the distribution of b_u again.
plot(density(user_avgs$b_u)
    ,main="Overlaid Distribution of Movie and User Effects"
    ,col="blue")
# Overlay with b_u density shaded blue.
polygon(density(user_avgs$b_u), col=rgb(0,0,1,0.2))
# Overlay the plot of b_i shaded red.
polygon(density(movie_avgs$b_i), col=rgb(1,0,0,0.2))
```



The b_u and b_i densities mostly overlap, though the blue b_u effect has a higher peak, with a mean very close to zero.

```
# Mean b_u
mean(user_avgs$b_u)
```

```
## [1] 0.04046446
```

A RMSE score was computed for the model to include the mean, movie effects and user effects.

```
# Compute new predictions with b_u added.
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by= 'movieId') %>%
  left_join(user_avgs, by= 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull (pred)

# Compute RMSE.
(rmse_mu_bi_bu <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 0.8485752
```

The addition of a user effect significantly improves the model, as measured by RMSE.

```
# Compare to prior RMSE scores.
rmse_mu
```

```
## [1] 1.052902
```

```
rmse_mu_bi
```

```
## [1] 0.9285891
```

```
rmse_mu_bi_bu
```

```
## [1] 0.8485752
```

User Effect Without Movie Effect: b_ua

Because of the overlap shown between b_i and b_u, the user effect was also tested alone (with mu but without the movie effect involved).

```
# Must recompute b_u to take out effect of b_i.
# Call this b_ua to avoid confusion.
user_avgs_alone <- edx_train  %>%
  group_by(userId) %>%
  summarize(b_ua = mean(rating-mu))

# Compute new predictions with only mu and b_ua.
predicted_ratings <- edx_test %>%
  left_join(user_avgs_alone, by= 'userId') %>%
  mutate(pred = mu + b_ua) %>%
  pull (pred)

# Compute RMSE.
(rmse_mu_bua <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 0.9701336
```

```
# Compare with movie only effect.
rmse_mu_bi
```

```
## [1] 0.9285891
```

The movie effect is more influential than the user effect, as evaluated by comparing the RMSE using b_ua alone versus b_i alone. The movie effect without the user effect returns a lower RMSE.

The computation of b_ua was done purely to illustrate that the *standalone impact* that a particular effect has on the model's RMSE is distinct from the *incremental impact* it adds to a model already containing other effects.

From here forward, the models excluded b_ua and included b_u to represent the user effect.

Gemre Combination Effect: b_g

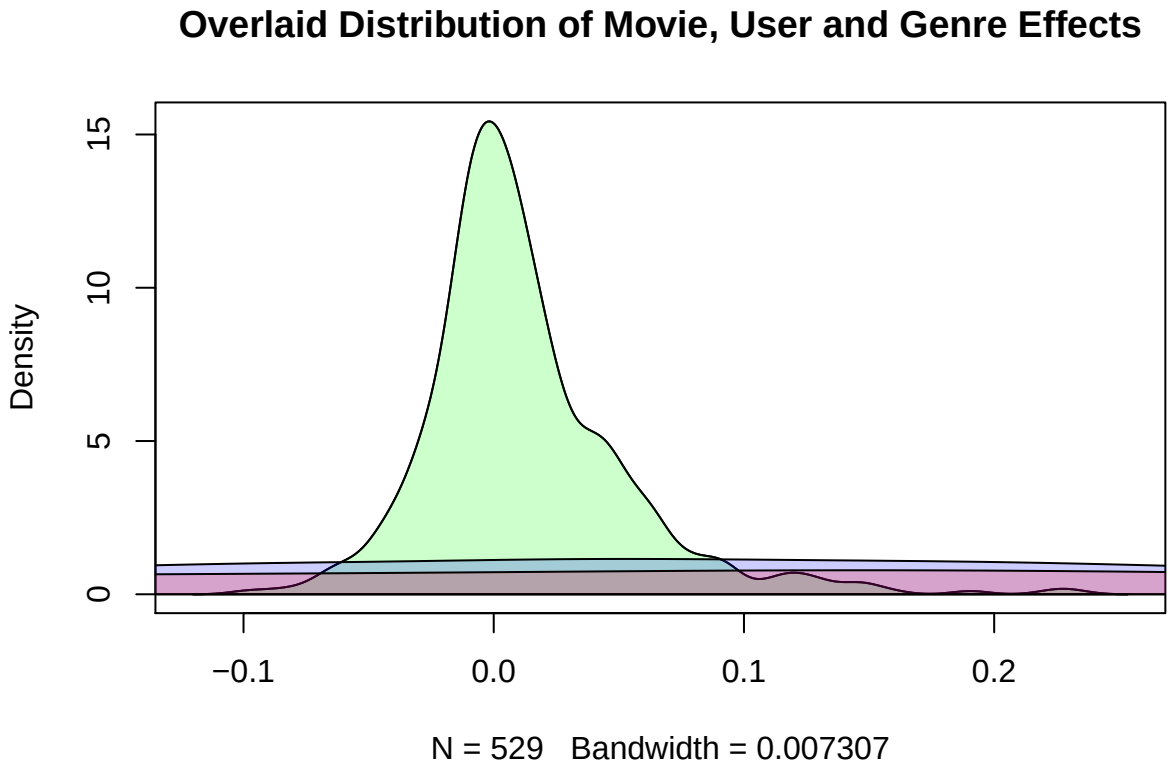
A genre combination effect b_g was computed and added next, using the same technique as for b_i and b_u.

The density of b_g observations was plotted in green and overlaid on the previously plotted densities for b_i and b_u.

```
# Compute genre combination averages.
genre_avgs <- edx_train %>%
  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))

# Preview results.
head(genre_avgs,5)
```

```
## # A tibble: 5 x 2
##   genres                                              b_g
##   <chr>                                             <dbl>
## 1 Action                                          -0.00967
## 2 Action|Adventure                                -0.00251
## 3 Action|Adventure|Animation|Children|Comedy|Sci-Fi  0.0209
## 4 Action|Adventure|Animation|Drama|Fantasy           0.0650
## 5 Action|Adventure|Animation|Drama|Fantasy|Sci-Fi   -0.0119
```

```
# Plot the distribution of b_g.
plot(density(genre_avgs$b_g)
    ,main="Overlaid Distribution of Movie, User and Genre Effects"
    ,col="black")
# Overlay b_g shaded green.
polygon(density(genre_avgs$b_g), col=rgb(0,1,0,0.2))
# Overlay b_u  shaded blue.
polygon(density(user_avgs$b_u), col=rgb(0,0,1,0.2))
# Overlay b_i shaded red.
polygon(density(movie_avgs$b_i), col=rgb(1,0,0,0.2))
```



The distribution of b_g shows most of the means in a much tighter range than either b_i or b_u. There are a relatively small number of genre combinations that have a strong impact on prediction, but b_g is worth keeping in a linear model to represent all genres.

Individual genres were analyzed in the *Decomposition of Genre Strings* sections above. They were not included in the models.

```
# Show extreme values of b_g
max(genre_avgs$b_g)
```

```
## [1] 0.2315427
```

```
min(genre_avgs$b_g)
```

```
## [1] -0.0983333
```

The most influential genre combinations are listed here (with absolute b_g values 0.1 and higher).

```
# Add an absolute value field and sort descending.
# Displaying this does not overwrite the actual averages.
genre_avgs %>% mutate(abs_g=abs(b_g)) %>%
  arrange(desc(abs_g)) %>%
  filter(abs_g >= 0.1)
```

```
## # A tibble: 18 x 3
```

```
##    genres                                                 b_g abs_g
##    <chr>                                                <dbl> <dbl>
##  1 Comedy|Romance|Thriller                              0.232 0.232
##  2 Comedy|Crime|Drama|Romance                           0.224 0.224
##  3 Thriller|War                                         0.190 0.190
##  4 Romance|Sci-Fi                                       0.158 0.158
##  5 Documentary|Drama|Musical                            0.148 0.148
##  6 Adventure|Animation|Children|Comedy|Fantasy|Musical 0.146 0.146
##  7 Adventure|Children|Drama|Fantasy                     0.146 0.146
##  8 Comedy|Drama|Mystery|Thriller                        0.136 0.136
##  9 Comedy|Crime|Mystery|Thriller                        0.130 0.130
## 10 Action|Crime|Drama|War                               0.129 0.129
## 11 Action|Comedy|Thriller|Western                       0.125 0.125
## 12 Children|Fantasy                                     0.122 0.122
## 13 Action|Crime|Fantasy                                 0.121 0.121
## 14 Adventure|Animation|Children|Comedy|Sci-Fi            0.117 0.117
## 15 Drama|Horror|Romance                                 0.117 0.117
## 16 Documentary|Drama|Romance|War                        0.113 0.113
## 17 Horror|Western                                       0.112 0.112
## 18 Action|Adventure|Comedy|Drama|War                    0.105 0.105
```

Predictions made with a model using mu, b_i, b_u and b_g did not improve RMSE significantly. This is in line with the relatively low number of influential genre combinations, and this being the 3rd feature added on top of the mean.

It is important to remember that RMSE is an overall measure. These results do not negate the impact that b_g may have on some specific movies as a standalone predictor.

Perhaps future refinement of genres (such as in the *Decomposition of Genre Strings* section above) can produce a more meaningful predictor. This is out of the current scope.

```
# Join all and compute RMSE.
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(genre_avgs, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull (pred)

# Evaluate new model.
(rmse_mu_bi_bu_bg <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 0.8484796
```

```
# Compare with previous models.
rmse_mu_bi_bu_bg
```

```
## [1] 0.8484796
```

```
rmse_mu_bi_bu
```

```
## [1] 0.8485752
```

```
rmse_mu_bi
```

```
## [1] 0.9285891
```

```
rmse_mu
```

```
## [1] 1.052902
```

Movie Release Year Effect: b_r

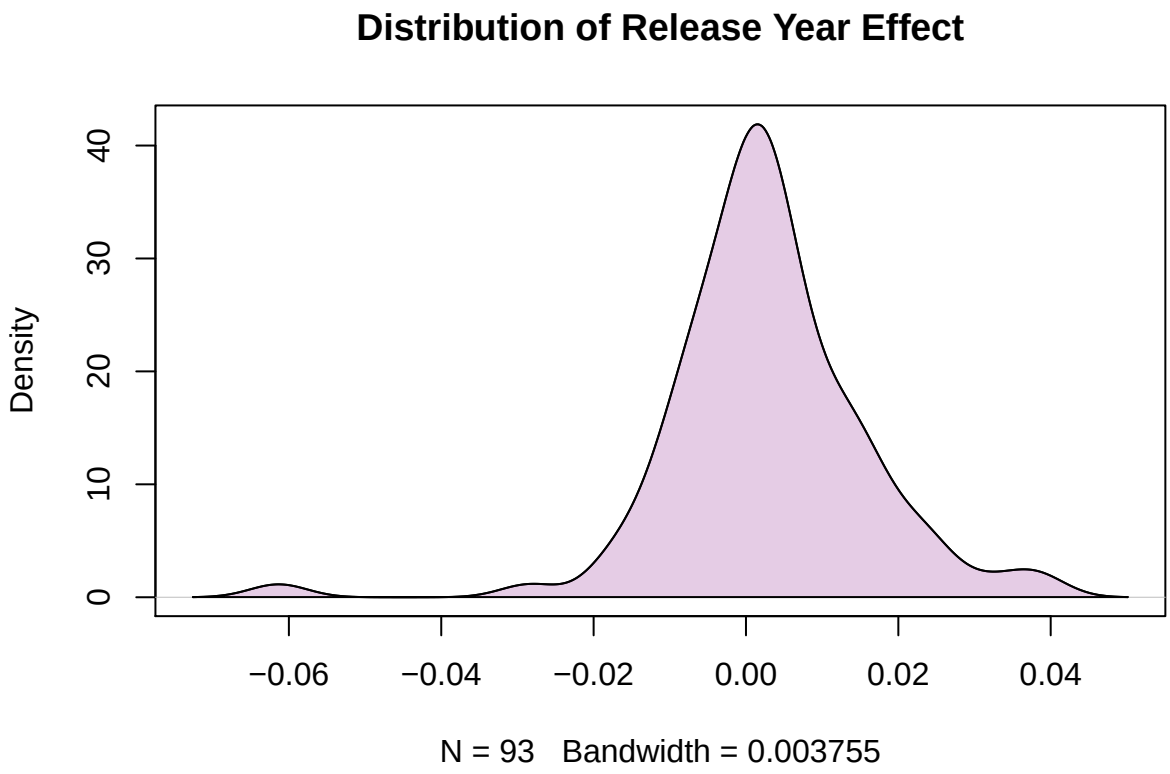
An effect for movie release year, as discussed in the *Movie Release Year* section above, was added. Popular memes or a gestalt of the release years could influence ratings.

The same methods as above were used in computation, plotting, and evaluation of the iterated model, adding b_r.

```
# Compute release year rating averages
release_avgs <- edx_train %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  left_join(genre_avgs, by="genres") %>%
  group_by(year) %>%
  summarise(b_r = mean(rating-mu-b_i-b_u-b_g))

# Preview results.
head(release_avgs,5)
```

```
## # A tibble: 5 x 2
##   year       b_r
##   <chr>    <dbl>
## 1 1917   0.0187
## 2 1918   0.0365
## 3 1919   0.00646
## 4 1920  -0.0111
## 5 1921  -0.0284
```

```
# Plot the distribution of b_r.
plot(density(release_avgs$b_r)
    ,main="Distribution of Release Year Effect"
    ,col="black")
# Overlay b_r shaded in purple.
polygon(density(release_avgs$b_r), col=rgb(0.5,0,0.5,0.2))
```



Most of the b_r values are concentrated near 0, but a significant few appear at minor peaks. These all appear to be movies from before 1930. The highest absolute values of b_r can make a difference in a rating model.

```
# Add an absolute value field and sort descending.
# Show the most influential release years.
release_avgs %>% mutate(ab_r=abs(b_r)) %>%
  arrange(desc(ab_r)) %>%
  filter(ab_r >= 0.06)
```

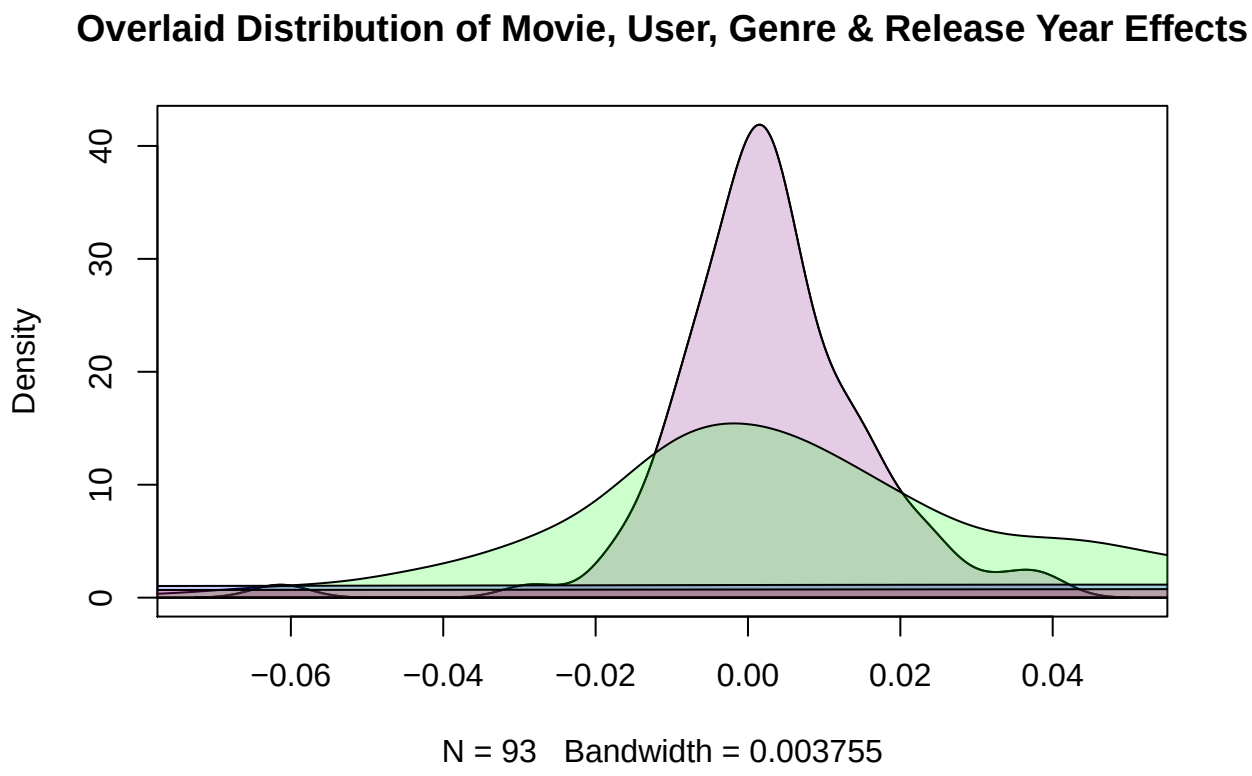
```
## # A tibble: 1 x 3
##   year      b_r   ab_r
##   <chr>   <dbl>  <dbl>
## 1 1922  -0.0613 0.0613
```

The movie release year shows the highest concentrated peak yet, of all predictors. Except for the noted exceptions above, the values of b_r are densely populated around 0, rendering additional benefit of adding release year marginal for most predictions.

However, it is worth observing that this is the release year being *added to the model* as opposed to using the release year alone.

```
# Plot the distribution of b_r.
plot(density(release_avgs$b_r)
    ,main="Overlaid Distribution of Movie, User, Genre & Release Year Effects"
    ,col="black")
```

```
# Overlay b_r shaded in purple.
polygon(density(release_avgs$b_r), col=rgb(0.5,0,0.5,0.2))
# Overlay b_g shaded green.
polygon(density(genre_avgs$b_g), col=rgb(0,1,0,0.2))
# Overlay b_u  shaded blue.
polygon(density(user_avgs$b_u), col=rgb(0,0,1,0.2))
# Overlay b_i shaded red.
polygon(density(movie_avgs$b_i), col=rgb(1,0,0,0.2))
```



As expected from the analysis, adding the release year did not improve the model. It is kept in the model going forward, for further comparison with other predictors and for later use in a regression tree model.

```
# Join with predictor averages and predict.
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by= 'movieId') %>%
  left_join(user_avgs, by= 'userId') %>%
  left_join(genre_avgs, by= 'genres') %>%
  left_join(release_avgs, by= 'year') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_r) %>%
  pull (pred)

# Evaluate prediction and compare with previous iterations.
(rmse_mu_bi_bu_bg_br <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 0.8484213
```

```
rmse_mu_bi_bu_bg
```

```
## [1] 0.8484796
```

```
rmse_mu_bi_bu
```

```
## [1] 0.8485752
```

```
rmse_mu_bi
```

```
## [1] 0.9285891
```

```
rmse_mu
```

```
## [1] 1.052902
```

Rating Lag Effect: b_lag

As examined in a the *Rating Lag Indicator* section above, a "whole year" integer difference between release year and rating year is used to proxy this lag.

```
# Compute rating averages by lag.
lag_avgs <- edx_train %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  left_join(genre_avgs, by="genres") %>%
  left_join(release_avgs, by="year") %>%
  group_by(ratinglag) %>%
  summarise(b_lag = mean(rating - mu- b_i - b_u - b_g - b_r))
```

The mean and standard deviation indicate a strong concentration of b_lag around 0. This could be due to user tendencies to rate more recently released movies.

```
mean(lag_avgs$b_lag)
```

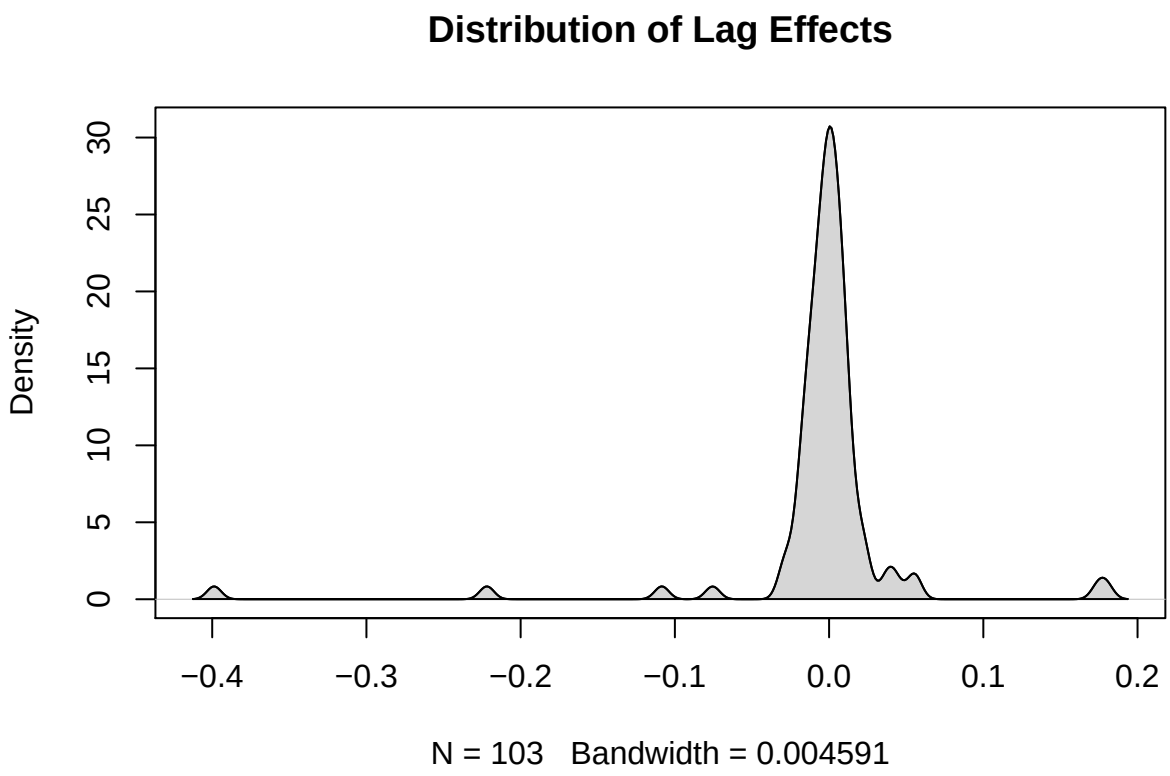
```
## [1] -0.003281321
```

```
sd(lag_avgs$b_lag)
```

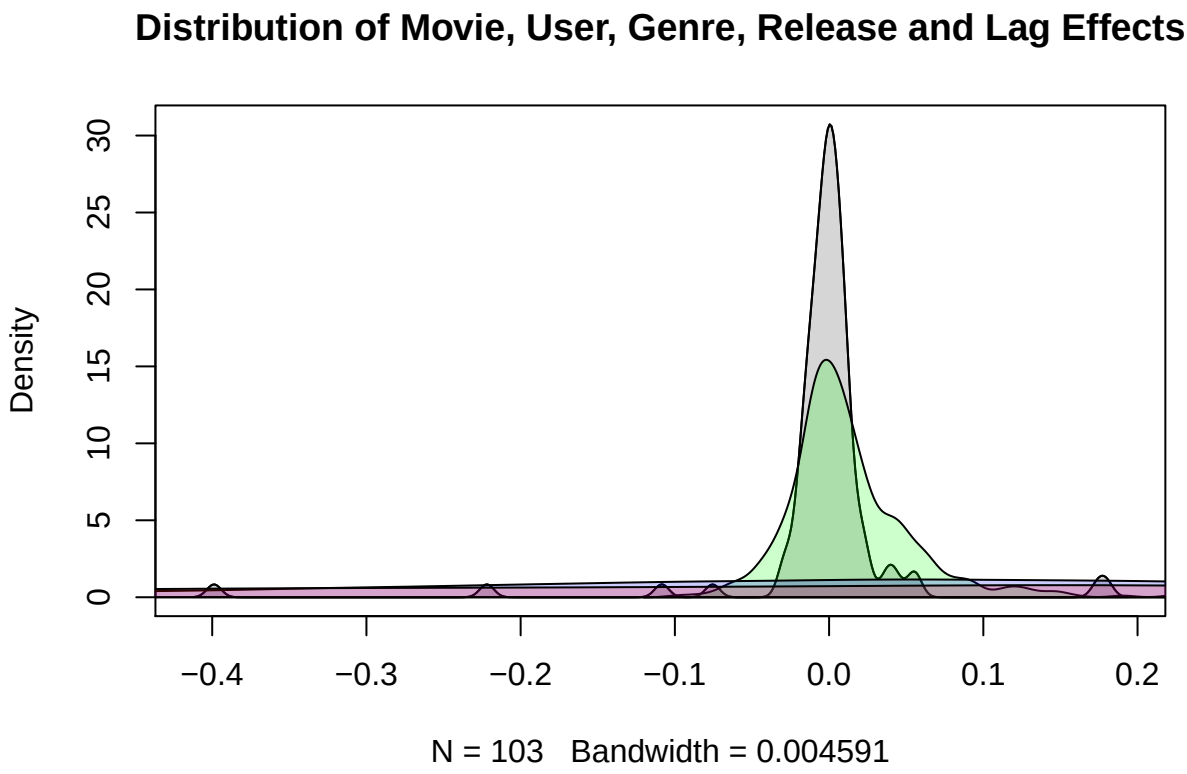
```
## [1] 0.05512243
```

A density plot of b_lag shows the majority of ratings concentrate near 0, with some interesting minor density peaks. There are a few ratings by lag effect that adjust a linear prediction by up to -0.4. The range of b_lag makes it an interesting predictor.

```
# Plot the distribution of b_lag.
plot(density(lag_avgs$b_lag)
    ,main="Distribution of Lag Effects"
    ,col="black")
# Overlay b_lag shaded in light grey.
polygon(density(lag_avgs$b_lag), col=rgb(0.2,0.2,0.2,0.2))
```




```
# Overlay plot with other predictors.
plot(density(lag_avgs$b_lag)
    ,main="Distribution of Movie, User, Genre, Release and Lag Effects"
    ,col="black")
# Overlay b_lag shaded in light grey.
polygon(density(lag_avgs$b_lag), col=rgb(0.2,0.2,0.2,0.2))
# Overlay b_g shaded green.
polygon(density(genre_avgs$b_g), col=rgb(0,1,0,0.2))
# Overlay b_u  shaded blue.
polygon(density(user_avgs$b_u), col=rgb(0,0,1,0.2))
# Overlay b_i shaded red.
polygon(density(movie_avgs$b_i), col=rgb(1,0,0,0.2))
```



A revised RMSE was calculated, adding b_lag as a predictor. No improvement was achieved.

```
# Join to predictor tables and predict, adding b_lag.
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by= 'movieId') %>%
  left_join(user_avgs, by= 'userId') %>%
  left_join(genre_avgs, by= 'genres') %>%
  left_join(release_avgs, by= 'year') %>%
  left_join(lag_avgs, by= 'ratinglag') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_r + b_lag) %>%
  pull (pred)

# Evaluate prediction and compare with previous iterations.
(rmse_mu_bi_bu_bg_br_blag <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 0.848384
```

```
rmse_mu_bi_bu_bg_br
```

```
## [1] 0.8484213
```

```
rmse_mu_bi_bu_bg
```

```
## [1] 0.8484796
```

```
rmse_mu_bi_bu
```

```
## [1] 0.8485752
```

```
rmse_mu_bi
```

```
## [1] 0.9285891
```

```
rmse_mu
```

```
## [1] 1.052902
```

Rating Lag Standalone Predictor: b_laga

Although the major peak for b_lag is near zero, the several peaks at various b_g values invited curiosity to measure a RMSE using b_lag alone. To avoid confusion, a b_laga variable was derived and not used in the main model.

```
# Compute b_laga using overall mean and rating lag.
# Call this b_ua to avoid confusion.
lag_avgs_alone <- edx_train  %>%
  group_by(ratinglag) %>%
  summarize(b_laga = mean(rating-mu))

# Compute new predictions with only mu and b_aga.
predicted_ratings <- edx_test %>%
  left_join(lag_avgs_alone, by= 'ratinglag') %>%
  mutate(pred = mu + b_laga) %>%
  pull (pred)
```

Using only rating lag and the overall average rating, a mild improvement is made over using the overall mean alone. Rating lag is kept in the model to avoid losing this benefit. It could be relevant among some particular movies, with the effect being diluted when RMSE is calculated across all movies.

```
# Compute RMSE.
(rmse_mu_blaga <- RMSE(edx_test$rating, predicted_ratings))
```

```
## [1] 1.051964
```

```
# compare with mu alone.
rmse_mu
```

```
## [1] 1.052902
```

Towards Tree Models

After finding it difficult to improve the linear model any more, a regression tree model was attempted, using the same training set. Other experiments were tried but not included in this report. Some findings not detailed in the main of this report, but worth mentioning are:

1. Random forest models were attempted, with varying sample sizes (from 30,000 to 200,000), number of trees (between 500 and 1000) and final node sizes (between 10 and 70). These did not achieve significantly better RMSE scores versus the linear model. Most of these crashed. When the linear predictions were included as an engineered feature, the RF models appeared to detract from the accuracy of the prediction.

2. With regression trees using the CART library, over several trials, the "genres" column was found to crash fit runs. It was therefore dropped from the regression tree models below.

A regression tree model (rather than a classification tree model) was chosen because rating is a quantitative value, not strictly a label. While ratings could indeed be rounded to the half-step values between 0.5 and 5.0 available to users and treated as ordered categories, decimal place predictions were allowed. Since the model is about predicting a user's probable rating to suggest or not suggest movies to users, this is and acceptable concept

However, the userId and movieId fields were stored as numeric values in the the original data. They represent labels for individual users and movies, so they were converted to factors. Otherwise, regression trees could mistake them for quantities and partition based on the "quantity value" of movieIds and userIds, providing benefit only by incidence or poorly understood relationship (ex. member number orders according to time or location of registration).

Standalone Regression Tree Model

Since userId and movieId were determined to be the most influential factors in above sections, the first regression tree uses only these two predictors to establish a baseline.

Training and test sets were prepared for the tree and ensemble models.

```
# Invoke trees librarie.
library(rpart)

# Transform userId and movieId fields into factors in training and test
#  Then, join back to averages tables to acquire linear effects.
#  Then, compute all-in linear prediction (pred)
#   linear prediction without genres effect (linearPred).
#   and error from rating (linearError).
edx_train_trees <- edx_train %>%
                mutate(year=str_sub(title,-5,-2)
               , ratingdate = as.POSIXct(`timestamp`, origin = "1970-01-01")
               , ratingyear = format(ratingdate,'%Y')
               , ratinglag = as.numeric(ratingyear)-as.numeric(year)
               , factorUserId = as.factor(userId)
               , factorMovieId = as.factor(movieId)) %>%
                  left_join(movie_avgs, by= 'movieId') %>%
                  left_join(user_avgs, by= 'userId') %>%
                  left_join(genre_avgs, by= 'genres') %>%
                  left_join(release_avgs, by= 'year') %>%
                  left_join(lag_avgs, by= 'ratinglag') %>%
                       mutate(pred = mu + b_i + b_u + b_g + b_r + b_lag
                           ,linearPred = mu + b_i + b_u + b_r + b_lag
                           ,linearError = rating-linearPred)

# Do the same for test.
edx_test_trees <- edx_test %>%
```

```
                  mutate(year=str_sub(title,-5,-2)
                , ratingdate = as.POSIXct(`timestamp`, origin = "1970-01-01")
                , ratingyear = format(ratingdate,'%Y')
                , ratinglag = as.numeric(ratingyear)-as.numeric(year)
                , factorUserId = as.factor(userId)
                , factorMovieId = as.factor(movieId)) %>%
                   left_join(movie_avgs, by= 'movieId') %>%
                   left_join(user_avgs, by= 'userId') %>%
                   left_join(genre_avgs, by= 'genres') %>%
                   left_join(release_avgs, by= 'year') %>%
                   left_join(lag_avgs, by= 'ratinglag') %>%
                        mutate(pred = mu + b_i + b_u + b_g + b_r + b_lag
                            ,linearPred = mu + b_i + b_u + b_r + b_lag
                            ,linearError = rating-linearPred)

# Do the same for Validation
validation_temp <- validation %>%
                  mutate(year=str_sub(title,-5,-2)
                , ratingdate = as.POSIXct(`timestamp`, origin = "1970-01-01")
                , ratingyear = format(ratingdate,'%Y')
                , ratinglag = as.numeric(ratingyear)-as.numeric(year)
                , factorUserId = as.factor(userId)
                , factorMovieId = as.factor(movieId)) %>%
                   left_join(movie_avgs, by= 'movieId') %>%
                   left_join(user_avgs, by= 'userId') %>%
                   left_join(genre_avgs, by= 'genres') %>%
                   left_join(release_avgs, by= 'year') %>%
                   left_join(lag_avgs, by= 'ratinglag') %>%
                        mutate(pred = mu + b_i + b_u + b_g + b_r + b_lag
                            ,linearPred = mu + b_i + b_u + b_r + b_lag
                            ,linearError = rating-linearPred)


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

# Add rows removed from validation set back into edx_train_trees set
removed <- anti_join(validation_temp, validation_extended)
edx_train_trees <- rbind(edx_train_trees, removed)
rm(validation_temp, removed)
```

A baseline regression tree was built using only movie and user identifiers.

```
# Build new model using factor user and factor movie.
fit_tree_u_i <- rpart(rating ~ factorUserId + factorMovieId,
                method="anova", data=edx_train_trees)

# Predict using tree model.
predictions <- predict(fit_tree_u_i, newdata = edx_test_trees)

# Compute and store RMSE.
```

```
(rmse_tree_u_i <- RMSE(edx_test_trees$rating, predictions))
```

```
## [1] 0.8897131
```

The resulting RMSE does not surmount that of the linear model.

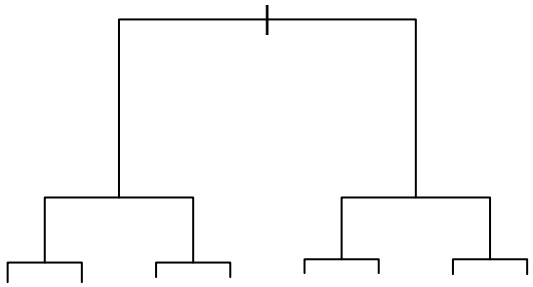
The default of 10-fold cross-validation was implied in the model:

1. The decision tree indeed found both movie and user factors useful. There are 8 possible predictions.
2. The basic levels of decision are drawn, but there are too many users and movie rules to display labels.
3. The number of splits (up to 7) tended to reduce the error in predictions.
4. As the complexity parameter (required RSS improvement per split) increased, the relative error decreased, to a cp value of 0.011.

```
# Print cross validation  analysis.
printcp(fit_tree_u_i)
```

```
##
## Regression tree:
## rpart(formula = rating ~ factorUserId + factorMovieId, data = edx_train_trees,
##     method = "anova")
##
## Variables actually used in tree construction:
## [1] factorMovieId factorUserId
##
## Root node error: 6292602/5660340 = 1.1117
##
## n= 5660340
##
##         CP nsplit rel error  xerror       xstd
## 1 0.146488      0   1.00000 1.00000 0.00059536
## 2 0.053546      1   0.85351 0.85512 0.00053514
## 3 0.050847      2   0.79997 0.80999 0.00051970
## 4 0.016118      3   0.74912 0.76698 0.00049960
## 5 0.012305      4   0.73300 0.75116 0.00049560
## 6 0.012000      5   0.72070 0.73709 0.00049184
## 7 0.011380      6   0.70870 0.72714 0.00048796
## 8 0.010000      7   0.69732 0.72071 0.00048344
```

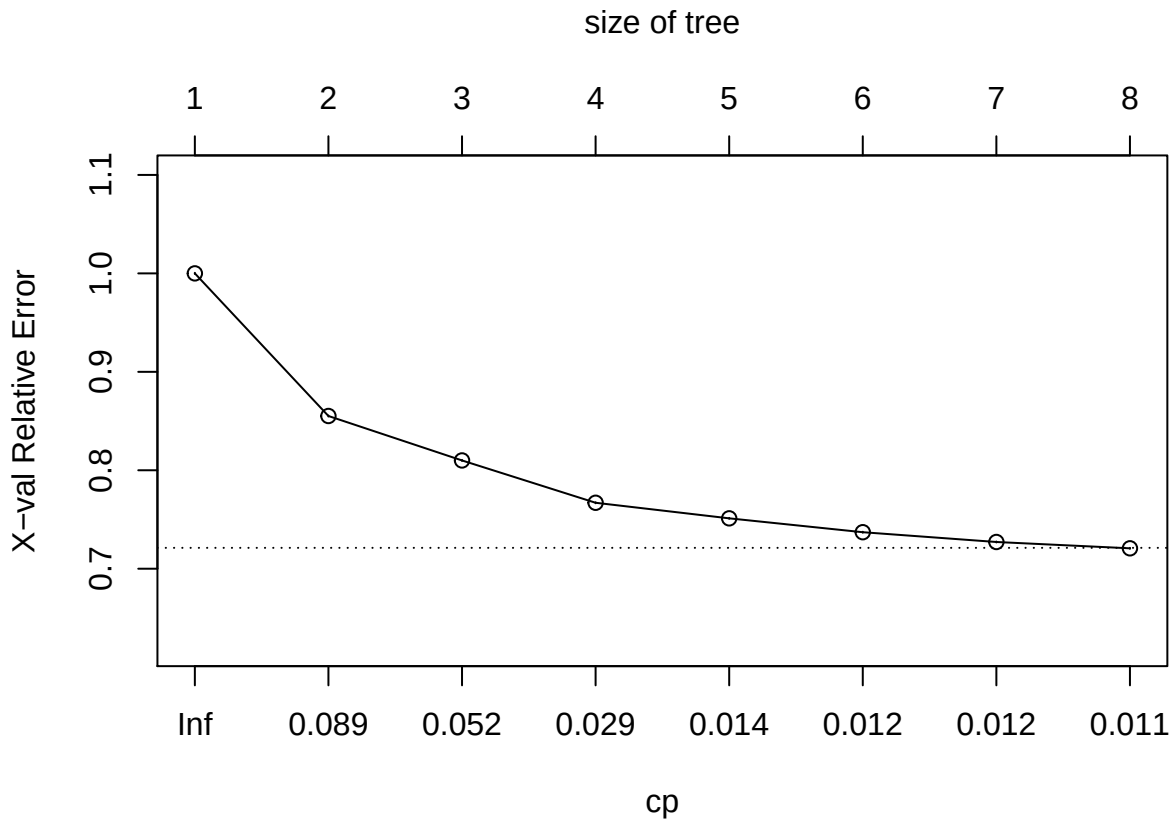
```
# Draw the decision tree structure.
plot(fit_tree_u_i, margin=0.4)
```



```
# Show possible prediction outputs.
unique(predictions)
```

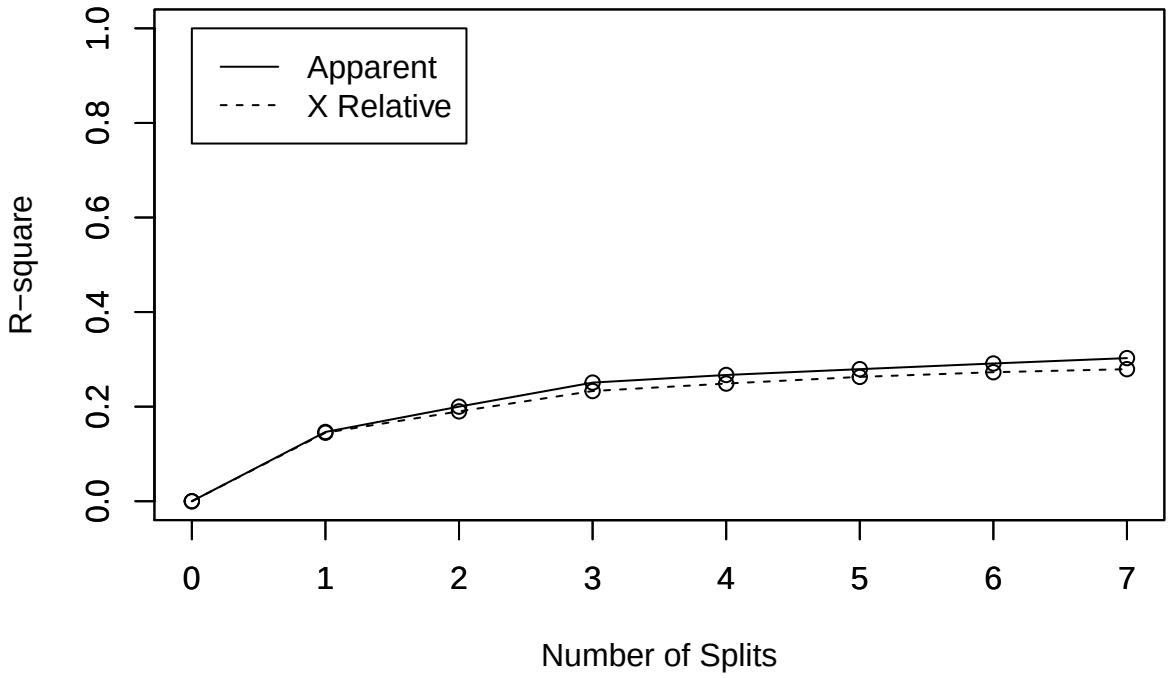
```
## [1] 3.147473 3.561227 4.249493 3.861359 2.727929 2.017540 3.716340 3.045620
```

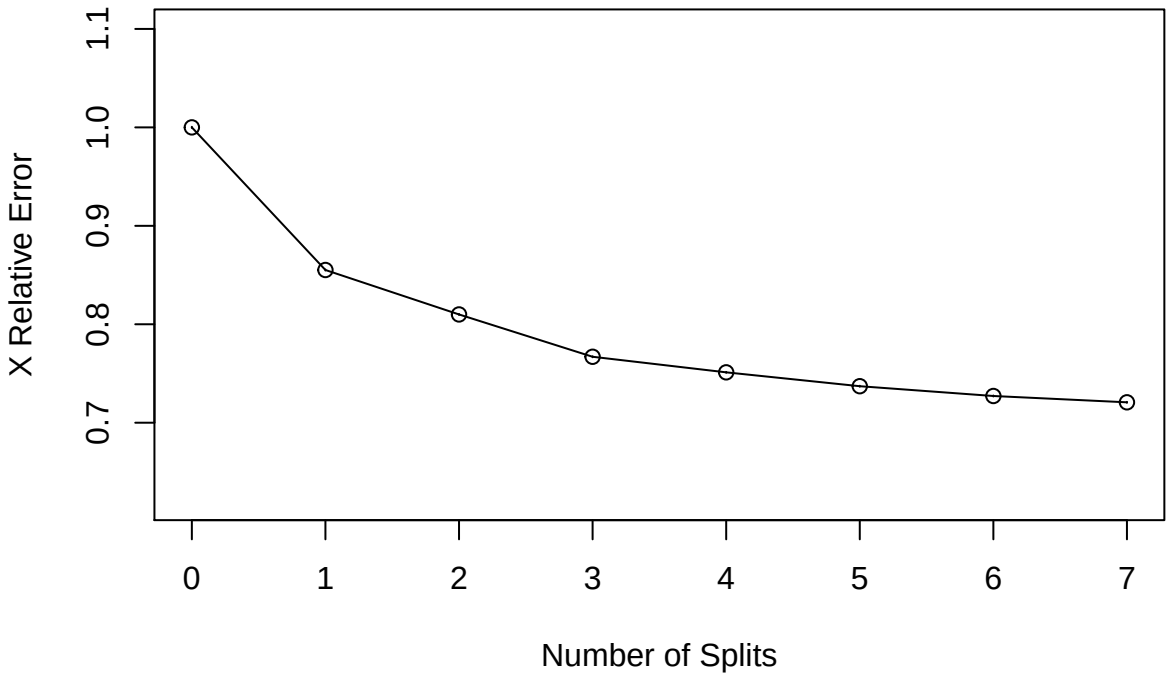
```
# Plot cross-validated complexity factor.
plotcp(fit_tree_u_i)
```




```
# Plot R-square by number of splits.
rsq.rpart(fit_tree_u_i)
```

```
##
## Regression tree:
## rpart(formula = rating ~ factorUserId + factorMovieId, data = edx_train_trees,
##     method = "anova")
##
## Variables actually used in tree construction:
## [1] factorMovieId factorUserId
##
## Root node error: 6292602/5660340 = 1.1117
##
## n= 5660340
##
##         CP nsplit rel error  xerror       xstd
## 1 0.146488      0   1.00000 1.00000 0.00059536
## 2 0.053546      1   0.85351 0.85512 0.00053514
## 3 0.050847      2   0.79997 0.80999 0.00051970
## 4 0.016118      3   0.74912 0.76698 0.00049960
## 5 0.012305      4   0.73300 0.75116 0.00049560
## 6 0.012000      5   0.72070 0.73709 0.00049184
## 7 0.011380      6   0.70870 0.72714 0.00048796
## 8 0.010000      7   0.69732 0.72071 0.00048344
```


Apparent
X Relative
R−square
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
Number of Splits



Next, all other predictors that were used in the linear model (except genres) were tried with the regression tree, but this resulted in an even worse outcome. The RMSE grew, meaning the model became less useful. This run is not shown here.

An ensemble was attempted next.

Ensemble: Using Linear Predictions as a Regression Tree Inputs.

It was hypothesized a regression tree could be used to refine linear predictions. Since the best regression tree RMSE was produced using only the factor userId and factor movieId fields, they are combined with the linear prediction as features.

```
# Build tree model usiog user, movie and linear prediction.
fit_ensemble_1 <- rpart(rating ~ factorUserId + factorMovieId  + linearPred
                        , method="anova", data=edx_train_trees)

# Predict using tree model.
predictions <- predict(fit_ensemble_1, newdata = edx_test_trees)

# Compute and store RMSE.
(rmse_ensemble_1 <- RMSE(edx_test_trees$rating, predictions))
```

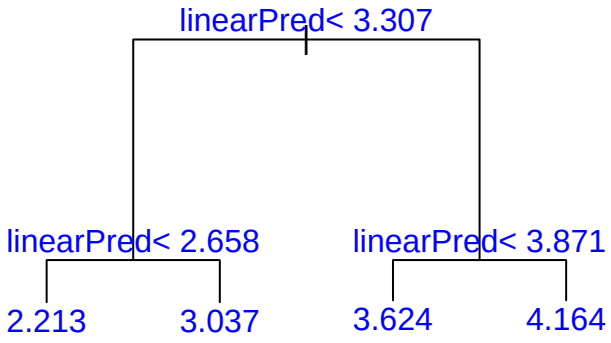
```
## [1] 0.8713684
```

```
# Build new model, adding rating date.
fit_ensemble_2 <- rpart(rating ~ factorUserId + factorMovieId  + ratingdate + linearPred
                        , method="anova", data=edx_train_trees)

# Predict using tree model.
predictions <- predict(fit_ensemble_2, newdata = edx_test_trees)

# Compute and store RMSE.
(rmse_ensemble_2 <- RMSE(edx_test_trees$rating, predictions))
```

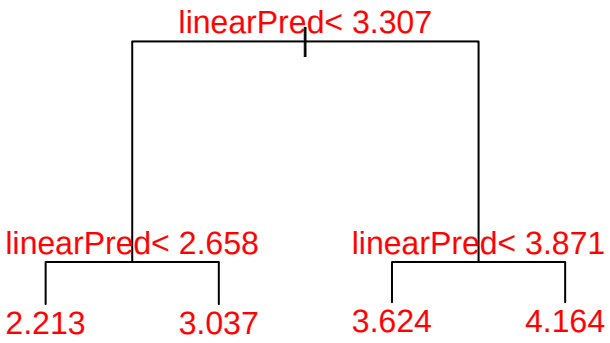
```
## [1] 0.8713684
```

Adding rating date did not change the outcome at all. In fact, the resulting decision trees are exactly the same in both models: Both ensemble trees used the linear prediction as the only relevant measure for decisions.

```
# Show the first ensemble.
plot(fit_ensemble_1, margin=0.4)
text(fit_ensemble_1, cex = 1, col="blue")
```




```
# Show the second ensemble.
plot(fit_ensemble_2, margin=0.4)
text(fit_ensemble_2, cex = 1, col="red")
```



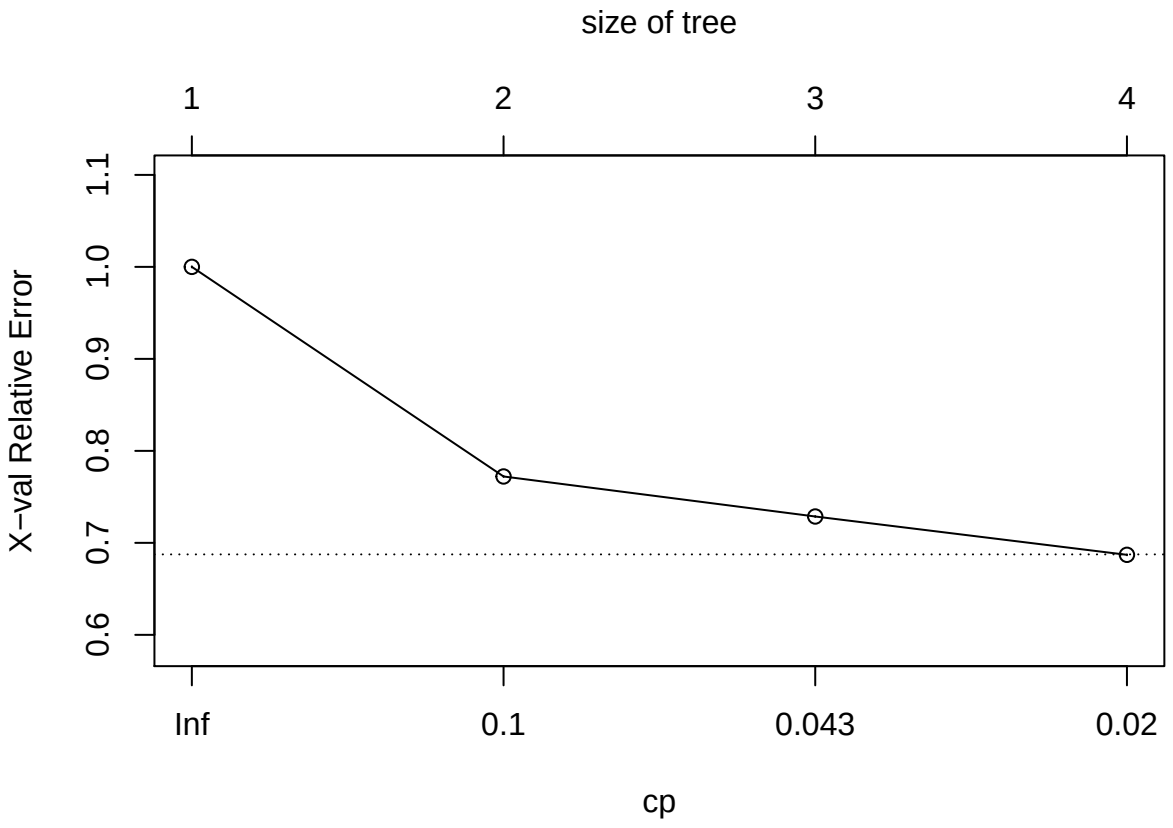
For the 2nd ensemble model:

1. All predictions by the model are one of the four listed in the tree.
2. The complexity parameter was reduced to 0.021 by cross-validation.
3. The number of splits were set to 3.

```
# Show unique predictions
predictions %>% unique()
```

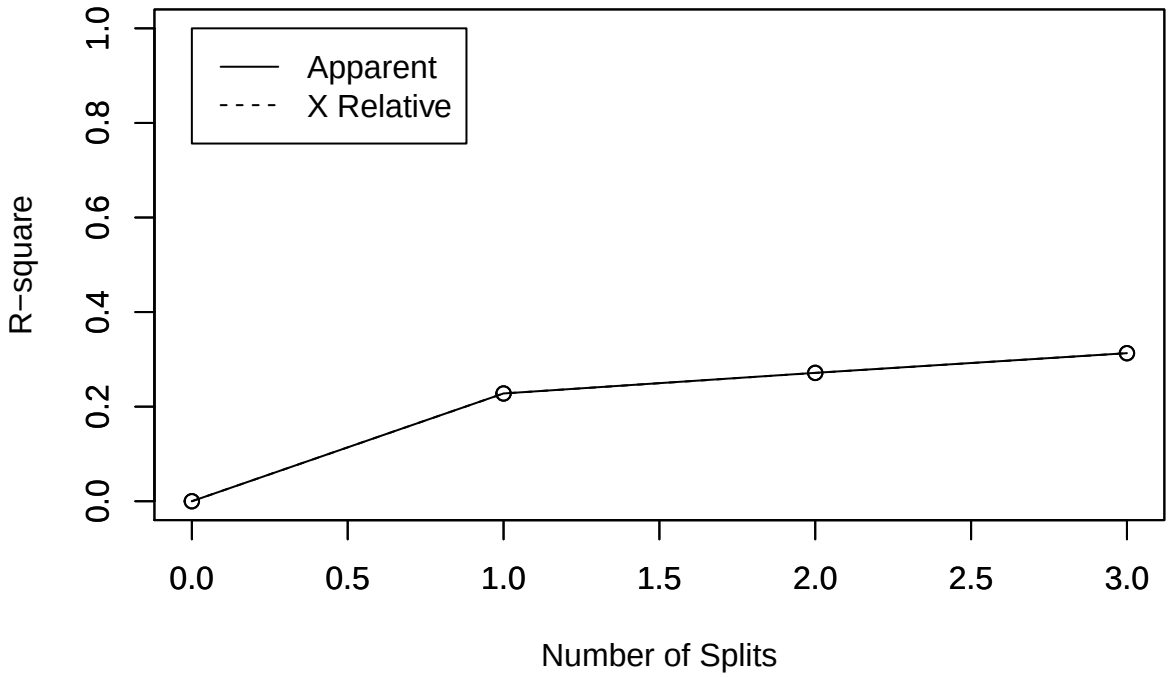
```
## [1] 3.036900 3.624299 4.164409 2.213083
```

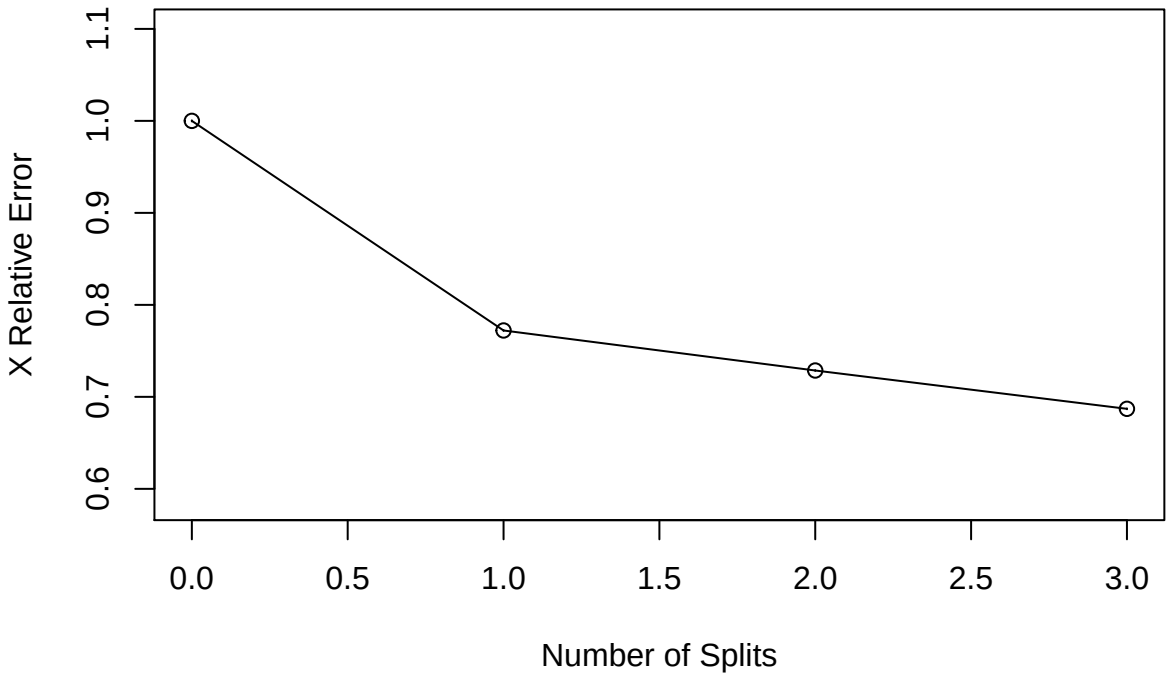
```
# Plot cross-validated complexity factor.
plotcp(fit_ensemble_2)
```




```
# Plot R-square by number of splits.
rsq.rpart(fit_ensemble_2)
```

```
##
## Regression tree:
## rpart(formula = rating ~ factorUserId + factorMovieId + ratingdate +
##     linearPred, data = edx_train_trees, method = "anova")
##
## Variables actually used in tree construction:
## [1] linearPred
##
## Root node error: 6292602/5660340 = 1.1117
##
## n= 5660340
##
##         CP nsplit rel error  xerror       xstd
## 1 0.227923      0   1.00000 1.00000 0.00059536
## 2 0.043574      1   0.77208 0.77215 0.00049355
## 3 0.041616      2   0.72850 0.72865 0.00047890
## 4 0.010000      3   0.68689 0.68700 0.00046768
```


Apparent
X Relative
R−square
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Number of Splits



Summary of Results

Comparison of RMSE scores.

Several RMSE's were calculated while building up the linear model. The process was educational and resulted in a significantly predictive model. Note that some predictors may have shown higher RMSE improvement increments if they were added earlier than the order chosen. The later an effect is added, the less improvement increment it contributes to the overall RMSE. This is partly because the remaining error (between prediction and actual rating) becomes smaller and smaller with each added effect. Some testing of this effect was done, showing that the movie effect was stronger than the user effect overall, if taken separately.

In constrast to building up the linear model manually, the regression tree and ensemble models were built using R packages pre-built to generate models. Exploration of the generated models presented opportunities for future model enhancement.

In the end, the linear model generated the best results. The RMSE scores generated from test data are ranked below. Only the lowest RMSE model was tested against the validation dataset.

```
# Create a vector of intermediary RMSE results, using just the test set.
RMSE_Test_Scores <- c(  rmse_mu_bi_bu_bg_br_blag
                        ,rmse_mu_bi_bu_bg_br
                        ,rmse_mu_bi_bu_bg
                        ,rmse_mu_bi_bu
                        ,rmse_mu_bi
                        ,rmse_mu
```

```
                          ,rmse_tree_u_i
                          ,rmse_ensemble_1
                          ,rmse_ensemble_2) %>% round(digits=5)


# Create a vector of descriptions.
Model_Description <- c(   "Mean, Movie, User, Genres, Release Year, Release Lag"
                      ,"Mean, Movie, User, Genres, Release Year"
                      ,"Mean, Movie, User, Genres"
                      ,"Mean, Movie, User"
                      ,"Mean, Movie"
                      ,"Mean"
                      ,"Regression Tree with userId & movieId"
                      ,"Ensemble 1: Regression Tree with Movie, User, Linear Prediction"
                      ,"Ensemble 2: Regression Tree with Movie, User, Rating Date, Linear Prediction"
                       )

# Combine into a tibble, sort from lowest to highest RMSE and display.
cbind(Model_Description,RMSE_Test_Scores) %>%
  data.frame() %>% arrange(RMSE_Test_Scores)
```

```
##                                                                Model_Description
## 1                        Mean, Movie, User, Genres, Release Year, Release Lag
## 2                                   Mean, Movie, User, Genres, Release Year
## 3                                              Mean, Movie, User, Genres
## 4                                                    Mean, Movie, User
## 5              Ensemble 1: Regression Tree with Movie, User, Linear Prediction
## 6 Ensemble 2: Regression Tree with Movie, User, Rating Date, Linear Prediction
## 7                                    Regression Tree with userId & movieId
## 8                                                         Mean, Movie
## 9                                                                Mean
##   RMSE_Test_Scores
## 1          0.84838
## 2          0.84842
## 3          0.84848
## 4          0.84858
## 5          0.87137
## 6          0.87137
## 7          0.88971
## 8          0.92859
## 9           1.0529
```

Validation of Final Model

The derived time columns were added to the validation data set above, to make it usable for final model validation. Deriving these fields is not training and does not sully the validation data. If the validation data set has a different structure from the final model being tested, it would not be possible to validate the model at all. The textbook examples for this course do the exact same type of field derivations.

The validation data has not been used with any other model.

```
# Pull final predictions using reserved validation data.
#   The linear predictionS are already in the pred column of the
#   extended validation set from above calculations,
head(validation_extended) %>% select(rating, pred)
```

```
##    rating     pred
## 1:    3.0 2.708844
## 2:    4.0 3.636825
## 3:    3.0 3.004314
## 4:    4.0 4.027325
## 5:    3.0 4.078808
## 6:    3.5 4.427502
```

```
# Calculate final RMSE.  This is simply validating the actual ratings
#  versus linear preidictions.
(rmse_validation <- RMSE(validation_extended$rating
                         ,validation_extended$pred))
```

```
## [1] 0.8481095
```

Conclusions

The best model was a linear model with effects for users, movies, genres, release years, and rating lags. The most influential features were the specific user and the specific movie.

Future Considerations

Some future considerations:

1. Adding demographic data might help improve predictions.
2. Detailed analysis of rating days, derived from the rating timestamps may reveal that users rate higher on certain days of the week or weekends. Week of year or month of year might also uncover seasonality effects.
3. Another potential ensemble model is to run a random forest first, to produce a classification column that can be used to group data and pull means by that grouping. This could be used as a predictor in a linear model.