Movie Recommendation Capstone Project

Ryan J. Cooper

10/1/2020

Contents

Introduction

This project was developed for the HarvardX Data Science Professional Certificate Program capstone course. The objective of this report is to use movie ratings data to build a recommendation system which will try to predict what rating a user would give any given movie that they have not yet rated.

This process of guessing unknown combinations is generally termed *matrix completion*, and the motivation for this project was the *Netflix Challenge*, a $1 Million prize which was offered to the team with the best score on a movie-ratings based matrix completion challenge. This is a fundamental problem in machine learning, to complete a sparse matrix of data.

This report has been divided into several sections beginning with initial Data Setup. In the Analysis & Methods section, exploratory data analysis (EDA) is conducted on the MovieLens data set, a table of movies and user ratings derived from an online movie ratings web site. This large dataset of 10 million movie reviews requires a strategic approach to create an effective, robust, and accurate recommendation system.

In the Model Construction section we will construct, encapsulate, and tune a machine learning model function that will combine the effects we identify. At each step we will assess how much of an impact the effect will have on the predictions, after confounders have been removed. The model will be judged on how effectively it reduces the root mean square error (RMSE) of our predictions vs. the data present in our internal working partitions. The model construction will also produce tuning parameters that we will use in the final model.

Finally, we will optimize and test the model we have constructed against the final validation set and present our final RMSE scores in the Results section. In the Conclusion, we will assess the performance of the model, and discuss possible improvements and next steps.

Load Packages

Several R packages were required to create this report. These packages provide data shaping & wrangling features, visualization enhancements, and other key functions.

```r
# Note: this process could take a couple of minutes
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
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")
if(!require(matrix)) install.packages("matrix", repos = "http://cran.us.r-project.org")
if(!require(reshape2)) install.packages("reshape2", repos = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
```

Data Setup

This section creates the data sets to be used throughout this project. The edx set is the data we have available. The validation set will be used to judge our results, and *must not be used* for any other reason.

```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)

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

fdir <- unzip(dl, "ml-10M100K/movies.dat")
movies <- str_split_fixed(readLines(fdir), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")

# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
#}

#remove some variables that will not be needed
rm(temp)
rm(removed)
rm(ratings)
rm(movielens)
rm(test_index)
rm(fdir)

#copy the edx set in memory in case we need to reset.
edx_backup <- edx
```

Analysis & Methods

The following analysis will help us examine the data we are working with and make decisions on how to provide further visualizations, identify key variables of interest, and discover relationships and trends in the data.

```r
#This will cause the remainder of the file to not use scientific notation
options(scipen = 999)

#table dimensions
dims <-dim(edx)
hdr <- head(edx)
#movies
nr <- nrow(edx)
#movies
nc <- ncol(edx)
#movies
tm <- length(unique(edx$movieId))
#users
tu <- length(unique(edx$userId))
#ratings
tr <- length(unique(edx$rating))
#min date of rating
fr <-min(as.Date(as.POSIXct(edx$timestamp, origin="1970-01-01")))
#max date of rating
lr <- max(as.Date(as.POSIXct(edx$timestamp, origin="1970-01-01")))
#ratings
med <- median(edx$rating)
mu <- mean(edx$rating)
sigma <- sd(edx$rating)

#overview table examining the data set
overview <- data.frame(rows = nr,
                       columns = nc,
                       movies = tm,
                       users = tu,
                       ratings=tr,
                       firstrating=fr,
                       lastrating=lr,
                       mu=mu,
                       med=med,
                       sigma=sigma)

#charts related to the same subjects will be colored the same
palette<-new.env()
palette[["movie"]]<-"darkslateblue"
palette[["user"]]<-"darkcyan"
palette[["rating"]]<-"chartreuse4"
palette[["year"]]<-"red3"
palette[["month"]]<-"darkorange3"
palette[["genre"]]<-"magenta4"
palette[["genreera"]]<-"maroon4"
palette[["genreuser"]]<-"palevioletred3"
palette[["popular"]]<-"darkgreen"
```

```
palette[["count"]]<-"goldenrod3"
palette[["other"]]<-"deeppink3"

overview %>% knitr::kable(label="Overview of EDX Dataset")
```

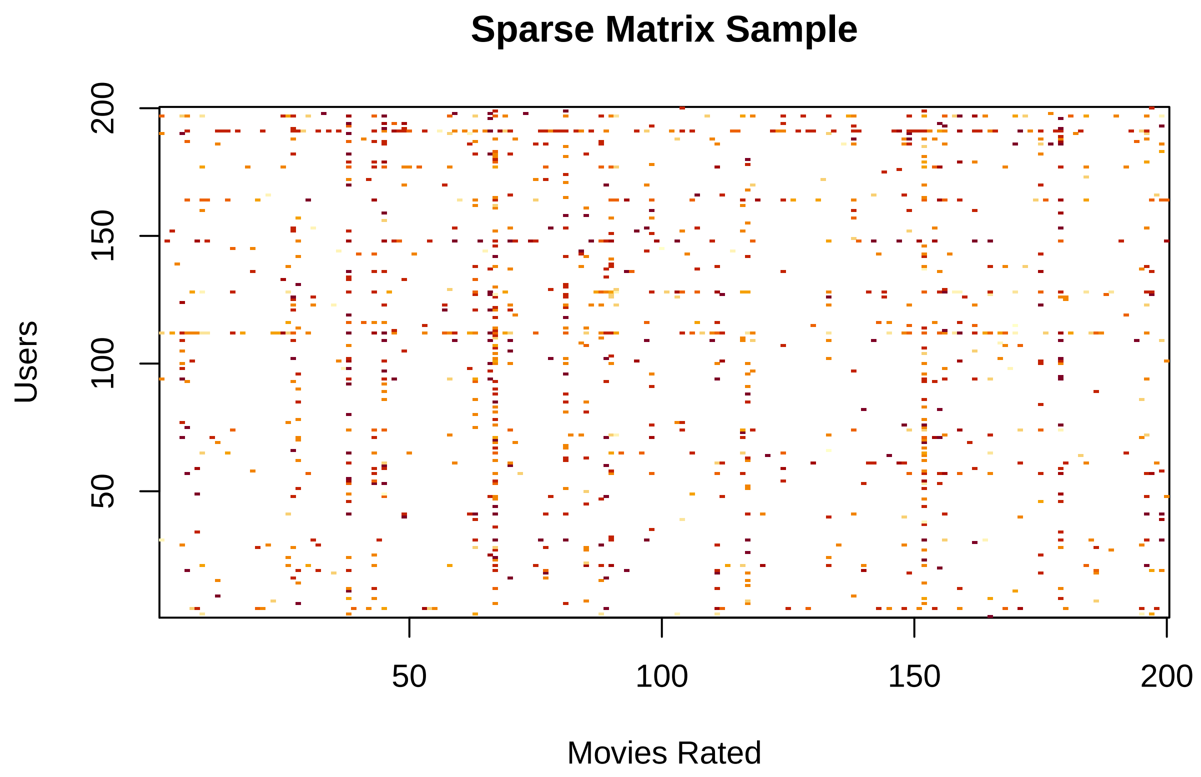
rows	columns	movies	users	ratings	firstrating	lastrating	mu	med	sigma
9000055	6	10677	69878	10	1995-01-09	2009-01-05	3.512465	4	1.060331

We have 6 original columns to work with, including a unique userId, movieId, timestamp, title, genre list, and rating for each movie. There are 69,878 individual users rating 10,677 movies for a total of 9,000,055 rows. There are 10 possible ratings between a .5 star and 5 stars.

As noted in the introduction, a matrix of users and movies rated is relatively *sparse*. Sampling 200 random users and movies, the white space in the chart below is all of the user-movie combinations for which we have no data. This white space is what the model will aim to fill in as accurately as possible.

Just taking this tiny slice of the matrix - it's clear that some users have rated a lot of movies (like where you see a relatively solid horizontal line), and some movies have a lot of ratings (like where you see a relatively solid vertical line) - but the vast majority of the matrix is empty.

```
#sample 200 random users & movies to create a matrix image
set.seed(1, sample.kind="Rounding")
topX <- 200
topXusers <- sample(unique(edx$userId),topX)

#image a sparse matrix
edx %>% filter(userId %in% topXusers) %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), topX)) %>%
  as.matrix() %>%
  t(.) %>%
  image(1:topX, 1:topX, . ,
        main="Sparse Matrix Sample",
        xlab="Movies Rated",
        ylab="Users")
```



Data Terminology

- Review: A row from the edx data containing userId, movieId, rating, rating date, etc.
- Rating: The 1-5 star movie rating given during a review
- User/Reviewer: A user who submitted a review. Users have been anonymized and are referred to by their numeric userId
- Movie/Film: One of the movies that have been reviewed one or more times
- Movie Year: The year the movie was released
- Rate Date: The date that the review / rating was submitted
- Popularity: How many reviews a movie received
- RMSE: Root mean square error The formula that will be used to assess the accuracy of each model can expressed mathematically as:

$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$

This formula will be defined as an R function in the model building section.

Fortify & Preprocess EDX dataset

Several variables can be ascertained by splitting certain fields into component values. In the code below we will extract several values related to the date of the review and the release date of each movie.

```
#reformat the date
edx_fortify <- edx
edx_fortify <- edx_fortify %>%
  mutate(ratedate = as.Date(as.POSIXct(timestamp, origin="1970-01-01")))
edx_fortify <- edx_fortify %>%
  mutate(ratept = as.POSIXct(timestamp, origin="1970-01-01"))
edx_fortify <- edx_fortify %>%
  mutate(rateyear = format(ratedate, "%Y"))
edx_fortify <- edx_fortify %>%
  mutate(ratemonth = format(ratedate, "%m"))

#convert the year in the title to a usable year variable and a title without the year
edx_fortify <- edx_fortify %>%
  mutate(movieyear = as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))")))
edx_fortify <- edx_fortify %>%
  mutate(titlenoyear = as.character(str_remove(title,"\\s\\(\\d{4}\\)")))

#check the results
edx_fortify %>%
  slice(1:10) %>%
  select(movieyear, titlenoyear, ratemonth, rateyear) %>%
  knitr::kable(label="Fortified EDX Data")
```

movieyear	titlenoyear	ratemonth	rateyear
1992	Boomerang	08	1996
1995	Net, The	08	1996
1995	Outbreak	08	1996
1994	Stargate	08	1996
1994	Star Trek: Generations	08	1996
1994	Flintstones, The	08	1996
1994	Forrest Gump	08	1996
1994	Jungle Book, The	08	1996
1994	Lion King, The	08	1996
1994	Naked Gun 33 1/3: The Final Insult	08	1996

We now have added columns available for the movie year, title without year, date of the rating as a date data type, plus several rating date components isolated.

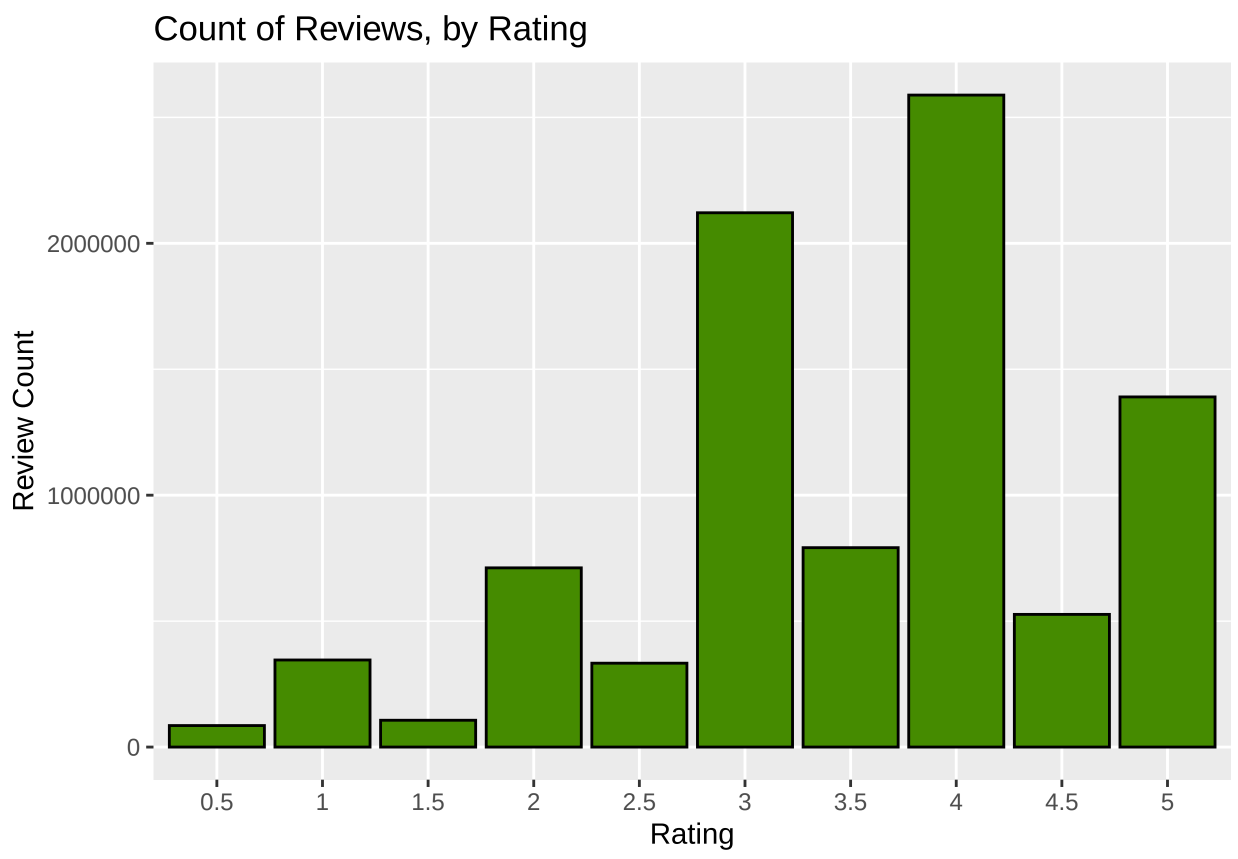
Exploratory Data Analysis

In this section we will examine the data set in its entirety to gain valuable insights about the data before we attempt to construct a linear regression model. We are looking for positive or negative correlations and candidate data points. During the course of analysis, we may look at either raw, uncorrected averages, or in some cases we will look at the residual averages, correcting for confounders by subtracting a predicted rating (based on certain biases) from the actual ratings that were given. We will create strata to facet and group data points, to examine correlations across multiple factors. We will use histograms, line charts, scatter plots and other visualization types to help justify each factor included in the final model.

Ratings Analysis

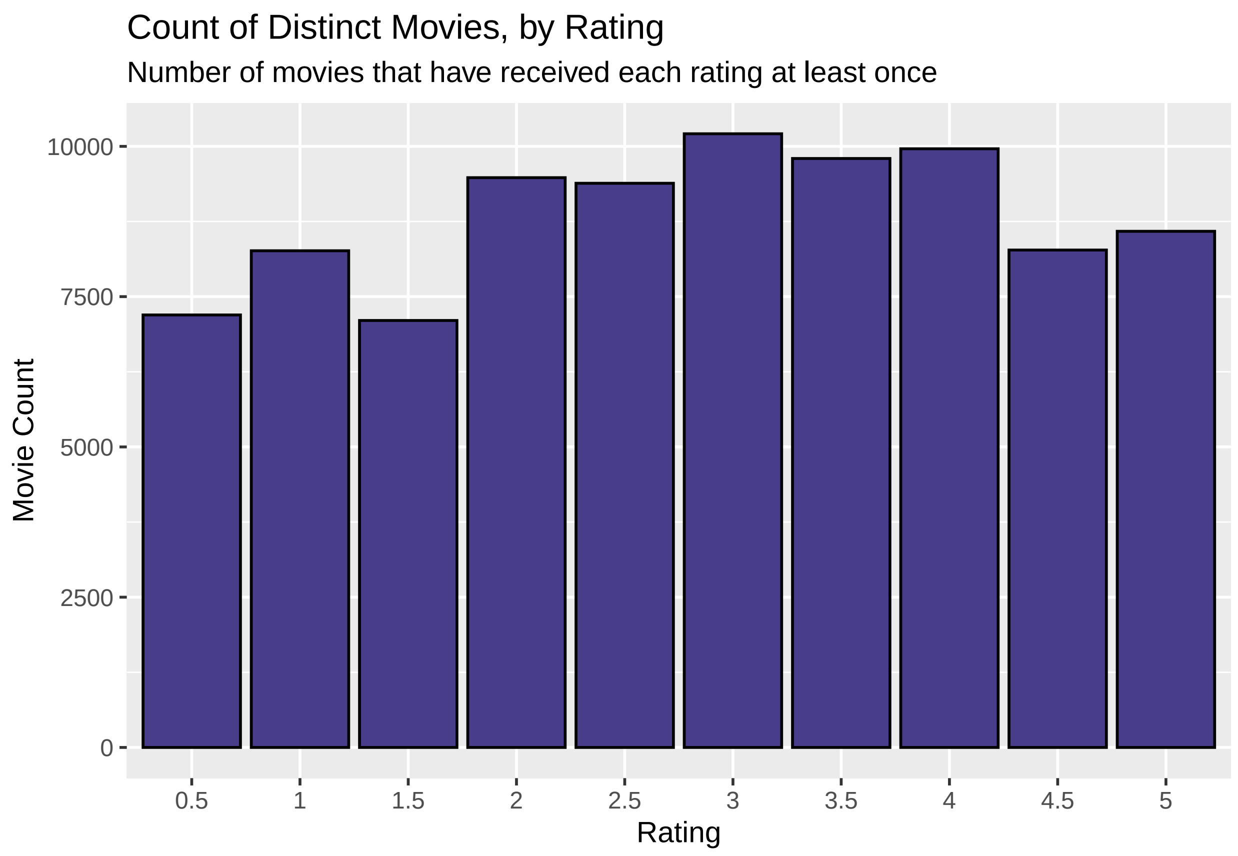
First we will explore characteristics of the ratings given by users.

```
#frequency of ratings
histrate <- edx_fortify %>% group_by(rating) %>% summarise(ratecount = n())
hrplot <- histrate %>%
  ggplot(aes(x=as.factor(rating),y=ratecount)) +
  geom_col(color = "black", fill=palette[["rating"]]) +
  labs(title="Count of Reviews, by Rating",
       x="Rating",
       y="Review Count")
hrplot
```



This chart shows that the ratings are concentrated on the even numbers, with ratings of 3 stars and 4 stars appearing the most. The mean μ is ~3.5 and the most common value (mode) is 4.

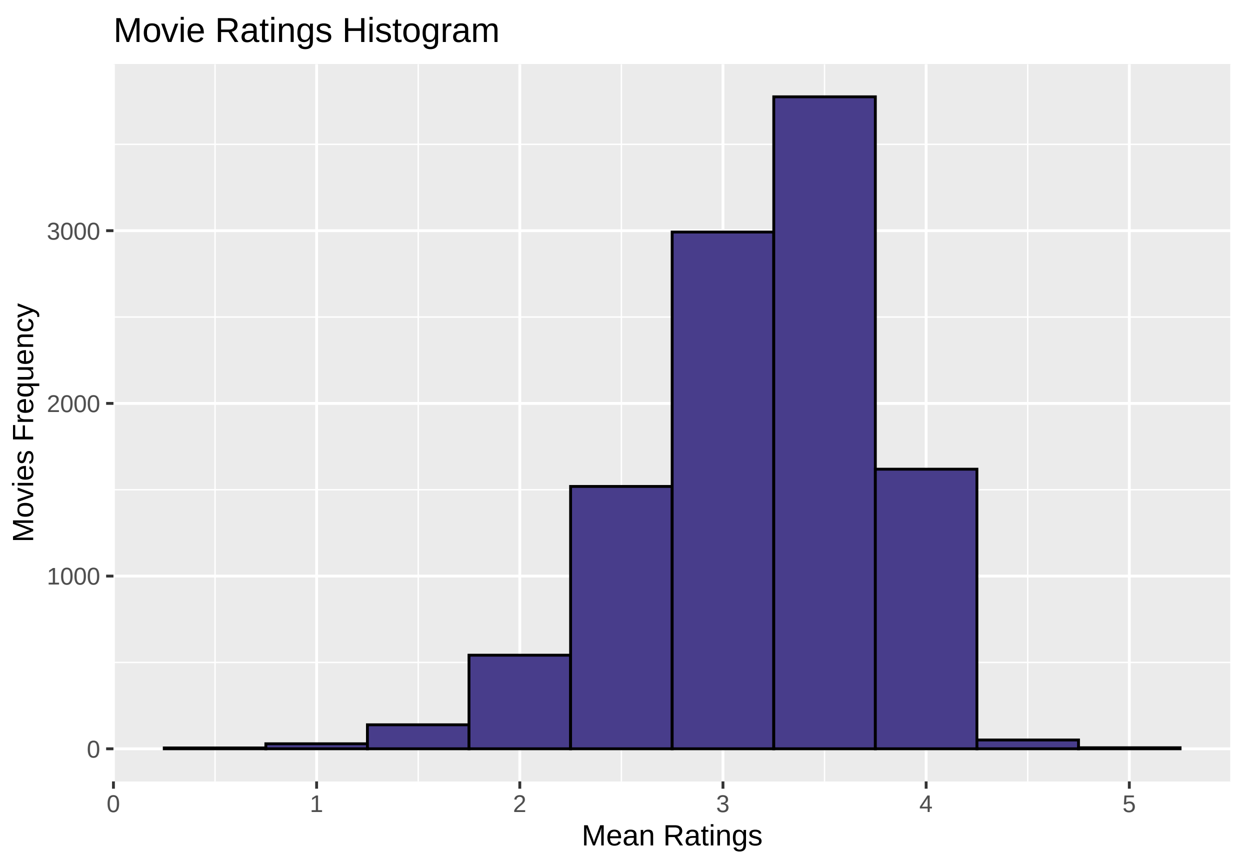
```
#show a distribution of ratings
histmovrate <- edx_fortify %>% group_by(rating) %>% summarise(ratecount = n_distinct(movieId))
hmplot <- histmovrate %>%
  ggplot(aes(x=as.factor(rating),y=ratecount)) +
  geom_col(color = "black", fill=palette[["movie"]]) +
  labs(title="Count of Distinct Movies, by Rating",
       subtitle="Number of movies that have received each rating at least once",
        x="Rating",
        y="Movie Count")
hmplot
```



A wide distribution of ratings have been given across a wide distribution of movies. 1.5 is the rating associated with the fewest movies, while a rating of 3 has been assigned to the greatest number of movies. As we already determined, 3.5 is the mean rating, a rating of 3 is the mode, slightly edging out ratings of 4 or 3.5.

Movies Analysis

```
#movie rating averages
movie_avgs <- edx_fortify %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating))

#show a histogram of raw movie ratings
movie_avgs %>%
  ggplot() +
  geom_histogram(aes(b_i),bins = 10, color="#000000", fill=palette[["movie"]])  +
  labs(title="Movie Ratings Histogram",
        x="Mean Ratings",
        y="Movies Frequency")
```



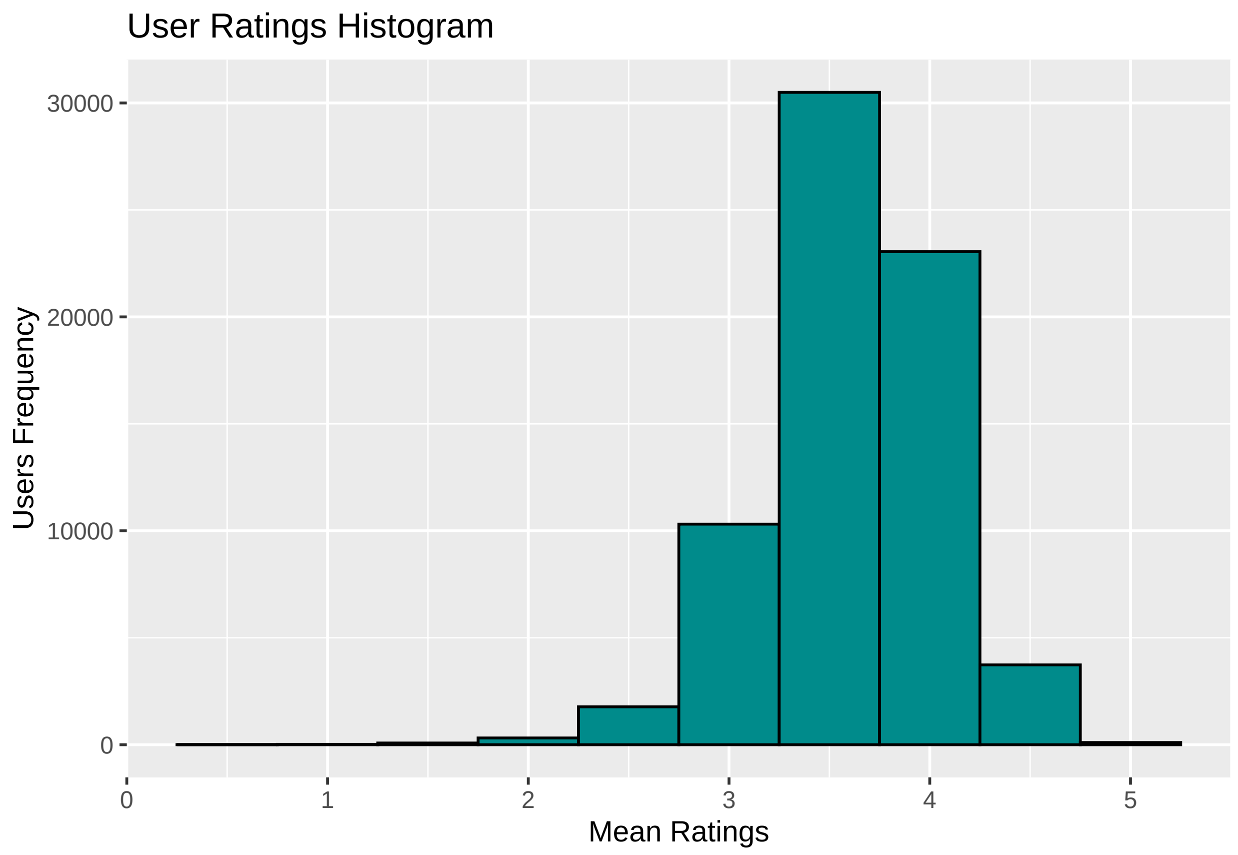
We have demonstrated that the overall average μ is about 3.5. This histogram shows how many movies average above or below this rating. There are thousands of films that average between .5 - 1 star above or below the mean, and at least 500 that average more than a full star above or below the mean. As demonstrated in Introduction to Data Science, adding a movie effect (bias) can be a powerful predictor.

The movie effect will be the first component in the construction of our model.

User Analysis

Now we take a look at user ratings behavior.

```
#show a histogram of raw user ratings
edx_fortify %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating)) %>%
     ggplot(aes(b_u)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["user"]]) +
  labs(title="User Ratings Histogram",
        x="Mean Ratings",
        y="Users Frequency")
```



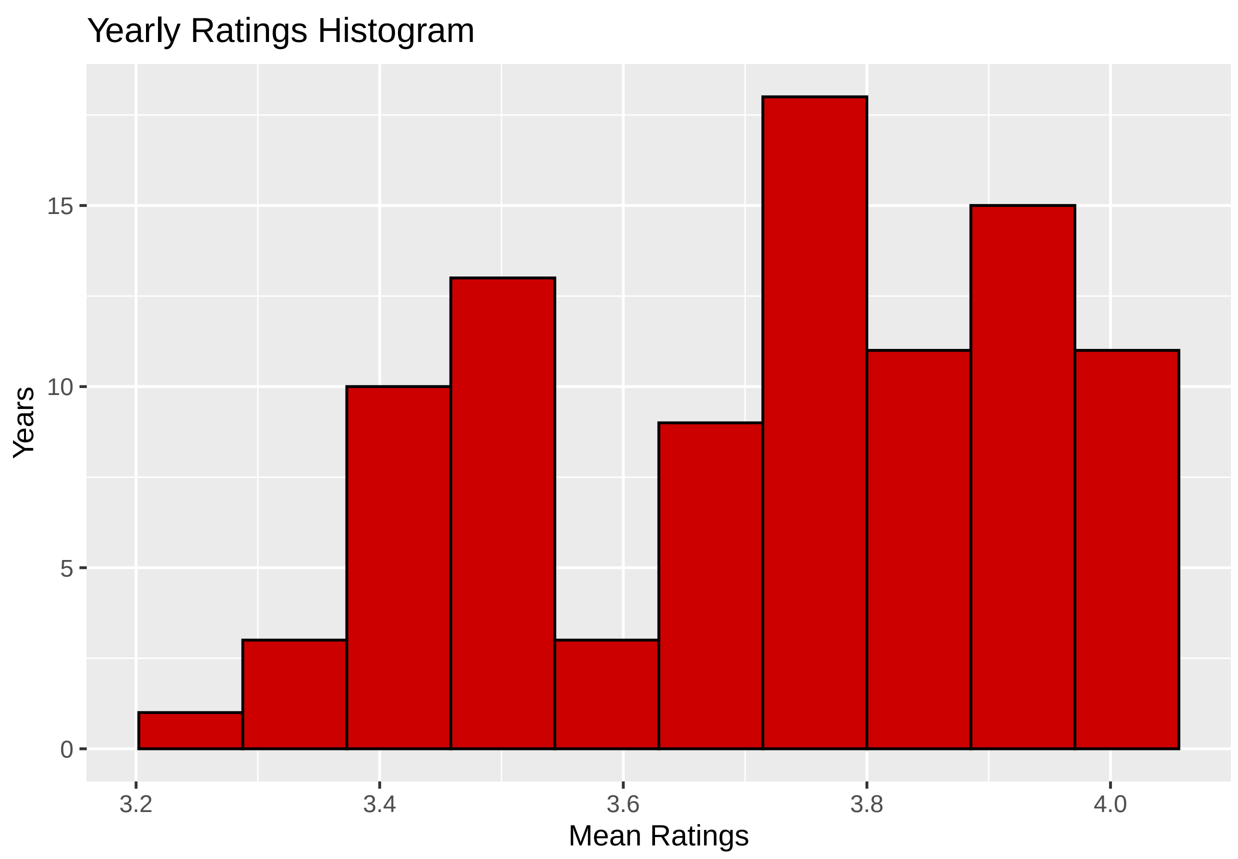
This histogram shows how many users average above or below the mean rating. There are more than 20,000 users that average closer to 4 than 3.5, and about 4,000 users who average closer to 4.5.

The user effect will be the second component in the construction of our model.

Movie Year Analysis

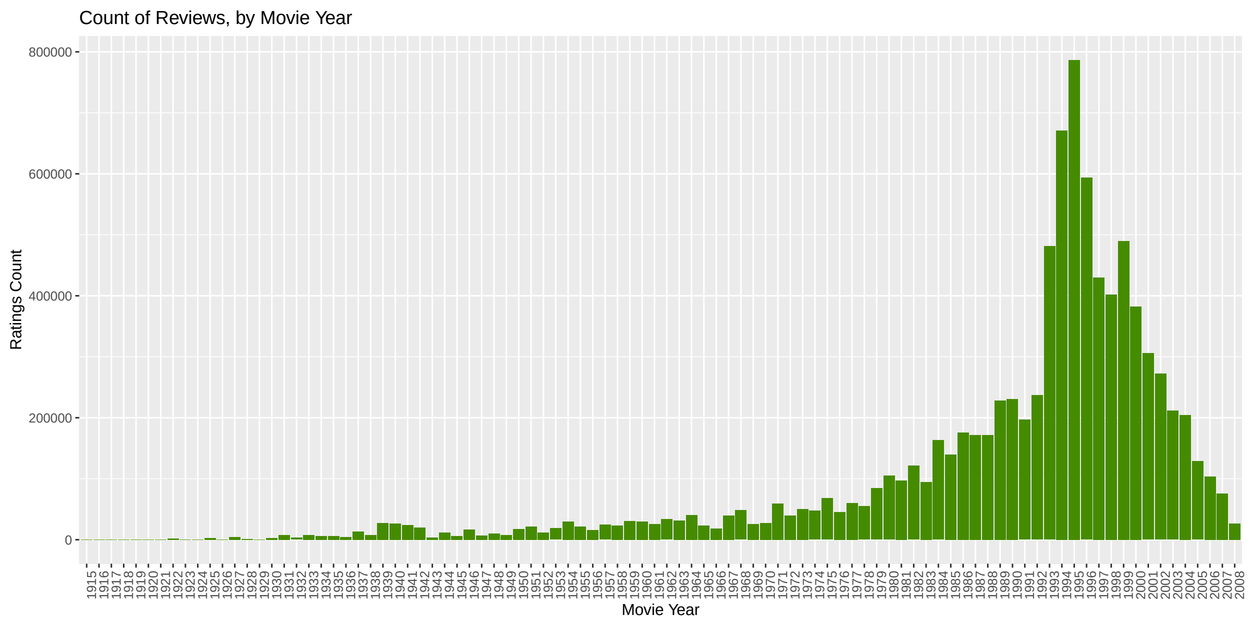
In this segment we will investigate how the release year of the movie (derived from the title) correlates with ratings given.

```r
#show a histogram of raw movie ratings by years
edx_fortify %>%
     group_by(movieyear) %>%
     summarize(b_y = mean(rating)) %>%
     ggplot(aes(b_y)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["year"]])+
  labs(title="Yearly Ratings Histogram",
        x="Mean Ratings",
        y="Years")
```



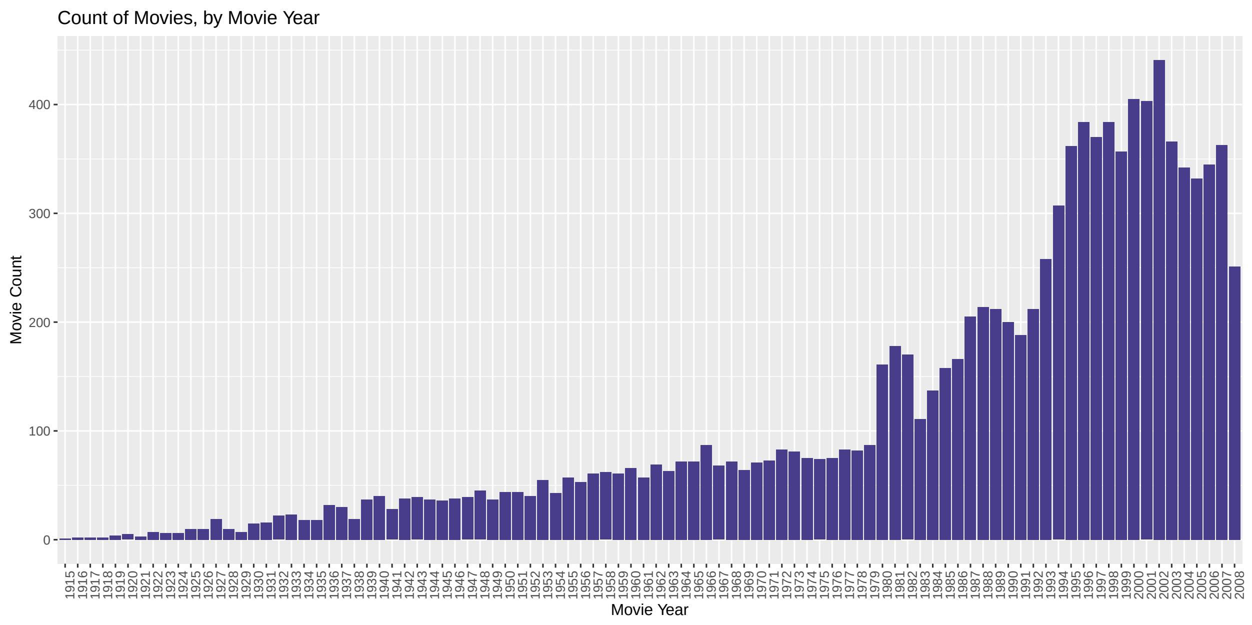
This histogram shows that many years averaged above or below the mean.

```r
#frequency of movie dates
histmovyear <- edx_fortify %>%
  group_by(movieyear) %>%
  summarise(ratecount = n())
hmplot <- histmovyear %>%
  ggplot(aes(x=as.factor(movieyear),y=ratecount)) +
  geom_col(fill=palette[["rating"]])  +
  labs(title="Count of Reviews, by Movie Year",
        x="Movie Year",
        y="Ratings Count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
hmplot
```



This chart shows that the movie year of the reviews span from 1915 to 2009. There are fewer reviews of movies in the 1920's to 1970's. The number of reviews grows from the 1980's and 2000's, peaking in the 1990's.

```r
#set a allgenres variable
allgenres <- nrow(unique(edx_fortify$genres))

#get move counts by year
moviecounts <- edx_fortify %>%
  group_by(movieyear) %>%
  summarise(moviecount = n_distinct(movieId))

#show a bar chart
hgplot <- moviecounts %>%
  ggplot(aes(x=as.factor(movieyear),y=moviecount)) +
  geom_col(fill=palette[["movie"]]) + theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Count of Movies, by Movie Year",
        x="Movie Year",
        y="Movie Count")
hgplot
```



This chart indicates that the number of different movies reviewed is greatest in the 1980s to 2000s.

To enable stratification of review data, we will assign 10-year eras to each movie. Movies of similar eras shared some broad characteristics. We will use these eras to compare movies from similar eras and look for patterns an correlations.

It is not practical to group eras evenly by number of movies- there are far more recent movies than old ones, and the groups would be very uneven in the time spans. Grouping by a set time interval (like decades) is also somewhat arbitrary, but it should help to account for the improvements in movie technology in each decade. The technology available at the time seems to have some impact on user ratings.

For example, Nosferatu (1922) receives a similar rating as Gladiator (2000). These two movies are both judged to be very good - around a 3.9 - although the former is a comparatively "crude" silent-era film in black and white, while the the latter is a modern, special effects-driven action blockbuster. Reviewers appear to judge movies from different eras using different criteria, considering the technology available at the time, and judging with some "relativity" based on the general era of the movie.

Just to add some historical context to this analysis - we will also name the eras according to some trends in movie making at the time. Below is a guide to the names. These era names and descriptions were added only for the purpose of data exploration, are not meant as exact dates, and are used only for illustrative purposes.

- 1910-1919: Early Film Era - Introduction of the first motion pictures
- 1920-1929: Era of Silent Films - Films use slides of text to communicate story
- 1930-1939: Talkies Introduced - Audio and color gradually improves experience
- 1940-1949: Technicolor Era - Technicolor adds first realistic color to films
- 1950-1959: Hollywood Blacklist Era- Communist scare, censorship & decline of large studios
- 1960-1969: Small Studio Era - Rise of small studios known as the "New Hollywood" era
- 1970-1979: Special Effects Era - Big budget visual special effects films gain popularity
- 1980-1989: Home Video Era - Home video increases popularity of lower budget films
- 1990-1999: Digital Era - Introduction of digital videography & distribution
- 2000-2009: Franchise Era - Multi-sequel epic movie franchises are popular

```r
#set mu to the overall average in the edx data.
mu <- mean(edx_fortify$rating)
```

```r
#set up averages data
histmovrate <- edx_fortify %>%
  mutate(movieyear = replace_na(movieyear, 1900)) %>%
  group_by(movieyear) %>%
  summarise(rateavg = mean(rating))


#create some long names for the eras
name_eras <- c("10s-Early Era","20s-Silent Era",
               "30s-Talkie Era","40s-Technicolor Era","50s-Blacklist Era",
               "60s-Small Studio Era","70s-Special Effects Era",
               "80s-Home Video Era","90s-Digital Era","00s-Franchise Era")

#create some short names for the eras
movie_eras <- c("early","silent","talkie","technicolor","blacklist",
                "smallstudio","specialfx","homevideo","digital","franchise")

year_eras <- c("10s","20s","30s","40s","50s",
                "60s","70s","80s","90s","2Ks")

#set the bounds of each era (we are using decades)
start_eras <- c(1910,1920,1930,1940,1950,1960,1970,1980,1990,2000)
end_eras <- c(1919,1929,1939,1949,1959,1969,1979,1989,1999,2009)
order_eras <- c(1:10)

#combine into a dataframe
mv_eragroups <- data.frame(name=name_eras,
                           code=movie_eras,
                           starts=start_eras,
                           ends=end_eras,
                           eraorder=order_eras)
mv_eragroups %>% knitr::kable(label="Eras")
```

name	code	starts	ends	eraorder
10s-Early Era	early	1910	1919	1
20s-Silent Era	silent	1920	1929	2
30s-Talkie Era	talkie	1930	1939	3
40s-Technicolor Era	technicolor	1940	1949	4
50s-Blacklist Era	blacklist	1950	1959	5
60s-Small Studio Era	smallstudio	1960	1969	6
70s-Special Effects Era	specialfx	1970	1979	7
80s-Home Video Era	homevideo	1980	1989	8
90s-Digital Era	digital	1990	1999	9
00s-Franchise Era	franchise	2000	2009	10

```r
mv_years <- c()
mv_eras <- c()
mv_names <- c()

#builds a table with an era assigned to each year
i <- 0
```

```
y <- 0
for(n in c(1:length(movie_eras)) ){
  i <- i + 1
    for (j in c(start_eras[n]:end_eras[n])){
      y <- y + 1
      mv_years[y] = j
      mv_eras[y] = movie_eras[i]
      mv_names[y] = name_eras[i]
    }
  }
mv_era <- NULL
mv_era <- data.frame(name=mv_names,era=mv_eras,movieyear=mv_years)

#join the era details to the edx data
if(is.null(edx_fortify$era)){
  edx_fortify <- edx_fortify %>% left_join(mv_era, by = "movieyear")
}
```

We have added the era details needed to stratify and (hopefully) improve the accuracy of our predictions.

We will now examine a few of the top movies from each era.

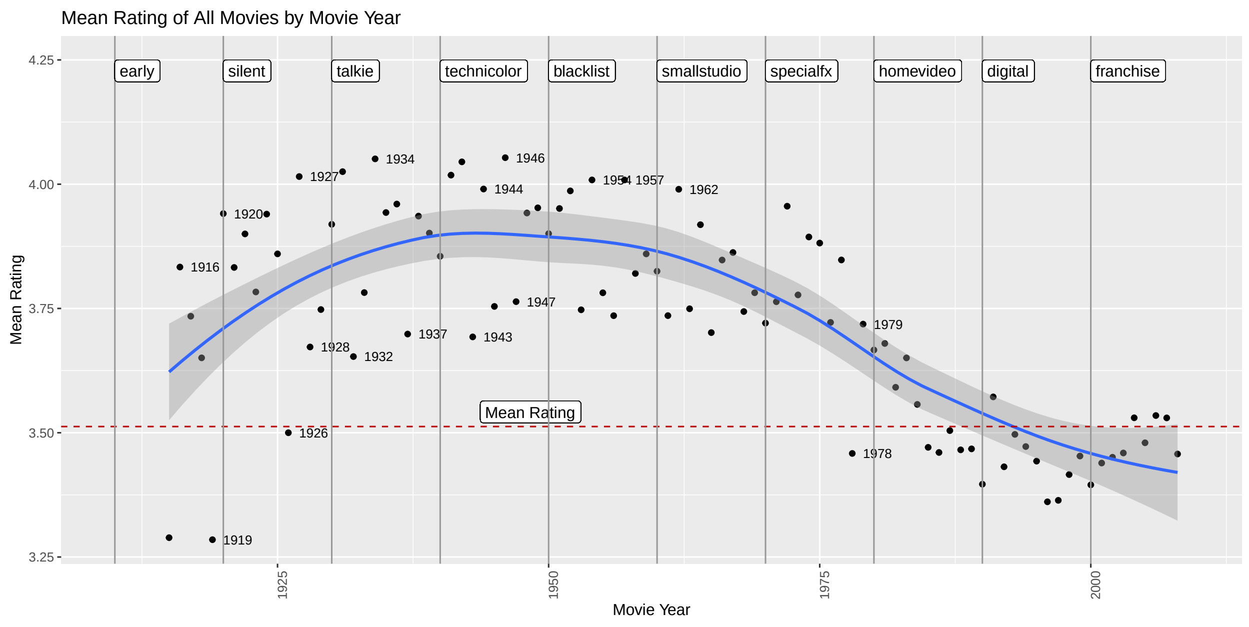
```
#top titles
top_annualrating <- edx_fortify %>%
  group_by(era, movieId, titlenoyear, movieyear) %>%
  summarise(ratecount = n(),
            avgrating = round(mean(rating),2),
            sdrating = sd(rating)) %>%
  arrange(-ratecount) %>%
  group_by(era) %>%
  slice(1:3) %>%
  mutate(titleny = paste(substr(titlenoyear, 1, 30),
                        ifelse(str_length(titlenoyear) > 30,'...','') )) %>%
  select(era,avgrating,ratecount, titleny, movieyear) %>%
  left_join(mv_eragroups, by = c("era"="code")) %>%
  arrange(eraorder, -avgrating) %>%
  select(name, era, titleny, avgrating, movieyear, ratecount)

top_annualrating %>% knitr::kable(label="Top 5 Most Rated Movies Each Era")
```

name	era	titleny	avgrating	movieyear	ratecount
10s-Early Era	early	Intolerance	3.85	1916	80
10s-Early Era	early	Daddy Long Legs	3.39	1919	73
10s-Early Era	early	Birth of a Nation, The	3.29	1915	180
20s-Silent Era	silent	General, The	4.27	1927	997
20s-Silent Era	silent	Metropolis	4.01	1927	2454
20s-Silent Era	silent	Nosferatu (Nosferatu, eine Sym ...	3.92	1922	1721
30s-Talkie Era	talkie	Wizard of Oz, The	4.00	1939	11607
30s-Talkie Era	talkie	Gone with the Wind	3.80	1939	7387
30s-Talkie Era	talkie	Snow White and the Seven Dwarf ...	3.62	1937	9308
40s-Technicolor Era	technicolor	Casablanca	4.32	1942	11232
40s-Technicolor Era	technicolor	Citizen Kane	4.19	1941	8584
40s-Technicolor Era	technicolor	It's a Wonderful Life	4.07	1946	6727

name	era	titleny	avgrating	movieyear	ratecount
50s-Blacklist Era	blacklist	Rear Window	4.32	1954	7935
50s-Blacklist Era	blacklist	North by Northwest	4.26	1959	7525
50s-Blacklist Era	blacklist	Vertigo	4.16	1958	6728
60s-Small Studio Era	smallstudio	Dr. Strangelove or: How I Lear …	4.30	1964	10627
60s-Small Studio Era	smallstudio	Psycho	4.09	1960	9394
60s-Small Studio Era	smallstudio	2001: A Space Odyssey	3.96	1968	11796
70s-Special Effects Era	specialfx	Godfather, The	4.42	1972	17747
70s-Special Effects Era	specialfx	Star Wars: Episode IV - A New …	4.22	1977	25672
70s-Special Effects Era	specialfx	Monty Python and the Holy Grai …	4.21	1975	14635
80s-Home Video Era	homevideo	Star Wars: Episode V - The Emp …	4.19	1980	20729
80s-Home Video Era	homevideo	Star Wars: Episode VI - Return …	4.00	1983	22584
80s-Home Video Era	homevideo	Batman	3.39	1989	24277
90s-Digital Era	digital	Silence of the Lambs, The	4.20	1991	30382
90s-Digital Era	digital	Pulp Fiction	4.15	1994	31362
90s-Digital Era	digital	Forrest Gump	4.01	1994	31079
00s-Franchise Era	franchise	Lord of the Rings: The Fellows …	4.16	2001	14406
00s-Franchise Era	franchise	Gladiator	3.94	2000	13931
00s-Franchise Era	franchise	Shrek	3.90	2001	13063

Now we can build a plot showing movie ratings over these various eras.

```r
#show chart of ratings over time
hmrplot <- histmovrate %>% ggplot() +
  geom_point(aes(x=movieyear,y=rateavg)) +
  geom_smooth(aes(x=movieyear,y=rateavg))

hmrplot <- hmrplot +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +

  geom_label(mapping=aes(x=1940,
                         y=3.52,
                         label='Mean Rating'),
            size=4,
            hjust=-0.4,
            vjust=0) +
  geom_text(size=3,aes(label = ifelse(abs(rateavg-lag(rateavg,1)) > 0.25,movieyear,''),
                        x=movieyear,
                        y=rateavg),
            hjust=0,
            nudge_x=1) +

  geom_vline(xintercept = start_eras[1], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[1], y=4.25,
                         label=movie_eras[1]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[2], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[2], y=4.25,
                         label=movie_eras[2]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[3], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[3], y=4.25,
                         label=movie_eras[3]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[4], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[4], y=4.25,
                         label=movie_eras[4]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[5], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[5], y=4.25,
                         label=movie_eras[5]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[6], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[6], y=4.25,
                         label=movie_eras[6]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[7], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[7], y=4.25,
                         label=movie_eras[7]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[8], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[8], y=4.25,
                         label=movie_eras[8]), size=4, angle=90, vjust=1, hjust=0) +

  geom_vline(xintercept = start_eras[9], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[9], y=4.25,
                         label=movie_eras[9]), size=4, angle=90, vjust=1, hjust=0) +
  geom_vline(xintercept = start_eras[10], colour="#999999") +
  geom_label(mapping=aes(x=start_eras[10], y=4.25,
                         label=movie_eras[10]), size=4, angle=90, vjust=1, hjust=0) +

theme(axis.text.x = element_text(angle = 90, hjust = 1))  +
  labs(title="Mean Rating of All Movies by Movie Year",
        x="Movie Year",
        y="Mean Rating")

hmrplot
```



The overall, all-time average rating is about 3.5. This value is represented by a dashed red line in the chart above, and in many of the following charts. Ratings follow a pattern over time. Early movies before 1930 trended lower, while movies in the 1940-1960 range get generally better than average ratings. The curve descends in 1960-1980 and then levels off slightly in the 1980-2000's.

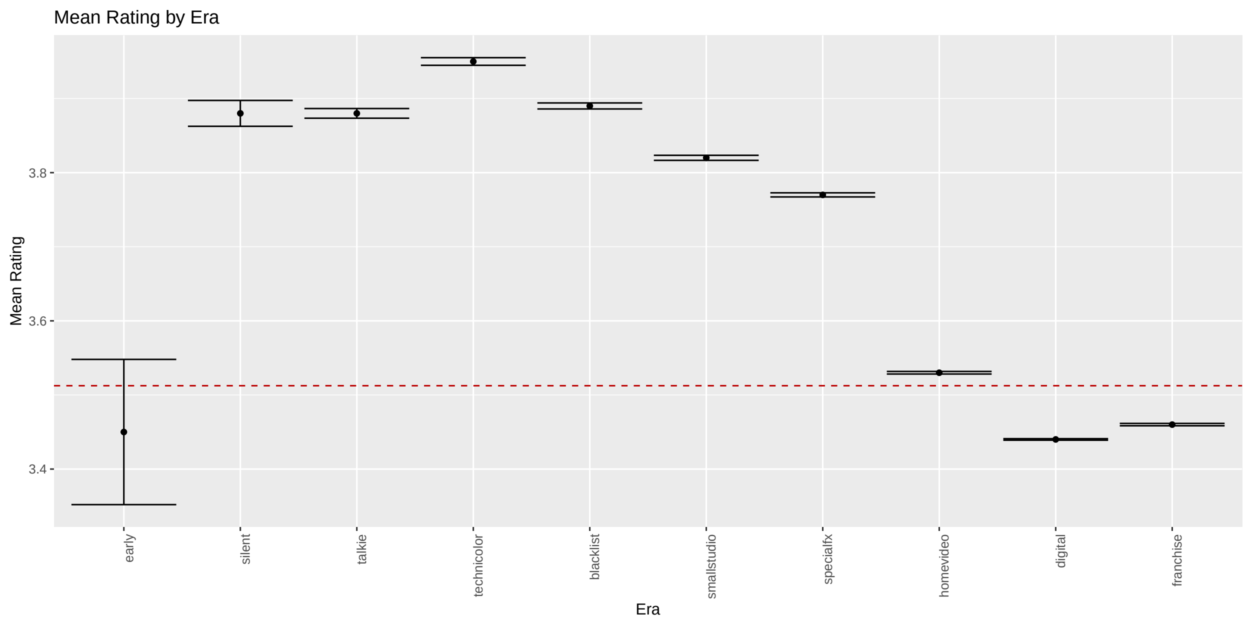
Before 1980, the average ratings have greater yearly variance. Each year which saw a greater than .25 increase/decrease on average from the prior year, is stamped on the chart above. The last year that saw this wide of variance (1979) is also when the number of movies reviewed starts to increase substantially.

```
#get data by era
histeras <- edx_fortify %>%
  group_by(era,name) %>%
  summarise(movies = n_distinct(movieId),
            avg = round(mean(rating),2),
            se = sd(rating)/sqrt(n()))

histeras$era_f = factor(histeras$era, levels=movie_eras)

#plot couunts by era/decade
histeras %>%
  arrange(era_f) %>%
  select(-era_f) %>%
  knitr::kable(label="Ratings & Movie Counts by Era")
```

era	name	movies	avg	se
early	10s-Early Era	11	3.45	0.0489983
silent	20s-Silent Era	83	3.88	0.0087229
talkie	30s-Talkie Era	230	3.88	0.0032599
technicolor	40s-Technicolor Era	377	3.95	0.0025708
blacklist	50s-Blacklist Era	520	3.89	0.0020285
smallstudio	60s-Small Studio Era	690	3.82	0.0017057
specialfx	70s-Special Effects Era	784	3.77	0.0013842
homevideo	80s-Home Video Era	1712	3.53	0.0008688

era	name	movies	avg	se
digital	90s-Digital Era	3022	3.44	0.0005061
franchise	00s-Franchise Era	3248	3.46	0.0007948

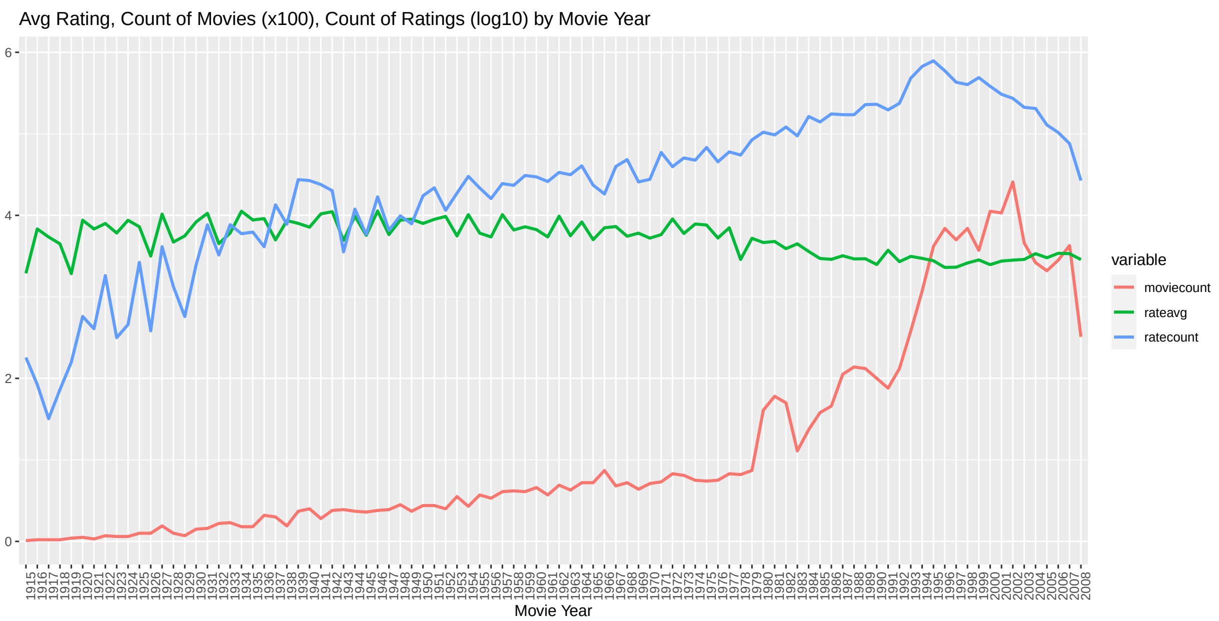
```
#plot ratings by era/decade
histeras %>%
  ggplot(aes(x = era_f, y = avg,
             ymin = avg - 2*se,
             ymax = avg + 2*se)) +
  geom_point() +
  geom_hline(yintercept = mu, colour="#BB0000", linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating by Era",
       x="Era",
       y="Mean Rating")
```



The average ratings by eras roughly match the average rating by year curve seen above. Each scatter plot shows a point for the the mean rating, with error bars to indicate the standard error range. This plot format will be used throughout the report.

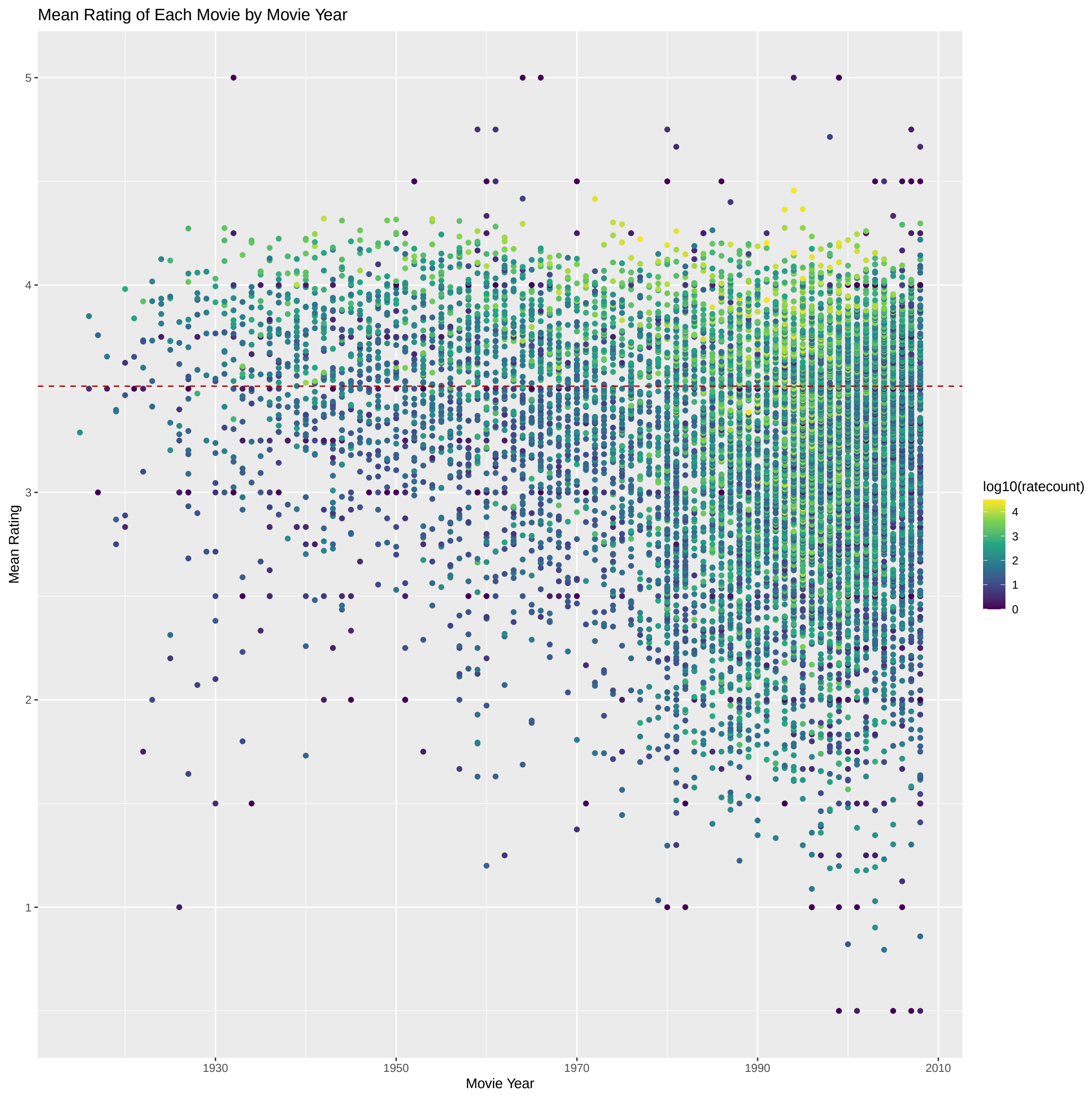
```
#build a table with movie count & rating count, & rating avg by year
histmovrate <- edx_fortify %>%
  group_by(movieyear) %>%
  summarise(rateavg = mean(rating), ratecount = n())

histmovrate <- histmovrate %>%
  left_join(moviecounts, by = "movieyear")

histmovrate <- histmovrate %>%
  mutate(moviecount = moviecount/100)
```

```
histmovrate <- histmovrate %>%
  mutate(ratecount =log10(ratecount))

#pivot to a tidy format
df2 <- tidyr::pivot_longer(histmovrate,
                           cols=c('rateavg', 'moviecount','ratecount'),
                           names_to='variable',
values_to="value")

#render plot
hmrplot <- df2 %>%
  ggplot(aes(x=as.factor(movieyear),
             y=value,
             color=variable,
             group=variable)) +
  geom_path(size=1) +
  geom_vline(xintercept=1980) +
  theme(axis.text.x = element_text(angle = 90,
                                   hjust = 1)) +
  labs(title="Avg Rating, Count of Movies (x100), Count of Ratings (log10) by Movie Year",
       x="Movie Year",
       y="")
hmrplot
```



The variance year by year could be partly explained by the lower number of movies represented in the earlier years, giving greater weight to fewer, more widely reviewed films. While there are far fewer movies from the 20-60's, they tend to be more favorably viewed.

```
#build a of each movies average rating by year.
moviedates <- edx_fortify %>%
  group_by(era, movieyear, movieId, title) %>%
  summarise(rateavg = mean(rating),
            ratecount = n()) %>%
  arrange(rateavg)

#render scatterplot with dots for each movie
moviedates %>% ggplot(aes(y=rateavg,
                          x=movieyear,
                          color=log10(ratecount))) +
  geom_point() +
   geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
 labs(title="Mean Rating of Each Movie by Movie Year",
        x="Movie Year",
        y="Mean Rating") + scale_color_viridis_c()
```

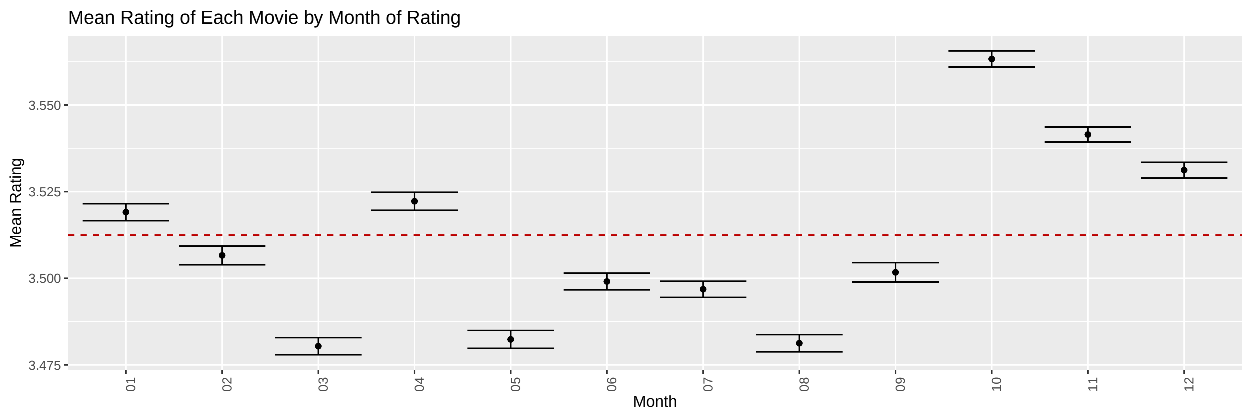


This chart colors the dots based on number of ratings submitted - which reveals another trend worth noting - that many of the above average rated moves (above the dotted line) are also movies with many ratings (lighter/greener color). This implies that we should explore the correlation between the average ratings, and number of ratings submitted - a term we will generally refer to as "popularity".

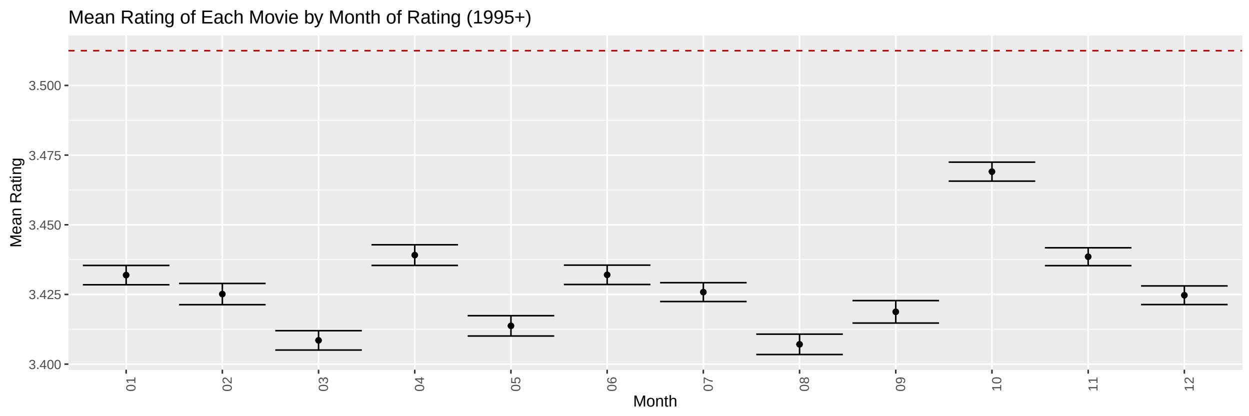
The trends observed with respect to film ratings over time make sense if you consider the history of movies, and the time periods for which reviews were being collected. Earlier films that survived to be widely reviewed 30, 50, or 70+ years after they were released, probably exhibited some qualities worth noting - a great story, prominent actors, notable effects, or important/historic value. Many older films were lost due to film decay, fires, and accidents. Better films were more likely to be preserved.

Based on the above analysis - it is clear that considering the movie year may impact the accuracy of the model. We will attempt to incorporate the movie "year effect" into the construction of the model.

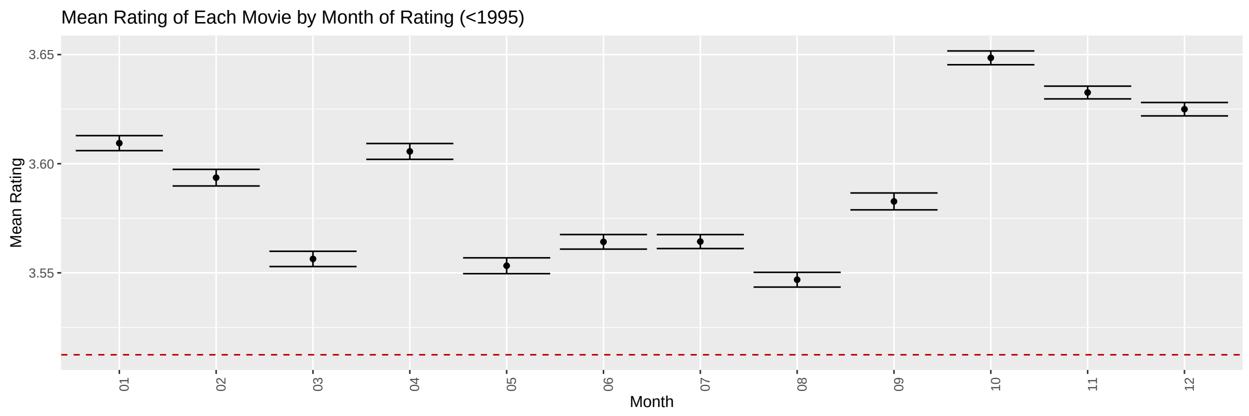
```r
#build a of each movies average rating by year.
moviemonths <- edx_fortify %>%
  group_by(ratemonth) %>%
  summarise(rateavg = mean(rating),
            ratesd = sd(rating),
            ratecount = n(),
            ratese = sd(rating)/sqrt(n())) %>%
  arrange(rateavg)

#compare this rating by month of movies during the period of the survey
moviemonthsnew <- edx_fortify %>%
  filter(movieyear >= 1995) %>%
  group_by(ratemonth) %>%
  summarise(rateavg = mean(rating),
            ratesd = sd(rating),
            ratecount = n(),
            ratese = sd(rating)/sqrt(n())) %>%
  arrange(rateavg)

#and movies that pre-dates the period of the ratings survey
moviemonthsold <- edx_fortify %>%
  filter(movieyear < 1995) %>%
  group_by(ratemonth) %>%
  summarise(rateavg = mean(rating),
            ratesd = sd(rating),
            ratecount = n(),
            ratese = sd(rating)/sqrt(n())) %>%
  arrange(rateavg)

#render scatterplot with dots for each movie
moviemonths %>%
  ggplot(aes(x = ratemonth,
             y = rateavg,
             ymin = rateavg - 2*ratese,
             ymax = rateavg + 2*ratese)) +
  geom_point() +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating of Each Movie by Month of Rating",
        x="Month",
        y="Mean Rating")
```



```
moviemonthsnew %>%
  ggplot(aes(x = ratemonth,
             y = rateavg,
             ymin = rateavg - 2*ratese,
             ymax = rateavg + 2*ratese)) +
  geom_point() +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating of Each Movie by Month of Rating (1995+)",
       x="Month",
       y="Mean Rating")
```



```
moviemonthsold %>%
  ggplot(aes(x = ratemonth,
             y = rateavg,
             ymin = rateavg - 2*ratese,
             ymax = rateavg + 2*ratese)) +
  geom_point() +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating of Each Movie by Month of Rating (<1995)",
       x="Month",
       y="Mean Rating")
```



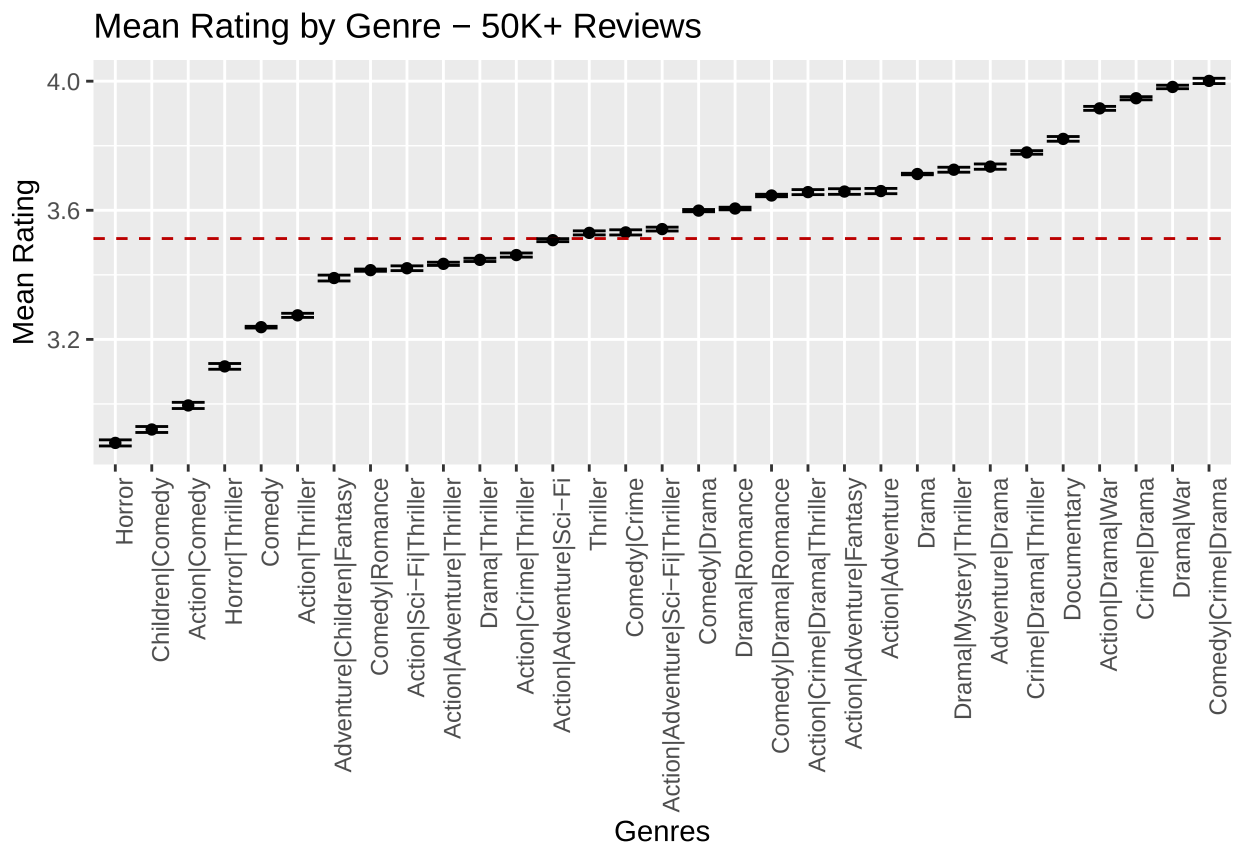
This chart illustrates an interesting phenomenon related to the date - it appears that the month of the review has some correlation with mean ratings given. Ratings submitted during Q4 (Months 10,11,12) seem to rate higher. This seasonal trend holds true even for movies released before 1995 when ratings began being submitted.

Perhaps it is related to the seasonal nature of movie releases. It is reasonable to assume that more people might be indoors during the winter and movies might be more popular during that time- and you might also see some correlation between types of movies reviewed in various seasons - like Horror movies in October, or Holiday movies in December. Incorporating a consideration of the month when the movie was rated could be useful to identify these seasonal trends in the type and quality of movies being released.

Genre Analysis

In this section we will look at how movie genres correlate with ratings.

```
#boxplot of movie ratings by genre
edx_fortify %>%
  group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n>50000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres,
             y = avg,
             ymin = avg - 2*se,
             ymax = avg + 2*se)) +
  geom_point() +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating by Genre - 50K+ Reviews",
       x="Genres",
       y="Mean Rating")
```



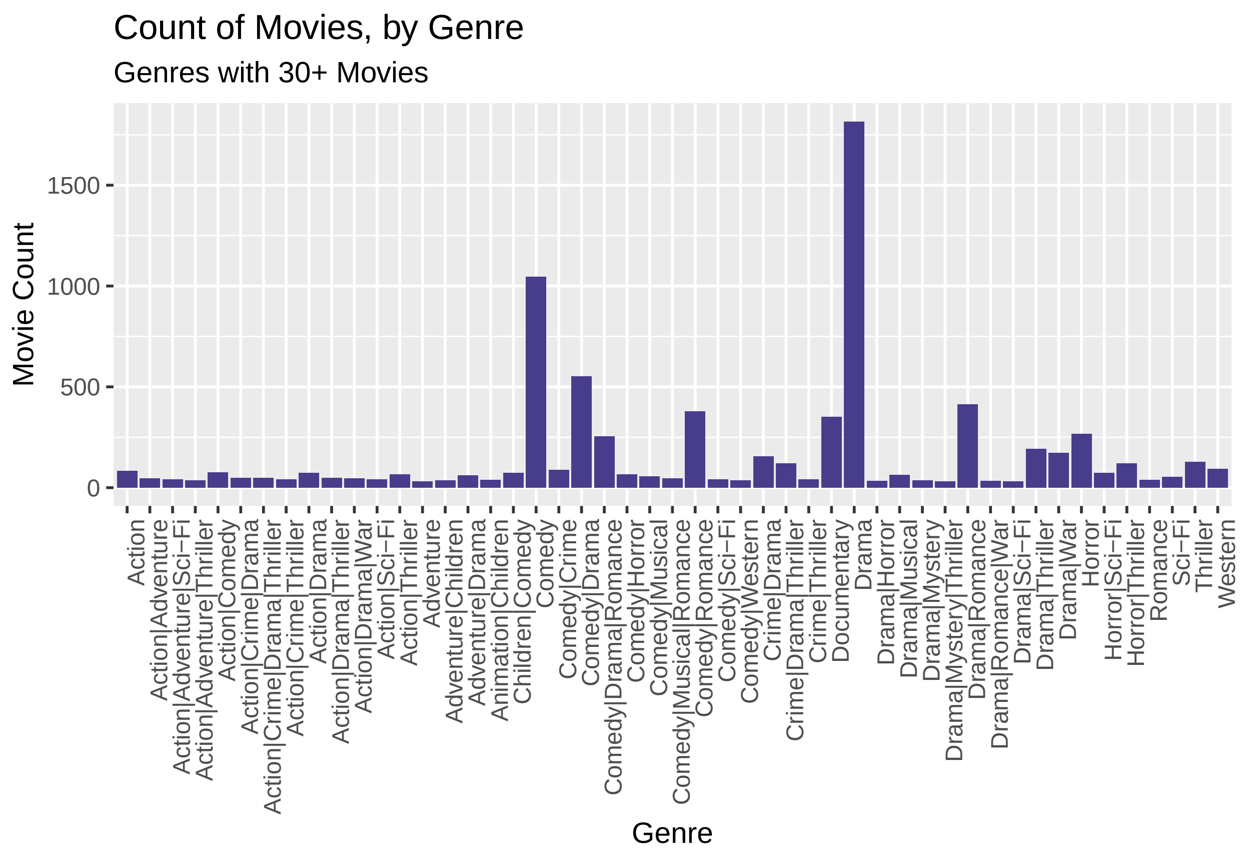
Some genres receive consistently higher ratings than others. Our model will consider this when making predictions. Movies matching a well liked genre, should be assigned a higher prediction.

One possible challenge with using the the genre data, is that the genre data is a multi-value list. There are 19 individual genres that are applied to create 797 different combination genres, across the ~10,000 movies. Some genres are very popular and somewhat general (like Drama) while others are very specific and less frequently assigned.

```
totgenres <- edx_fortify %>%
  summarise(gcount = n_distinct(genres))
totgenres
```

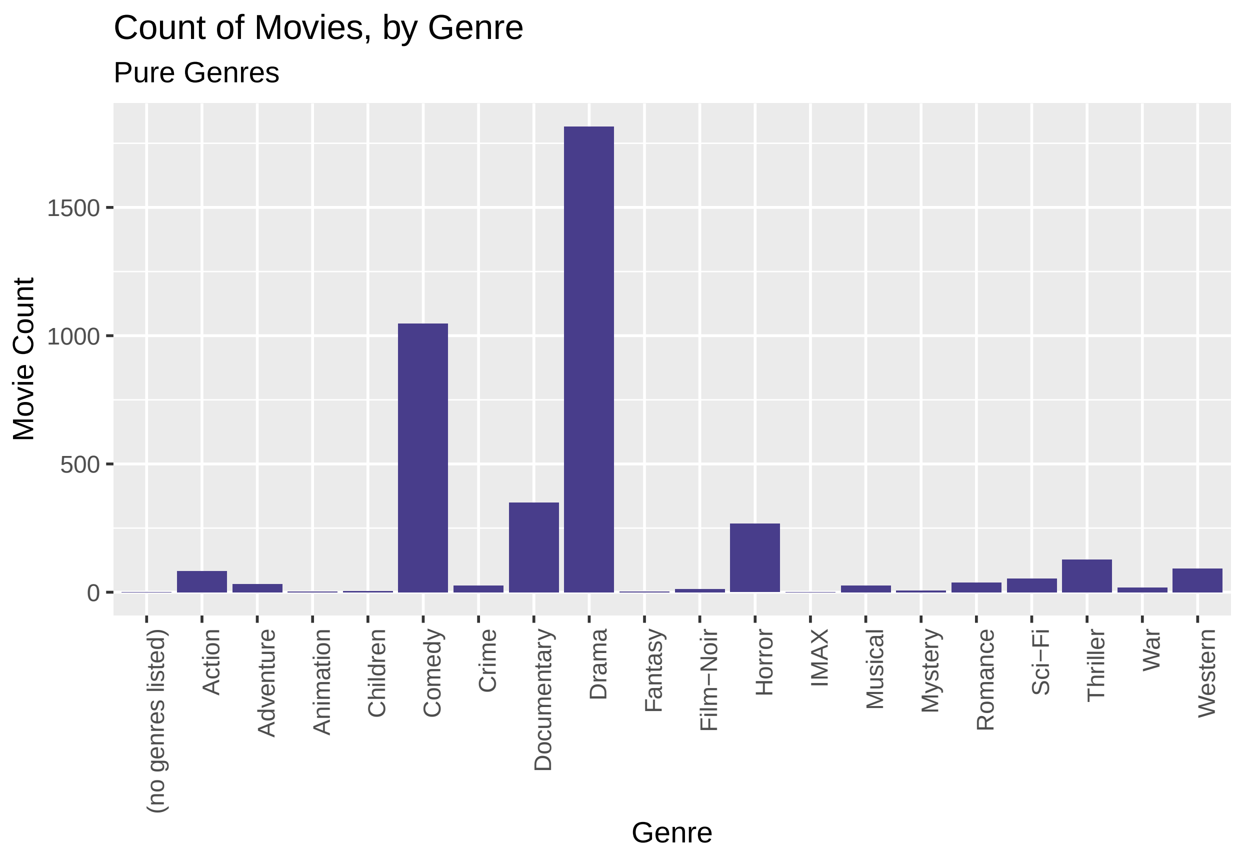
```
##   gcount
## 1    797
```

```
#get details about the genres
histgenres <- edx_fortify %>%
  group_by(genres) %>%
  summarise(ratingcount = n(),
            moviecount = n_distinct(movieId),
            rateavg = mean(rating),
            ratesd = sd(rating))
histgenres <- histgenres %>%
  mutate(gcount = 0)
histgenres$gcount <- lengths(strsplit(as.character(histgenres$genres), "|", fixed = TRUE))

#show a bar chart
hgplot <- histgenres %>% filter(moviecount>30) %>%
  ggplot(aes(x=as.factor(genres),y=moviecount)) +
  geom_col(fill=palette[["movie"]]) + theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Count of Movies, by Genre",
       subtitle="Genres with 30+ Movies",
        x="Genre",
        y="Movie Count")
hgplot
```

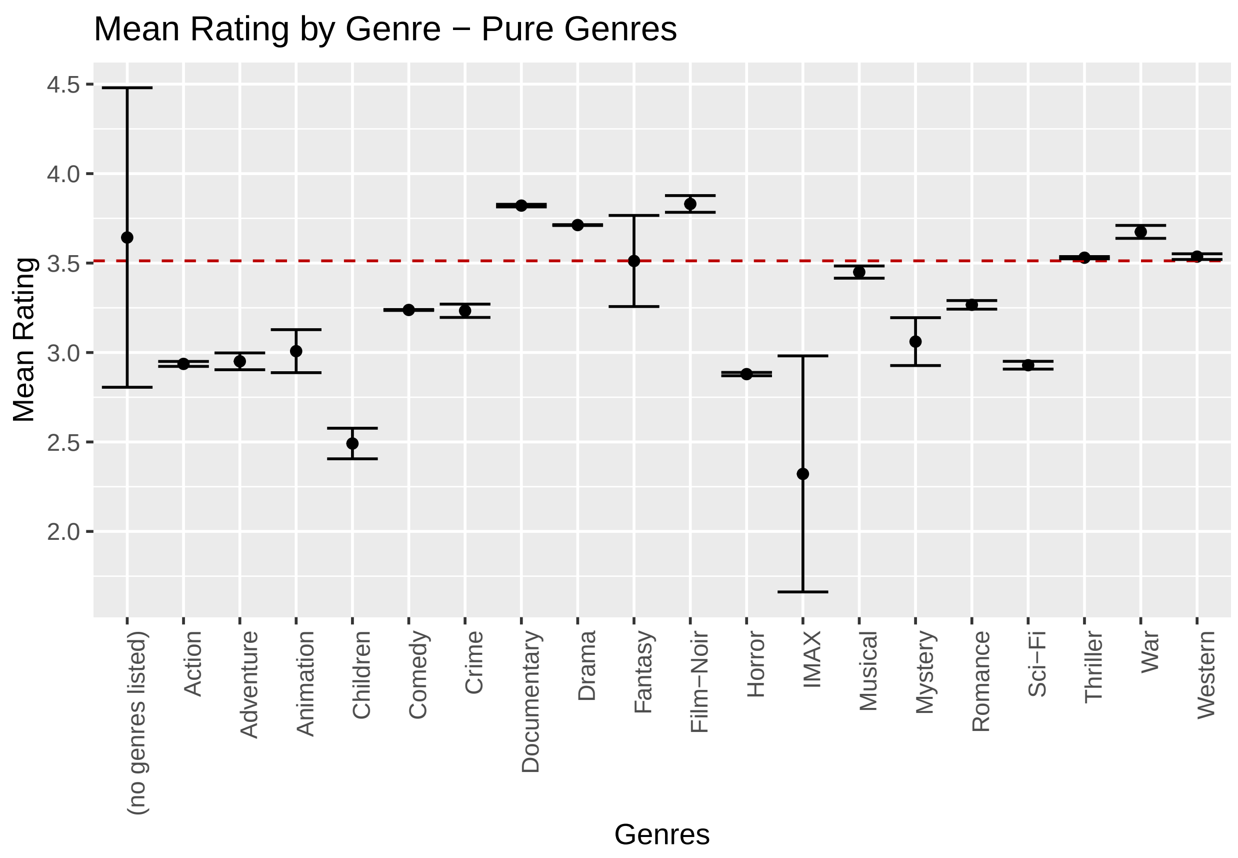


Out of over 10,000 movies, 4,004 have only one genre assigned. We will call these *pure genre* movies. We must consider how movies in these pure genres like Comedy, Drama, and Action are rated, and how these ratings compare to various combination genres.

```
#get the "pure" genres for use in the box plot below
puregenres <- edx_fortify %>%
  left_join(histgenres, by = "genres") %>%
  filter(gcount == 1) %>%
  group_by(genres) %>%
  summarize(movies = n_distinct(movieId))

pgplot <- puregenres %>%
  ggplot(aes(x=as.factor(genres),y=movies)) +
  geom_col(fill=palette[["movie"]]) + theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Count of Movies, by Genre",
       subtitle="Pure Genres",
        x="Genre",
        y="Movie Count")
pgplot
```

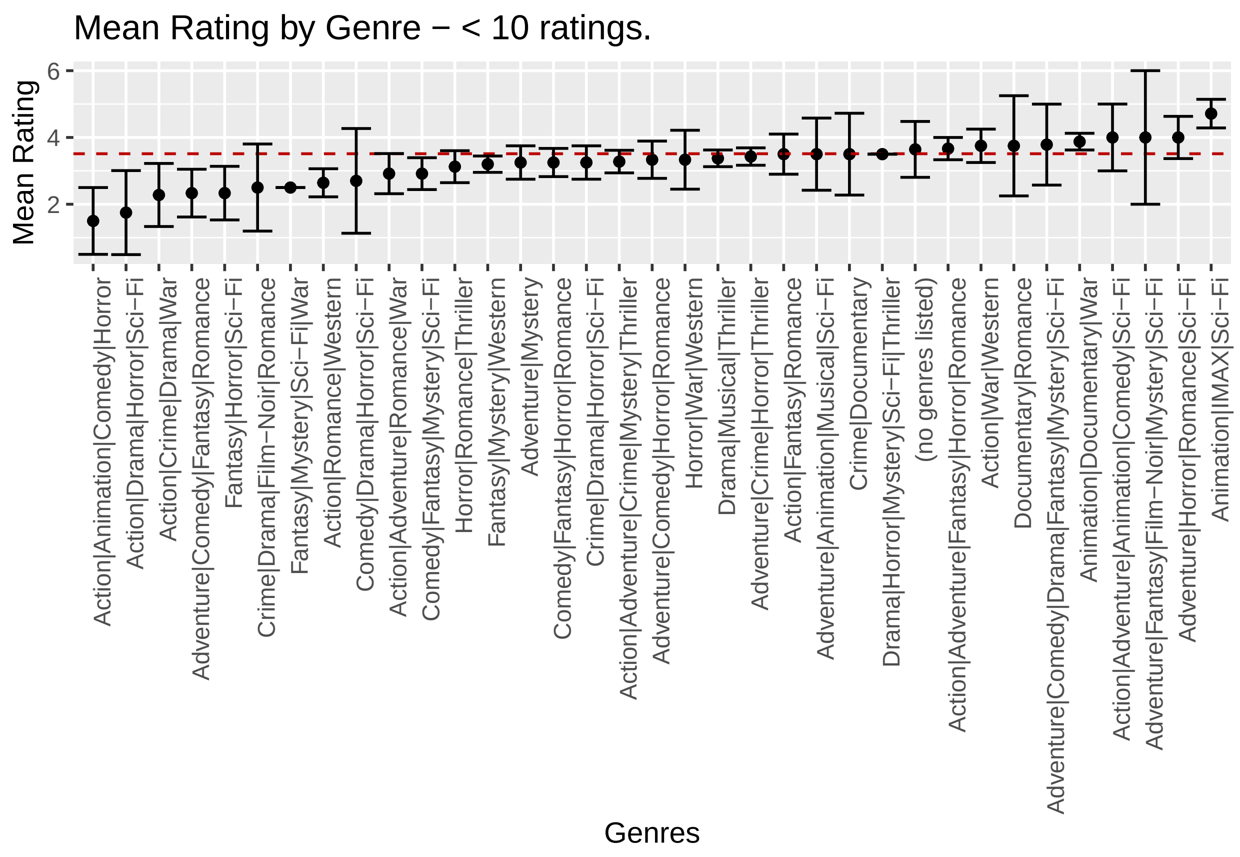


```
#show box plot
edx_fortify %>%
  group_by(genres) %>%
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n()))  %>%
  left_join(histgenres, by = "genres") %>%
  filter(gcount == 1) %>%
  ggplot(aes(x =genres,
             y = avg,
             ymin = avg - 2*se,
             ymax = avg + 2*se)) +
  geom_point() +
  geom_hline(yintercept = mu, colour="#BB0000", linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating by Genre - Pure Genres",
        x="Genres",
        y="Mean Rating")
```



This plot shows that there is substantial variance in the number of movies assigned pure genres and their mean ratings. There are many pure Comedy and Drama, and very few pure Fantasy, Adventure, etc. Five of the pure genres have fewer than 10 movies in them. One has "No Genres Listed".

```
#boxplot of movie ratings by genre
edx_fortify %>% group_by(genres) %>%
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n())) %>%
  filter(n<10) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres,
             y = avg,
             ymin = avg - 2*se,
             ymax = avg + 2*se)) +
  geom_point() +
  geom_hline(yintercept = mu, colour="#BB0000", linetype="dashed") +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating by Genre - < 10 ratings.",
        x="Genres",
        y="Mean Rating")
```

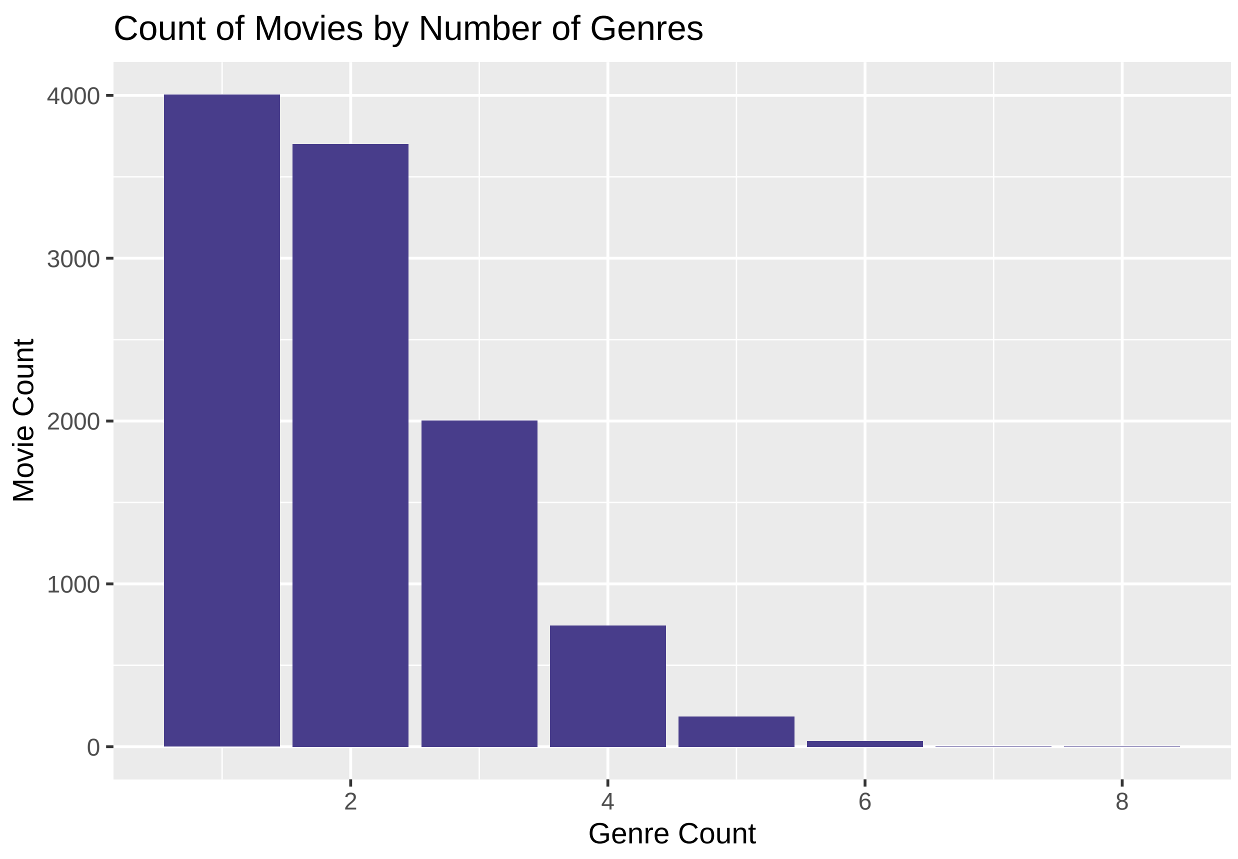


There many ways to approach the use of these categorical values. These could be split into columns of multiple variables, and try to derive an average rating across all movies containing any shared genres. Another approach could treat the combined genre as a unique value and consider the rating average of only those movies with the same combination of genres.

It appears that the combined genre method may capture more predictive rating data - Some genres shown like Comedy|Crime|Drama (rating ~ 4.0 on average) rate higher than any of the individual components like Comedy (3.25), Crime (3.25), or Drama (3.75) alone. A combination genre may also indicate the movie is unlike a movie that is one or the other genre.

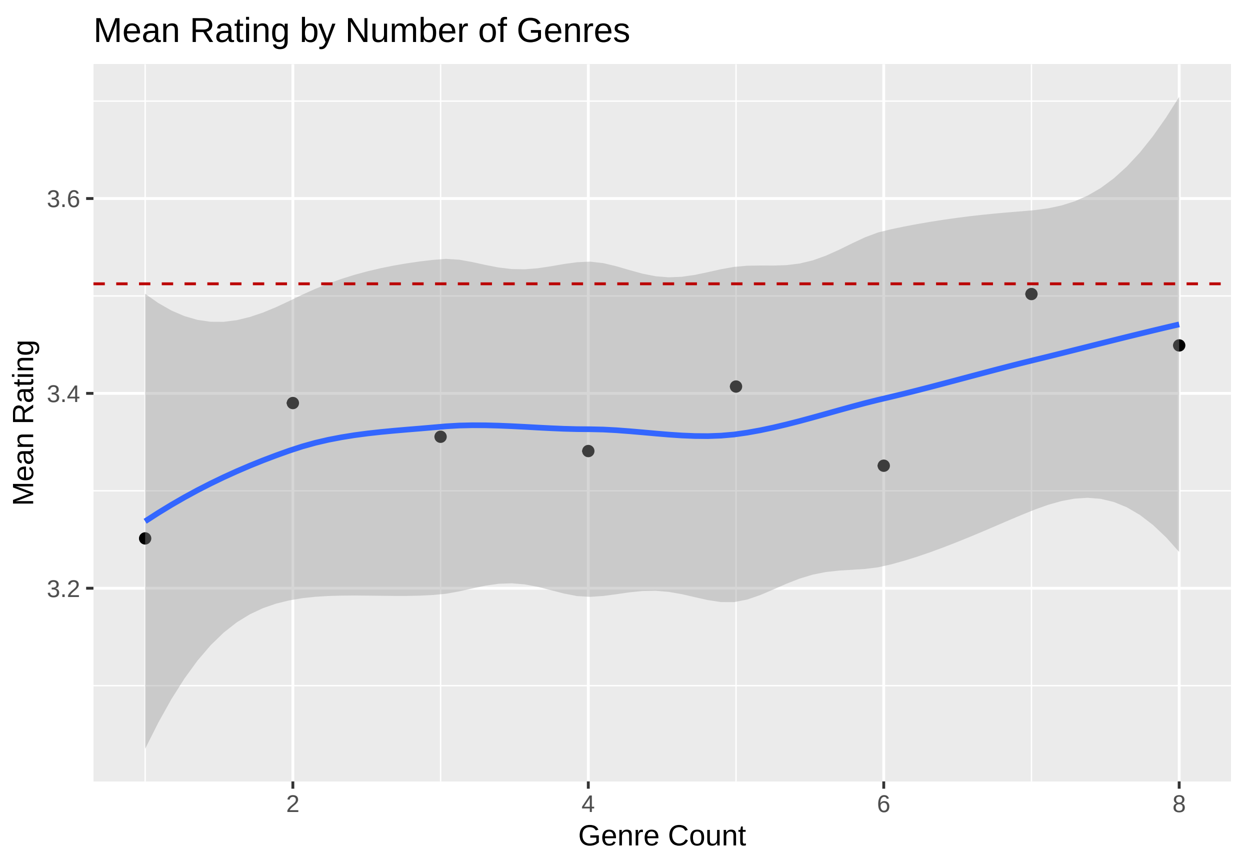
For example, a "Comedy|War", may have little in common with a typical War or Comedy film. The chart above demonstrates that most of the more obscure genres with very few ratings submitted, also get slightly lower mean ratings.

```
#get movie counts by genre counts
genrecounts <- histgenres %>%
  group_by(gcount) %>%
  summarize(moviecount = sum(moviecount),
            rateavg = mean(rateavg)
             )

#show bar chart
plotgenrecount <- genrecounts %>%
  ggplot(aes(x=gcount,y=moviecount)) +
  geom_col(fill=palette[["movie"]]) + labs(title="Count of Movies by Number of Genres",
        x="Genre Count",
        y="Movie Count")
plotgenrecount
```



This chart shows that most movies have 1-3 genres listed. There are about 1,000 movies with 4+ genres. 8 is the greatest number of genres assigned to a single movie.

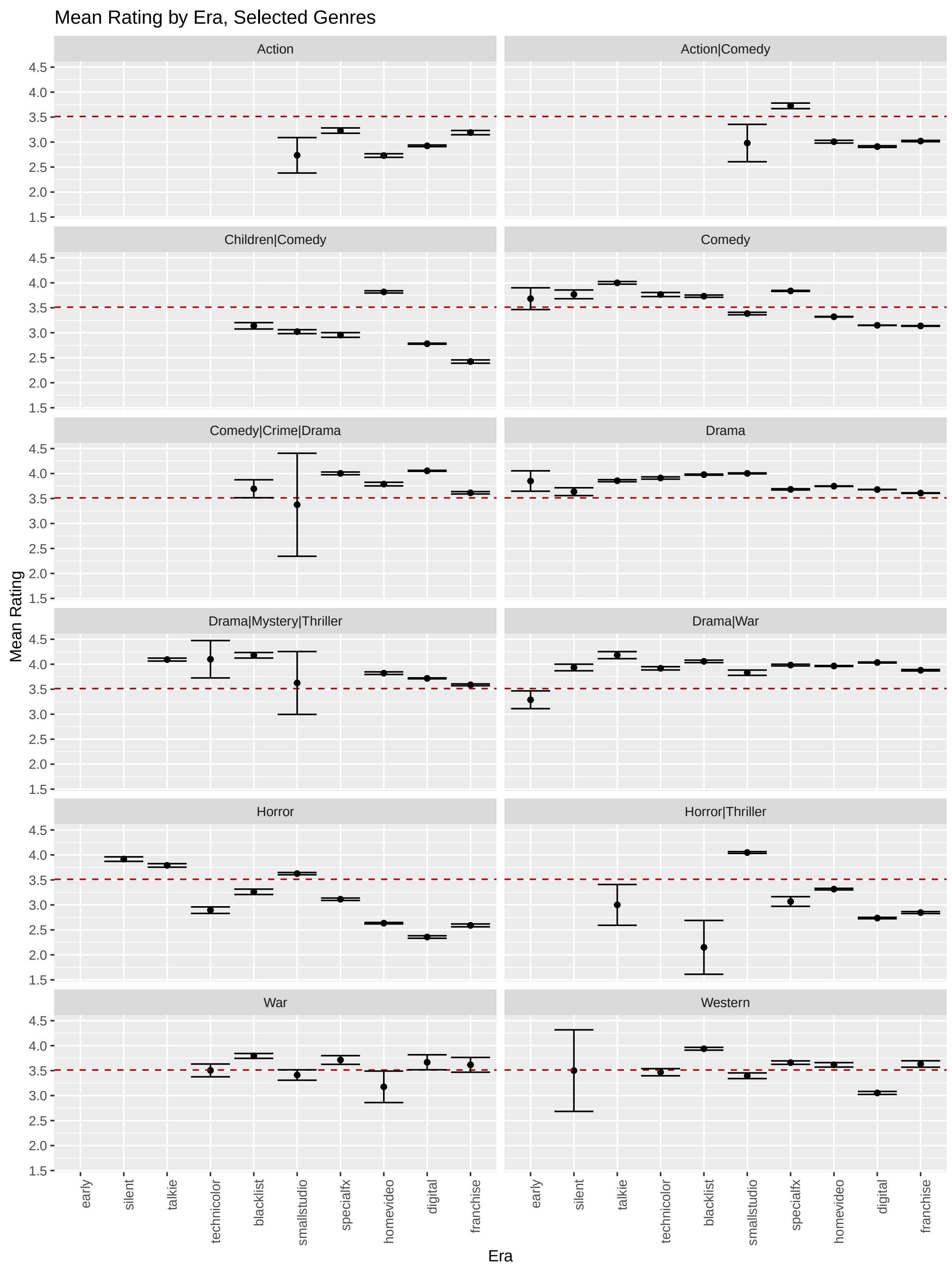
```
#show rating averages by # of genres
plotgenreratings <- genrecounts %>%
  ggplot(aes(x=gcount,y=rateavg)) +
  geom_point() +
  geom_smooth(method = "loess", level=.95) +
   geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  labs(title="Mean Rating by Number of Genres",
        x="Genre Count",
        y="Mean Rating")

plotgenreratings
```



There is a slight correlation of movies with fewer genres and lower mean ratings - but this does not appear to be a very strong effect. A 95% confidence interval includes mu (3.5124652) across all the genre count groups except 1 and 2 genre movies, which fall slightly below average.

We will examine how genre effects the rating - and given what we know about the ratings over time - do the ratings of genres stay the same, or change over time? This requires that the ratings by genres are stratified by each time era defined.

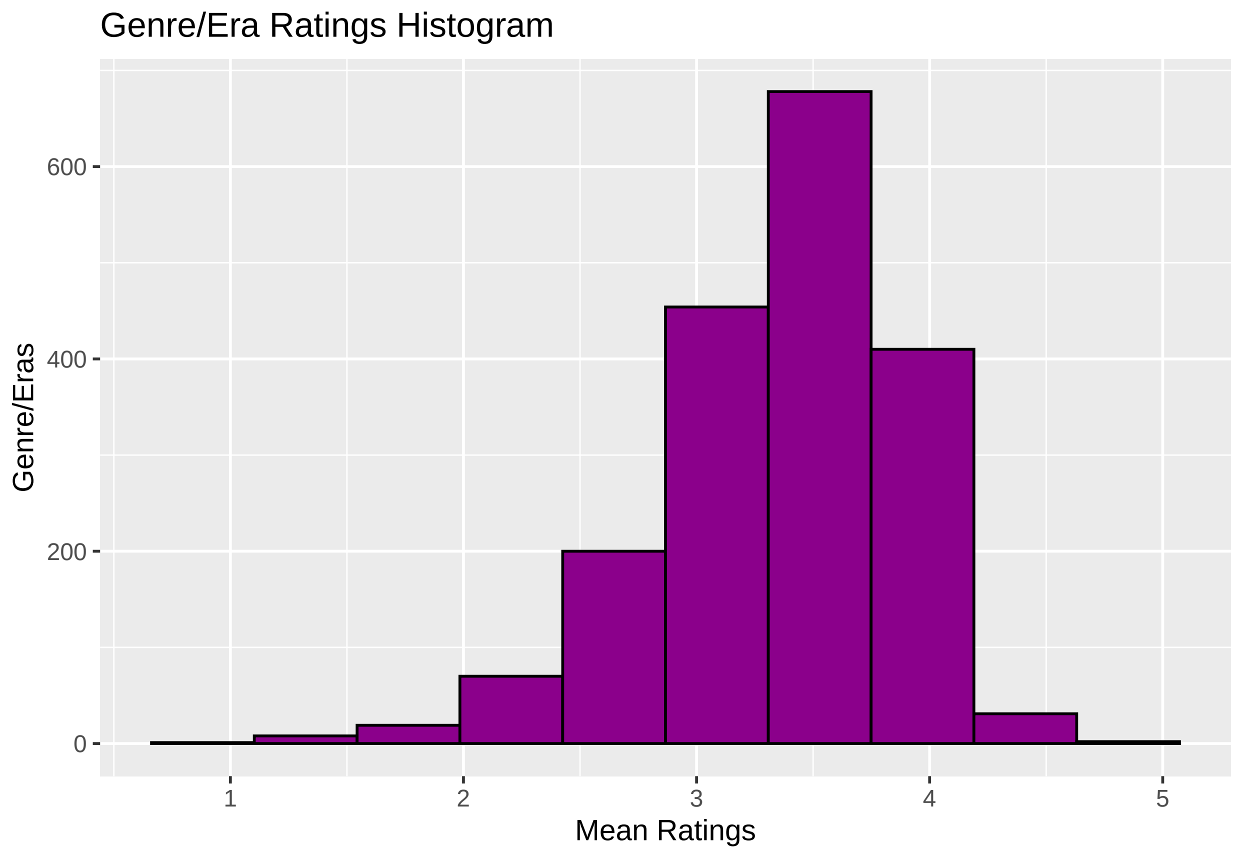
```r
#list some popular genres to examine
trygenres <- c("Comedy|Crime|Drama","Drama|Mystery|Thriller",
               "Children|Comedy","Drama|War","Action|Comedy",
               "Horror|Thriller","Drama","Action","Comedy","War","Western","Horror")


#boxplot of movie ratings by genre
ratingsovertime <- edx_fortify %>%
  filter(genres %in% trygenres) %>%
  group_by(era,genres) %>%
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n()))

ratingsovertime$era_f = factor(ratingsovertime$era, levels=movie_eras)

#show box plot
ratingsovertime %>% ggplot(aes(x = era_f,
                               y = avg,
                               ymin = avg - 2*se,
                               ymax = avg + 2*se)) +
  geom_point() +
   geom_hline(yintercept = mu,
              colour="#BB0000",
              linetype="dashed") +
  geom_errorbar() +
 facet_wrap(~genres, ncol=2) +
 theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating by Era, Selected Genres",
        x="Era",
        y="Mean Rating")
```



It appears that some genres have like Drama|War have stayed consistently above average while others like

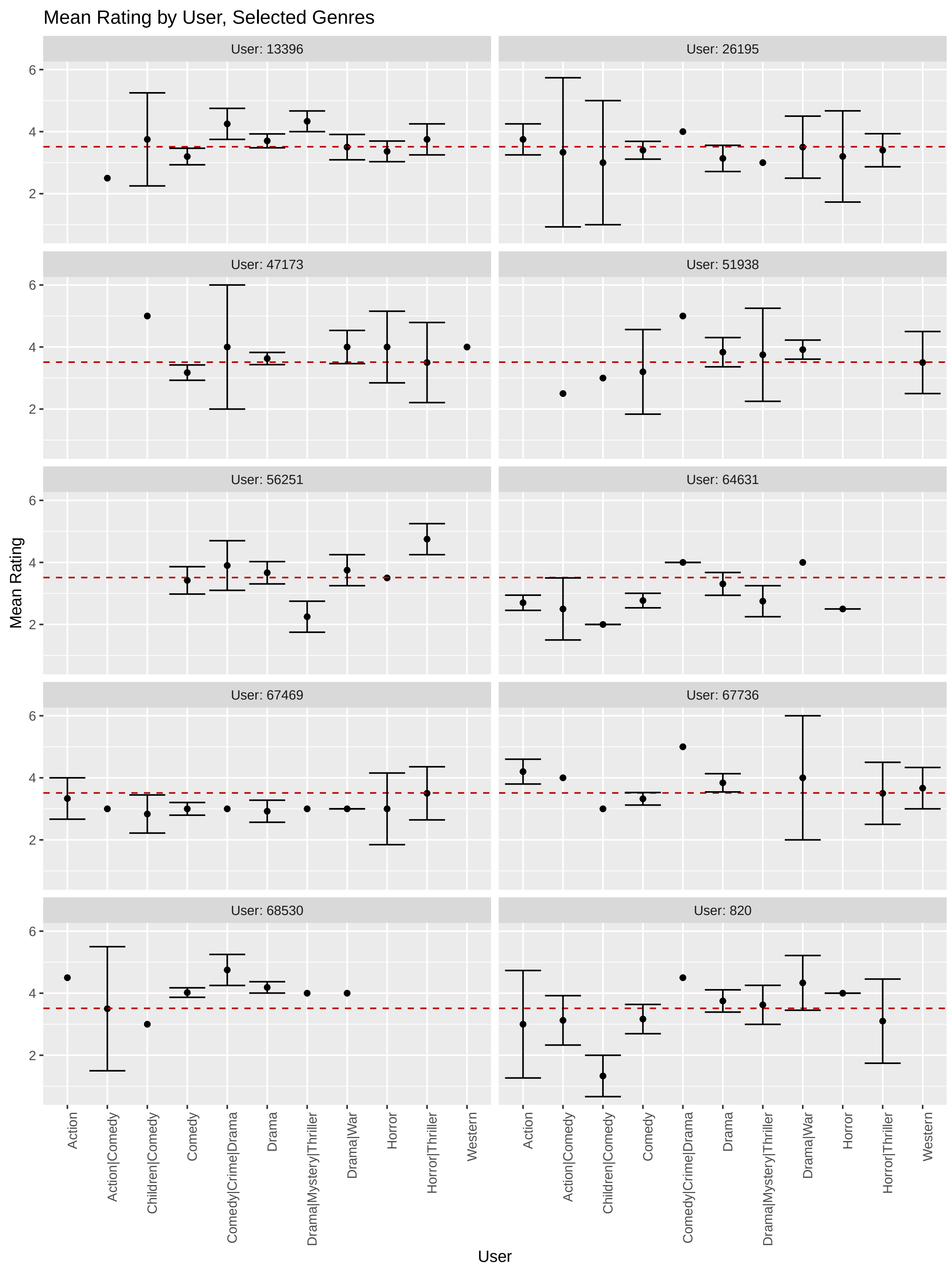
Western and Horror|Thriller have seen periods both above and below the mean, following a similar curve as all movies' average ratings shown earlier. So it may be valuable to consider genre effects to be *era-specific* since it appears some genres become more well-liked or less well-liked over time.

```
#show a histogram
edx_fortify %>%
    group_by(era,genres) %>%
     summarize(b_g = mean(rating))  %>%
     ggplot(aes(b_g)) +
     geom_histogram(bins = 10, color = "black",fill=palette[["genre"]]) +
  labs(title="Genre/Era Ratings Histogram",
        x="Mean Ratings",
        y="Genre/Eras")
```



This chart indicates that many of the genre-era combinations vary from the average. We will investigate various genre related effects in the construction of the model below.

```
#list some popular genres to examine
trygenres <- c("Comedy|Crime|Drama","Drama|Mystery|Thriller",
               "Children|Comedy","Drama|War","Action|Comedy",
               "Horror|Thriller","Drama","Action","Comedy","War","Western","Horror")

#select 10 random users with 200+ ratings
tryusers <- edx_fortify %>%
  group_by(userId) %>%
  summarize(rcount = n()) %>%
  filter(rcount > 200) %>%
  sample_n(10) %>%
  pull(userId)

#boxplot of movie ratings by genre
ratingsoveruser <- edx_fortify %>%
  filter(genres %in% trygenres) %>%
  filter(userId %in% tryusers) %>%
  group_by(userId,genres) %>%
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n()))


#show box plot
ratingsoveruser %>% ggplot(aes(x = genres,
                               y = avg,
                               ymin = avg - 2*se,
                               ymax = avg + 2*se)) +
  geom_point() +
   geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  geom_errorbar() +
 facet_wrap(~paste("User:",userId), ncol=2) +
 theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title="Mean Rating by User, Selected Genres",
        x="User",
        y="Mean Rating")
```

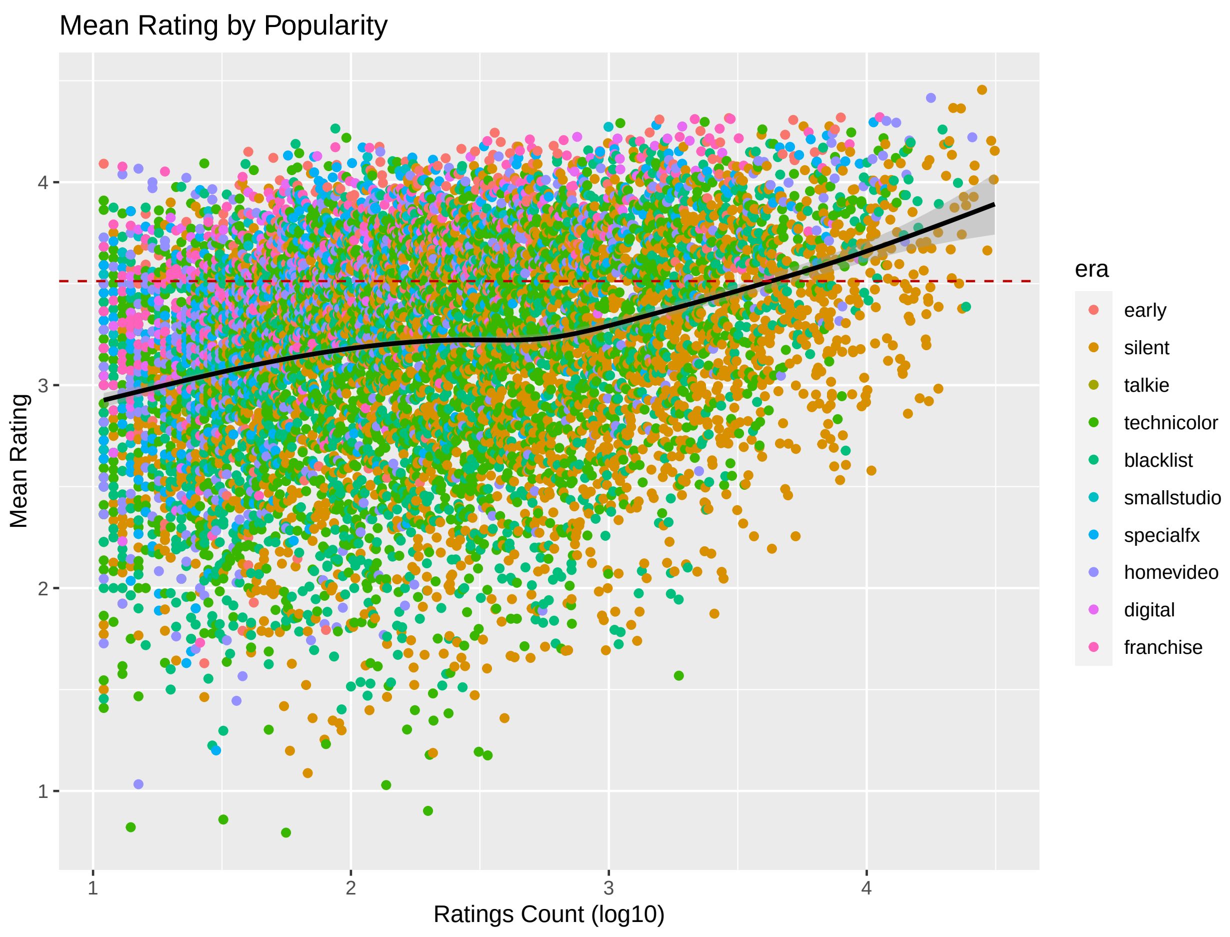


Here we can see some evidence of a genre preferences of users related to the same genres selected above. We will attempt to incorporate a genre-user bias in the model construction process.

Popularity Analysis

Finally, we will examine how popularity, as measured by the number of ratings received, affects the overall movie ratings.

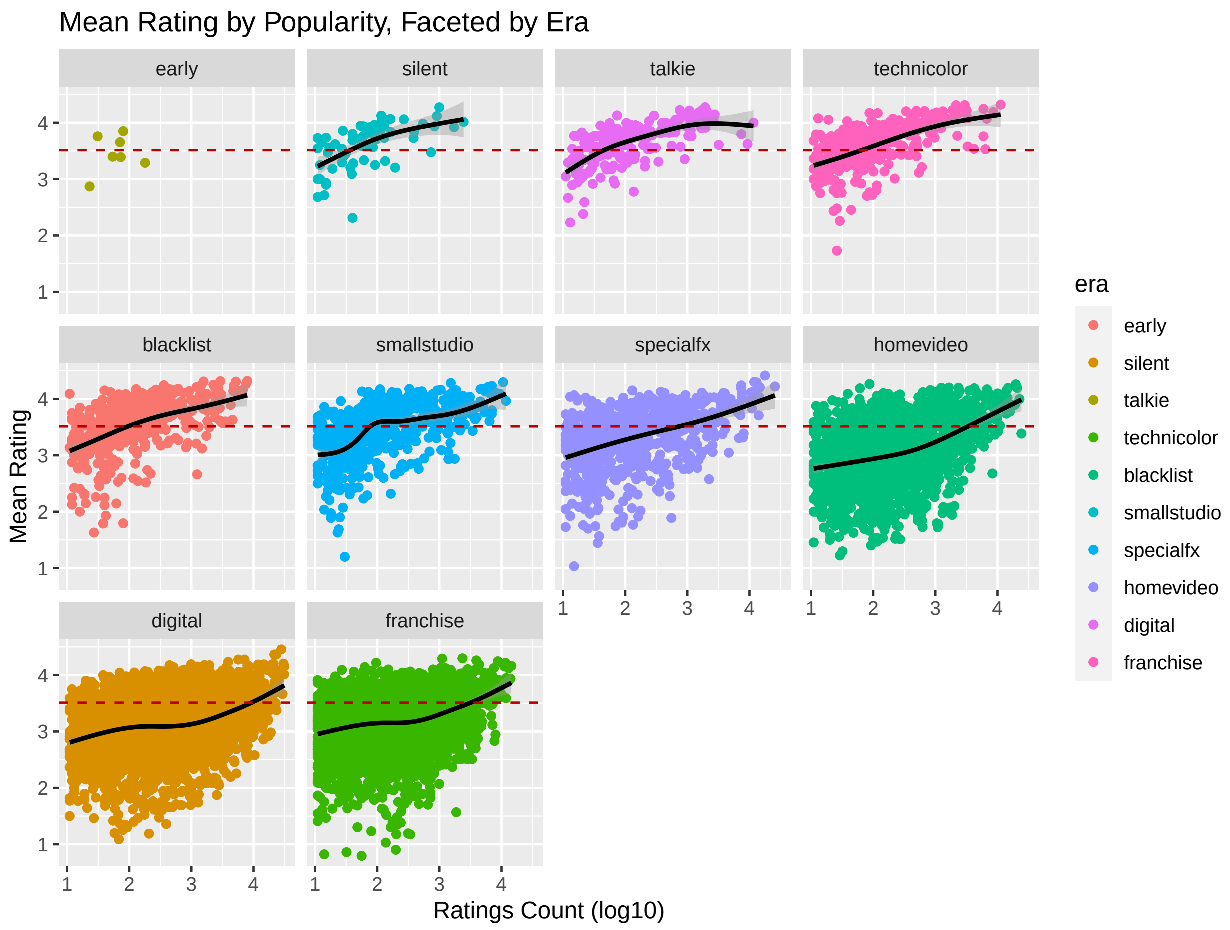
```r
#convert the number of ratings of the movie to a popularity rating
#save the table as popgroups which can be joined to the test set as needed
popgroups <- edx_fortify %>%
          group_by(movieId) %>%
          summarize(popgroup = as.factor(round(log10(n()),1)),
                    pg = as.numeric((round(log10(n()),1))))

#get data
mostrated <- edx_fortify %>%
  group_by(era, movieyear, movieId, title) %>%
  summarise(rateavg = mean(rating),
            ratecount = n()) %>%
  arrange(ratecount) %>%
  filter(ratecount > 10)

#show scatterplot
mostrated %>%
  ggplot(aes(y=rateavg,x=log10(ratecount),color=era)) +
  geom_point() +
  geom_smooth(color="#000000") +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
  labs(title="Mean Rating by Popularity",
       x="Ratings Count (log10)",
       y="Mean Rating") +
  scale_color_discrete(labels=movie_eras)
```



These charts show how popularity is somewhat correlated with higher ratings. As demonstrated by the many intermixed colors, movies from all eras are shown here, and the number of data points is very large.

It is difficult to see if this popularity correlation holds true over time, so we will again facet the data by the eras we established earlier, with the same colors applied. Since the popularity is highly variable, we must create popularity groups that place movies of similar popularity together. The number of ratings per movie vary widely - so we will set the popularity groups (popgroups) to a *log10 scaled count of the number of reviews, rounded to the nearest tenth.* This will be called the *popgroup* of each movie.

```
#get most rated movies by era
mostrated <- edx_fortify %>%
  group_by(era, movieyear, movieId, title) %>%
  summarise(rateavg = mean(rating), ratecount = n()) %>%
  arrange(ratecount) %>%
  filter(ratecount > 10)

mostrated$era_f = factor(mostrated$era, levels=movie_eras)

#get mean ratings faceted by era
mostrated %>%
  ggplot(aes(y=rateavg,x=log10(ratecount),color=era)) +
  geom_point() +
  geom_smooth(color='#000000') +
   geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
 labs(title="Mean Rating by Popularity, Faceted by Era",
        x="Ratings Count (log10)",
        y="Mean Rating") +
  scale_color_discrete(labels=movie_eras) +
  facet_wrap(~era_f)
```

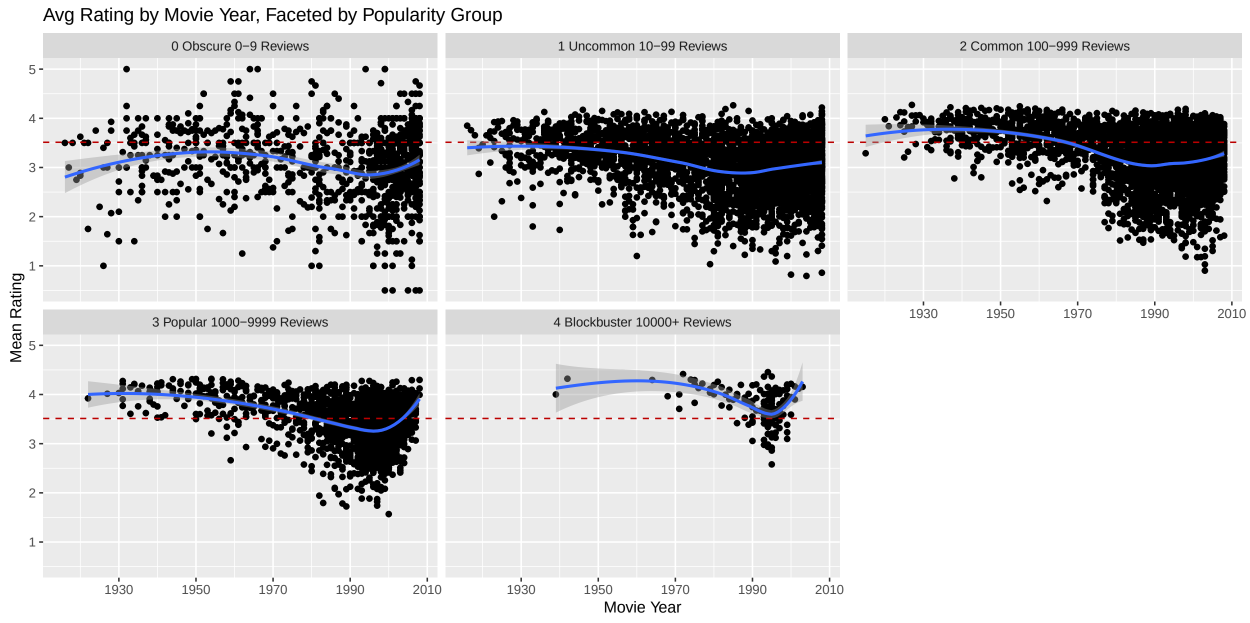


```
#determine correlation coefficient
popularity_cor <- cor(mostrated$rateavg,log10(mostrated$ratecount))
pop_cor_pct <- paste("Correlation Coef.:",round(popularity_cor*100,2),"%")

#show the correlation
paste("Correlation:",round(popularity_cor*100,2),"%")
```

```
## [1] "Correlation: 27.42 %"
```

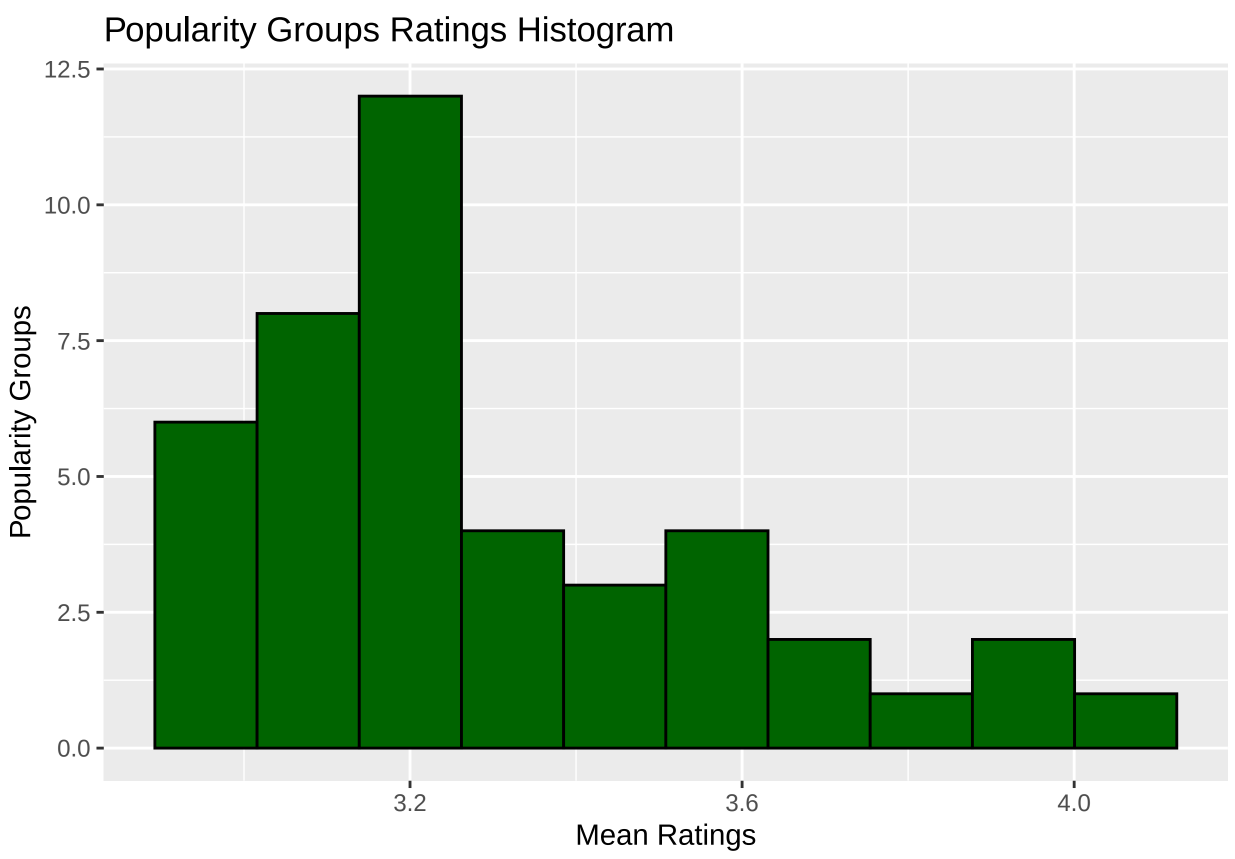
From this chart it appears that across all eras, movies that are more popular (receive more reviews) also tend to receive higher ratings. (Correlation Coef.: 27.42 %) This makes sense - a movie getting very low numbers of reviews is probably not as well known, and may be a smaller budget movie with less special effects or less popular actors.

```
#setup variables
popgroup_index <- c(0,1,2,3,4)
popgroup_names <- c('Obscure','Uncommon','Common','Popular','Blockbuster')
popgroup_desc <- c('0-9 Reviews','10-99 Reviews','100-999 Reviews','1000-9999 Reviews','10000+ Reviews'

popgroup_info <- data.frame(ratepop = popgroup_index,
                            popname = popgroup_names,
                            popdesc = popgroup_desc)

#create data for rating averages faceted by popularity
firstratedates <- edx_fortify %>%
  group_by(era, movieyear, movieId, title) %>%
  summarise(rateavg = mean(rating),
            yearavg = mean(movieyear),
            ratecount = n()) %>%
  arrange(rateavg)  %>%
  mutate(ratepop = floor(log10(ratecount))) %>%
  left_join(popgroup_info,by = 'ratepop')

#plot a scatterchart
firstratedates %>% ggplot(aes(y=rateavg,x=yearavg)) +
  geom_point() +
  geom_hline(yintercept = mu,
             colour="#BB0000",
             linetype="dashed") +
 geom_smooth(method="loess") +
  facet_wrap(~ paste(ratepop,popname,popdesc))  +
  labs(title="Avg Rating by Movie Year, Faceted by Popularity Group",
       x="Movie Year",
       y="Mean Rating")
```



Looking at the same data from another perspective - we can see that the ratings over time follow the similar year by year S curve already discussed. But each popularity group averages a bit higher. While the obscure group has an curve entirely below the average- the blockbuster group has a curve that is entirely above the average. Based on these observations - it appears that adding an adjustment to the rating based on popularity would be justified.

```r
#show a histogram
edx_fortify %>%
     left_join(popgroups, by="movieId") %>%
     group_by(popgroup) %>%
     summarize(b_p = mean(rating))  %>%
     ggplot(aes(b_p)) +
     geom_histogram(bins = 10, color = "black",fill=palette[["popular"]])+
  labs(title="Popularity Groups Ratings Histogram",
        x="Mean Ratings",
        y="Popularity Groups")
```



This histogram indicates that many of the popularity groups would deviate from the overall mean. We will test a popularity effect in the construction of the model.

Model Construction & Testing

To facilitate testing and validation of our model, we will create several partitions of the edx_fortify data. The validation "hold out" data represents unknown data, so it should be used only to validate our models, after all tuning and optimizations are completed using the 9M rows of edx data we have available. We will build the model and test the RMSE on these internal partitions first. A 90%/10% training/test split will best simulate the proportion of training and test data in the final model.

We have already fortified the data in the edx_fortify data frame, by adding movieyear and other variables needed by the model- so these will be available to both sides of these partitions. The partition logic below uses the same basic partition logic as provided, which will be used on the final edx/validation data. This ensures that the test data contains only userIds and movieIds that appear in the training data.

```r
makepartitions <- function(dataset,rows,testportion,numparts,seed){
   returnobj <- NULL
    returnobj <- data.frame()

    set.seed(seed, sample.kind="Rounding")
        if(rows != -1){
          dataset <- dataset %>% sample_n(rows)

        }

    edx_part_train_index <- createDataPartition(y = dataset$rating,
                                                 times = numparts,
                                                 p = testportion,
                                                 list = FALSE)

    edx_part_train <- edx_fortify[-edx_part_train_index,]
    edx_part_temp <- edx_fortify[edx_part_train_index,]

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

    # Add rows removed from validation set back into edx set
    edx_part_removed <- anti_join(edx_part_temp, edx_part_valid)
    edx_part_train <- rbind(edx_part_train, edx_part_removed)

    rm(edx_part_temp)
    rm(edx_part_removed)

  #return the RMSE, prediction, penalty factor, & benchmark time
      returnobj <-list(
                    trainindex=edx_part_train_index,
                    trainDS=edx_part_train,
                    testDS=edx_part_valid
                    )

      class(returnobj) <- "partition"
      returnobj

}

#We will first choose 3 random seed values for the three partitions
#this is important to make sure we are not overfitting the training parameters
seed1 <- 981
seed2 <- 523
seed3 <- 267

#each of these 90/10 partitions will have different rows
```

```
#in the training and test data (with some overlapping)
part9010_A <- makepartitions(edx_fortify,-1,.1,1,seed1)
part9010_B <- makepartitions(edx_fortify,-1,.1,1,seed2)
part9010_C <- makepartitions(edx_fortify,-1,.1,1,seed3)
```

Building a Machine Learning Model

In this section we will begin constructing a model based on the EDA we have already conducted. In each step we will examine how the overall effect has been impacted by subtracting all previous effects. We will test at each step how the model is impacting the RMSE on the training set. If the RMSE is improved (reduced) over the previous step, it should indicate that this is a viable effect to use as part of the model.

We will test each successive step using only the training/test data sets we just created. Our 90%/10% partitions should give a good ballpark estimate of the accuracy since the proportion of training and test data will be the same as in our final model. We will build out the model using partition A, then train it using all three (A,B,C), and finally optimize the lambda penalty value, and perform the final test against our complete edx/validation hold out data.

As we build the model - several ranges of possible variables will be tried, and the best values will bee used after the model is encapsulated.

The variables will include

- weight: a multiplier added to each individual bias factor that increases or decreases the weight of that factor
- cutoff: minimum number counts to use as a low-cut/ high-pass filter for each bias factor as needed
- penalty: a regularization penalty to apply to the final data to prevent over fitting

To determine an appropriate weight or cutoff, we will conduct a loop operation on each effect and return the parameter(s) that produce the lowest RMSE.

Establishing a Baseline

A naive Bayes approach could use the mean μ of all ratings as a best guess for any unknown movie/user combination. Throughout the model construction, this variable will be called “mu”. A model that predicts mu for every user-movie combo will be referred to as the baseline or naive Bayes model. We will also record how long each model step takes to make the predictions as a benchmark, since performance is an important factor to consider in processor-intensive computing operations. Each step is named for the last effect added, and incorporates all previous models.

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

#partition a training and test data set from the fortified edx
#data that has already been partitioned
#we will use the "A" partition for the construction of the model
trainDS <- part9010_A$trainDS
testDS <- part9010_A$testDS

#IMPORTANT - Resetting mu here to the training set
#this is the baseline to which all effects will be added/subtracted
```

```r
mu <- mean(trainDS$rating)

#create a vector of predictions that are all the same (mu) value
#modstart and modfinish will time the routine for benchmarking
modstart <- Sys.time()
predictions <- rep(mu, nrow(trainDS))
modfinish <- Sys.time()

#get the RMSE of the naive model
naive_rmse <- RMSE(testDS$rating,predictions)

#determine the operation speed. This operation will be done for each model
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#modelstep will increment for each new test
modelstep <- 0
modelstep <- modelstep + 1

#show the RMSE results in a kable
rmse_results <- data_frame(step=modelstep,
                           method = "Baseline",
                           RMSE = naive_rmse,
                           benchmark = benchmark,
                                     weight = 1,
                                     lowcut = 0)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.061591	0.02 secs	1	0

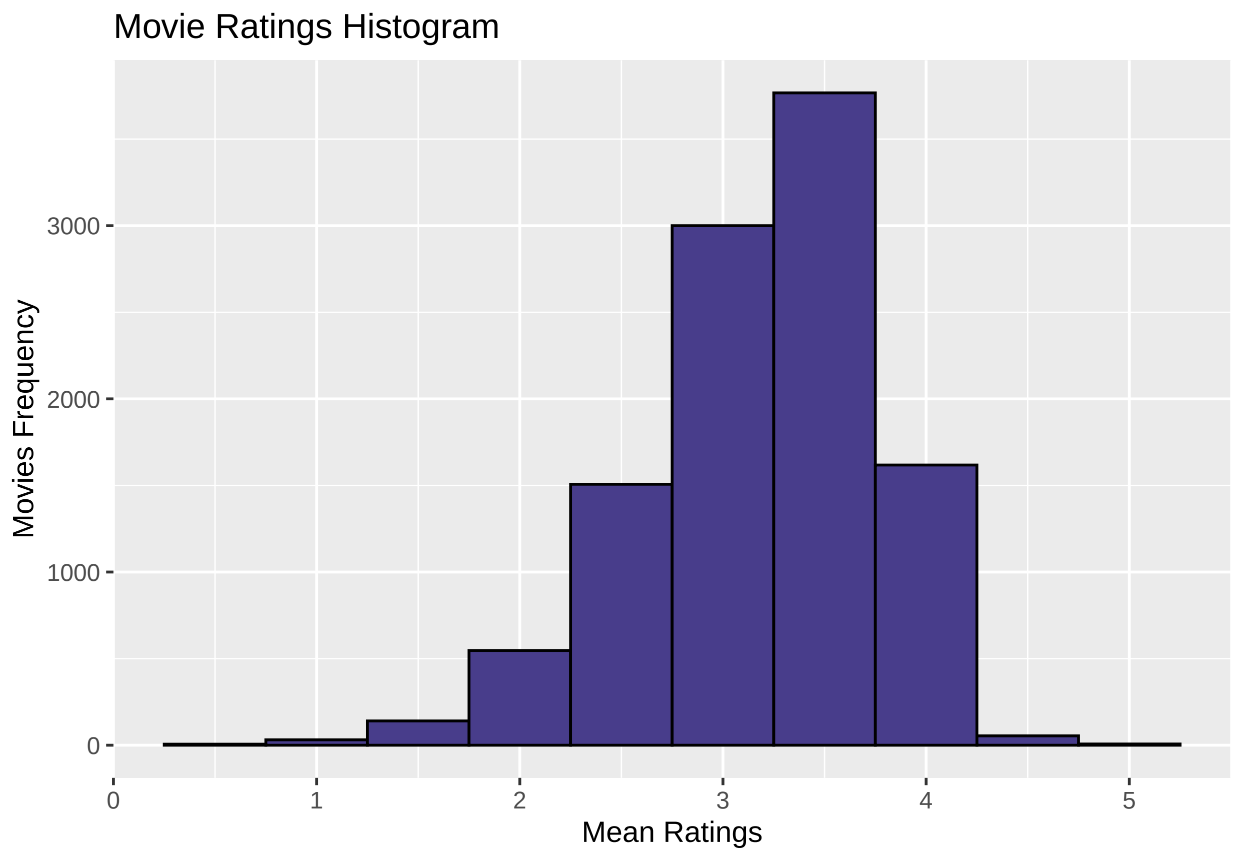
The average rating across our training set μ is 3.5125317. Using this average as our guess for every movie, we get an RMSE of about 1.0615912. This means that our guesses would be off by an average of about 1.0615912 stars. The goal is to reduce this number as much as possible.

Movie Effect

Taking into account the fact that some movies get consistently higher or lower ratings, we will add a movie-effect based approach. This will allow our model to offset from just the average, modifying the predicted rating for movies that are usually rated higher or lower. We will set the weight of each variable to 1 (the same) to start, The weights may be slightly modified for each effect during the final model construction. The weight factor will add more or less to the prediction change for each effect individually. We will also add a single penalty effect which will be the same for all factors.

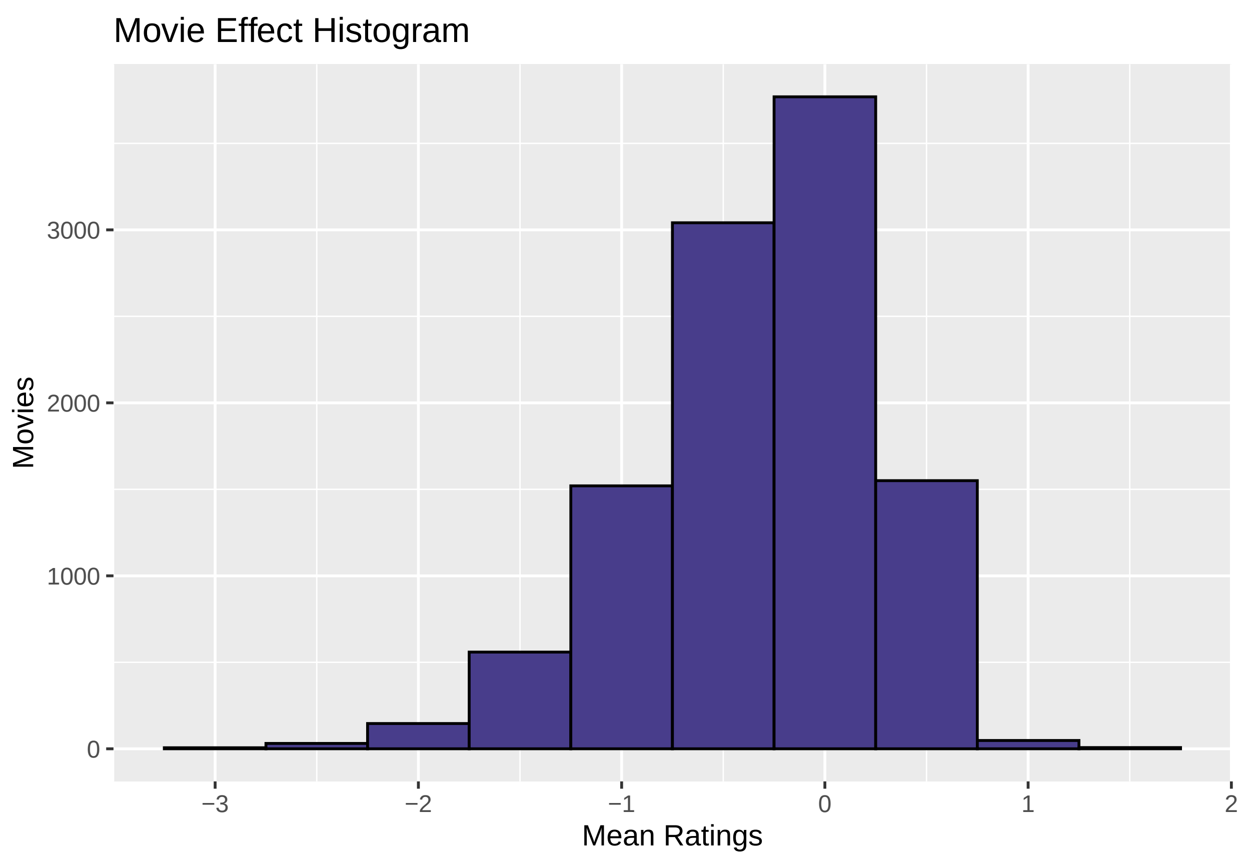
```r
#setup variables
movieweight <- 1

#movie rating averages
movie_avgs <- trainDS %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating))
```

```
#show a histogram of raw movie ratings
movie_avgs %>%
  ggplot() +
  geom_histogram(aes(b_i),bins = 10, color="#000000", fill=palette[["movie"]])  +
  labs(title="Movie Ratings Histogram",
        x="Mean Ratings",
        y="Movies Frequency")
```



Revisiting some of the raw ratings histograms from the data analysis section, we will now explore how each of the bias effects shown above will be combined into the final model.

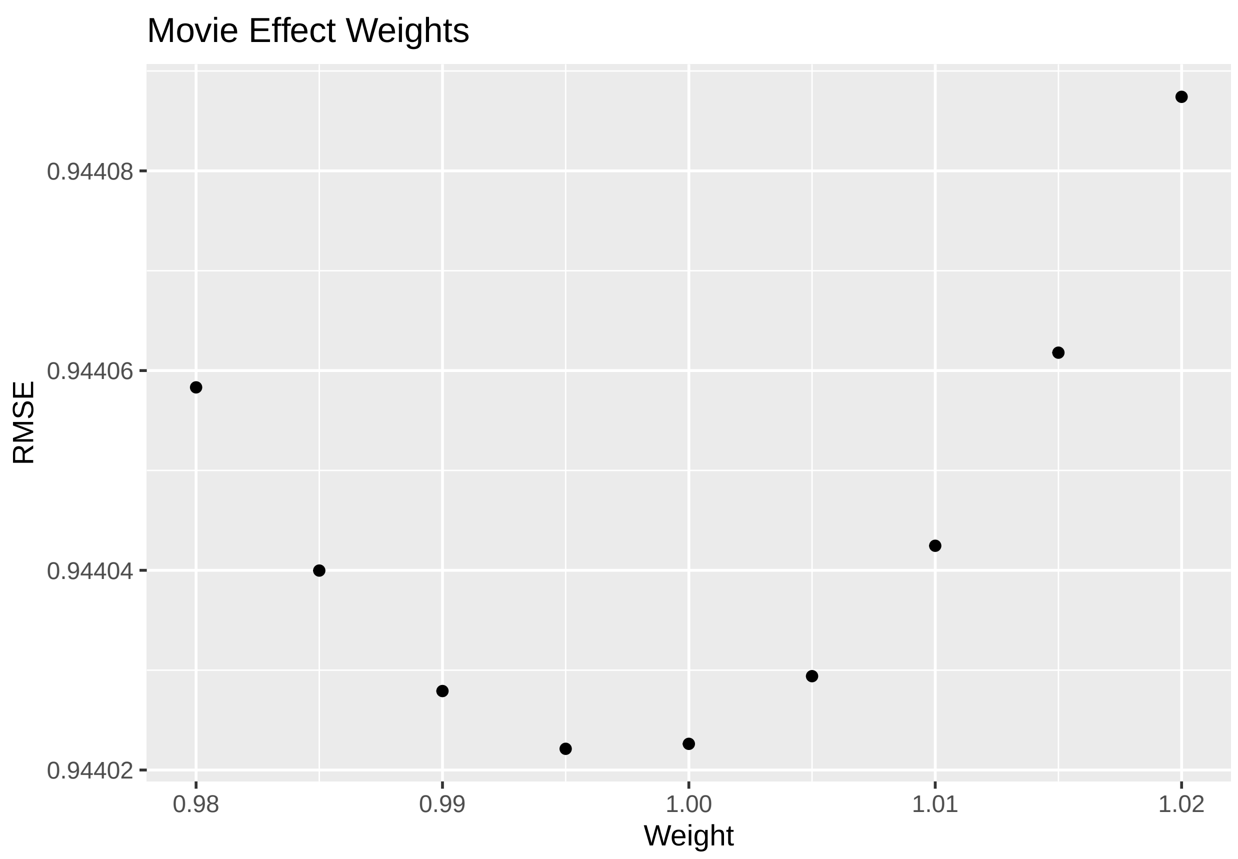
```
#determine the mean movie rating
movie_avgs <- trainDS %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu) * movieweight)

#plot a histogram
movie_avgs %>%
  ggplot(aes(b_i)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["movie"]]) +
     labs(title="Movie Effect Histogram",
        x="Mean Ratings",
        y="Movies")
```



The plot above shows how our movie effect (b_i) will move the ratings on average. This seems to be a powerful predictor, with about 4,000 movies losing .5 to 1 star and about 2,000 gaining .5 to 1 star.

```
#setup variables
movieweights <- seq(.980,1.02,by=.005)
movie_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
movie_avgs_best <- c()
last_RMSE <- 1000

for (w in movieweights){

modstart <- Sys.time()

movie_avgs <- trainDS %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu) * w)

#using the testDS set only to make a prediction.
predicted_ratings <- mu + testDS %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i

modfinish <- Sys.time()
```

```r
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#check prediction vs the results in the testDS set
model_1_rmse <- RMSE(testDS$rating, predicted_ratings)

if(model_1_rmse < last_RMSE){
      movie_avgs_best <- c()
      movie_avgs_best <- movie_avgs
      last_RMSE <- model_1_rmse
    }


modelname <- paste("Movie Effect Model")

#checking my prediction vs the results in the testDS set
movie_results <- bind_rows(movie_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_1_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0
                                     ))
}

#plot the results of the tuning operation
movie_results %>%
  ggplot(aes(x=weight,y=RMSE)) +
     geom_point() +
     labs(title="Movie Effect Weights",
        x="Weight",
        y="RMSE")
```



```r
#keep the best RMSE
movie_best <-  movie_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, movie_best)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0

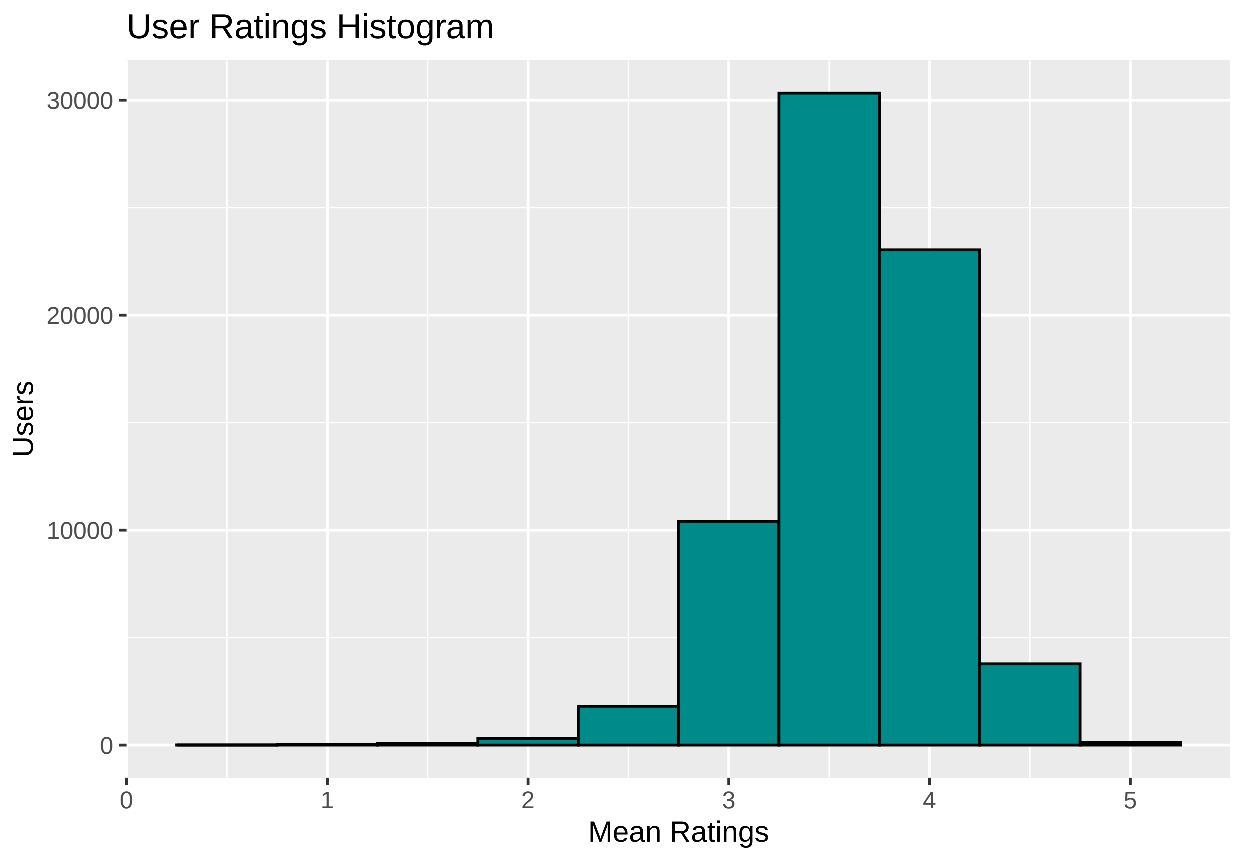
```r
movie_best_RMSE  = movie_best$weight
movieweight = movie_best$weight
```

As expected this powerful predictor improves our guess by a significant margin. The lowest RMSE of 0.995 is observed at a weight of 0.995.

User Effect

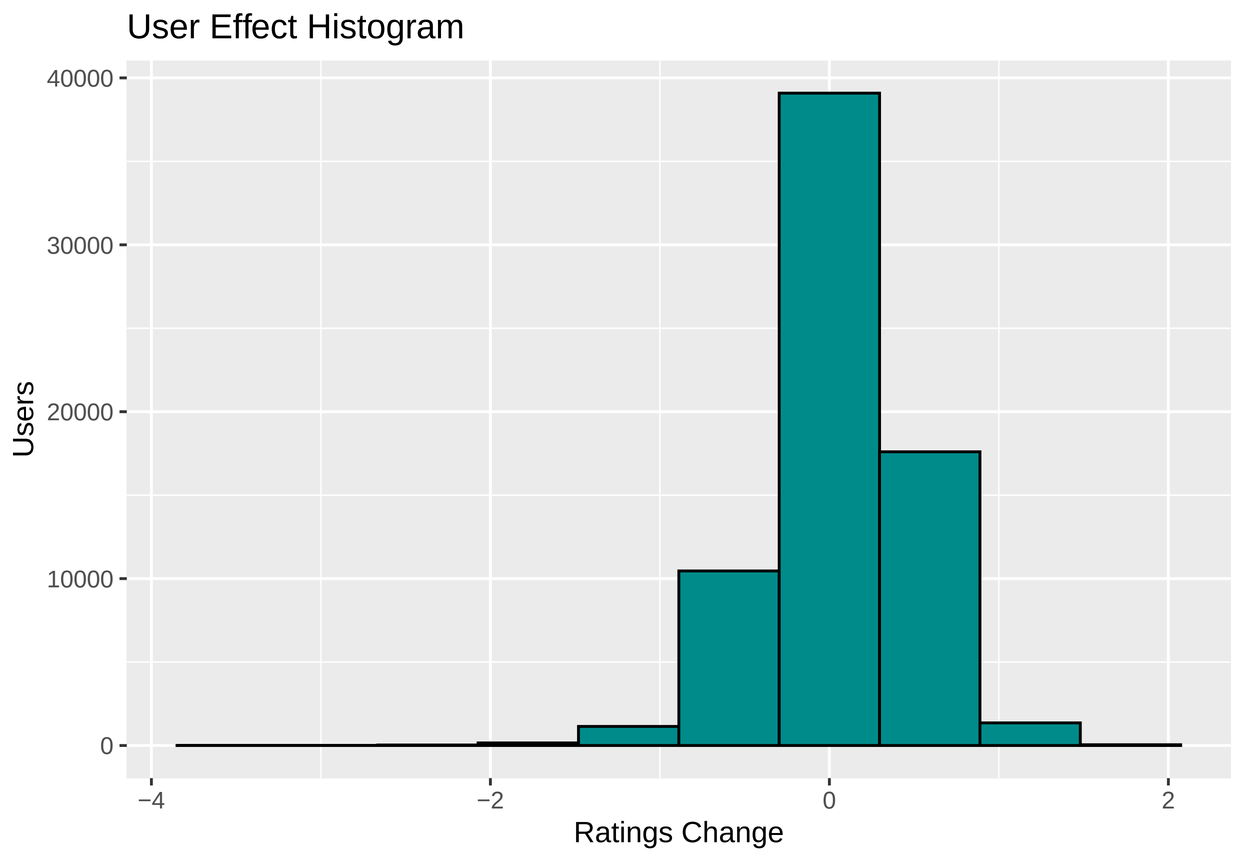
In addition to the Movie Effect, we also should consider User rating behavior. If this user is, on average, a higher or lower rater then we can use that.

```
#determine the mean user rating
trainDS %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating)) %>%
     ggplot(aes(b_u)) +
     geom_histogram(bins = 10, color = "black",fill=palette[["user"]]) +
     labs(title="User Ratings Histogram",
        x="Mean Ratings",
        y="Users")
```



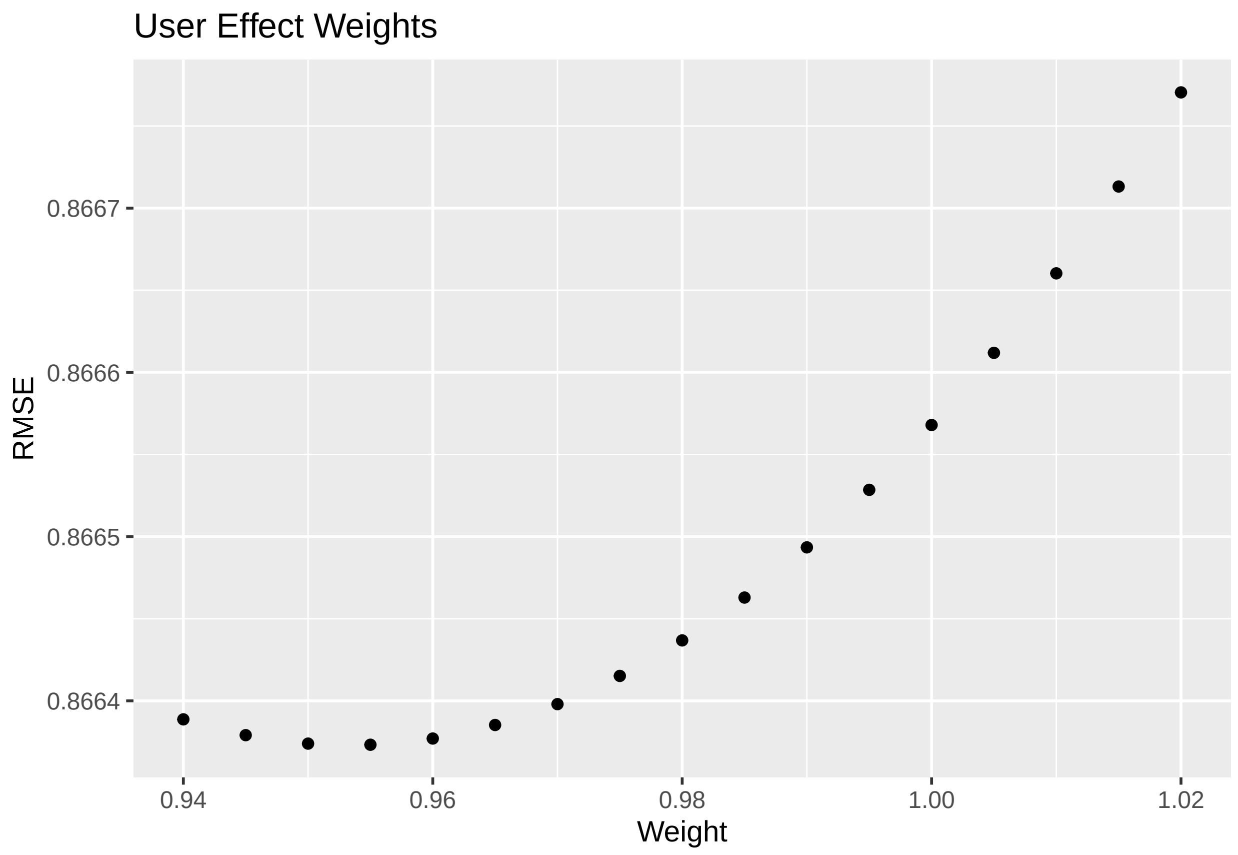
This chart shows the rating behavior of users. The distribution peaks around 3.5 as expected, but there are some users who usually rate movies higher/lower than mu. Nearly 20,000 users have average ratings closer to 4 than 3. The Movie Effect must be removed as a confounder from the User Effect average to produce an accurate bias calculation.

```
#setup variables
userweight <- 1

#determine the mean user rating after deducting prior effects
user_avgs <- trainDS %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_i) * userweight)

#plot a histogram
user_avgs %>%
  ggplot(aes(b_u)) +
     geom_histogram(bins = 10, color = "black",fill=palette[["user"]]) +
     labs(title="User Effect Histogram",
        x="Ratings Change",
        y="Users")
```



The plot above shows how the User Effect bias (b_u) will affect the ratings on average. After accounting for the movie effect, we will see 25,000+ users that will will be shifted up or down by about .5 to 1 star. By repeating the above process we can improve on the model. We will take both the movie effect, and the user effect. Our multi-variable offset should match even better.

```
#setup variables
userweights <- seq(.940,1.02,by=.005)
user_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
user_avgs_best <- c()
last_RMSE <- 1000

for (w in userweights){
modstart <- Sys.time()

#determine the mean user rating after deducting prior effects
user_avgs <- trainDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_i) * w)

#predict ratings on the test set
predicted_ratings <- testDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     mutate(pred = mu + b_i + b_u) %>%
     .$pred

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

modelname <- paste("User Effect Model")

#calculate the RMSE
model_2_rmse <- RMSE(testDS$rating, predicted_ratings)


if(model_2_rmse < last_RMSE){
      user_avgs_best <- c()
      user_avgs_best <- user_avgs
      last_RMSE <- model_2_rmse
   }


#show the results
user_results <- bind_rows(user_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_2_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0
                                     ))
}

#plot the RMSE curve
user_results %>%
  ggplot(aes(x=weight,y=RMSE)) +
     geom_point() +
     labs(title="User Effect Weights",
        x="Weight",
        y="RMSE")
```



```r
#kep the best RMSE
user_best <-  user_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, user_best)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0

```r
user_best_RMSE <- user_best$RMSE
userweight = user_best$weight
```

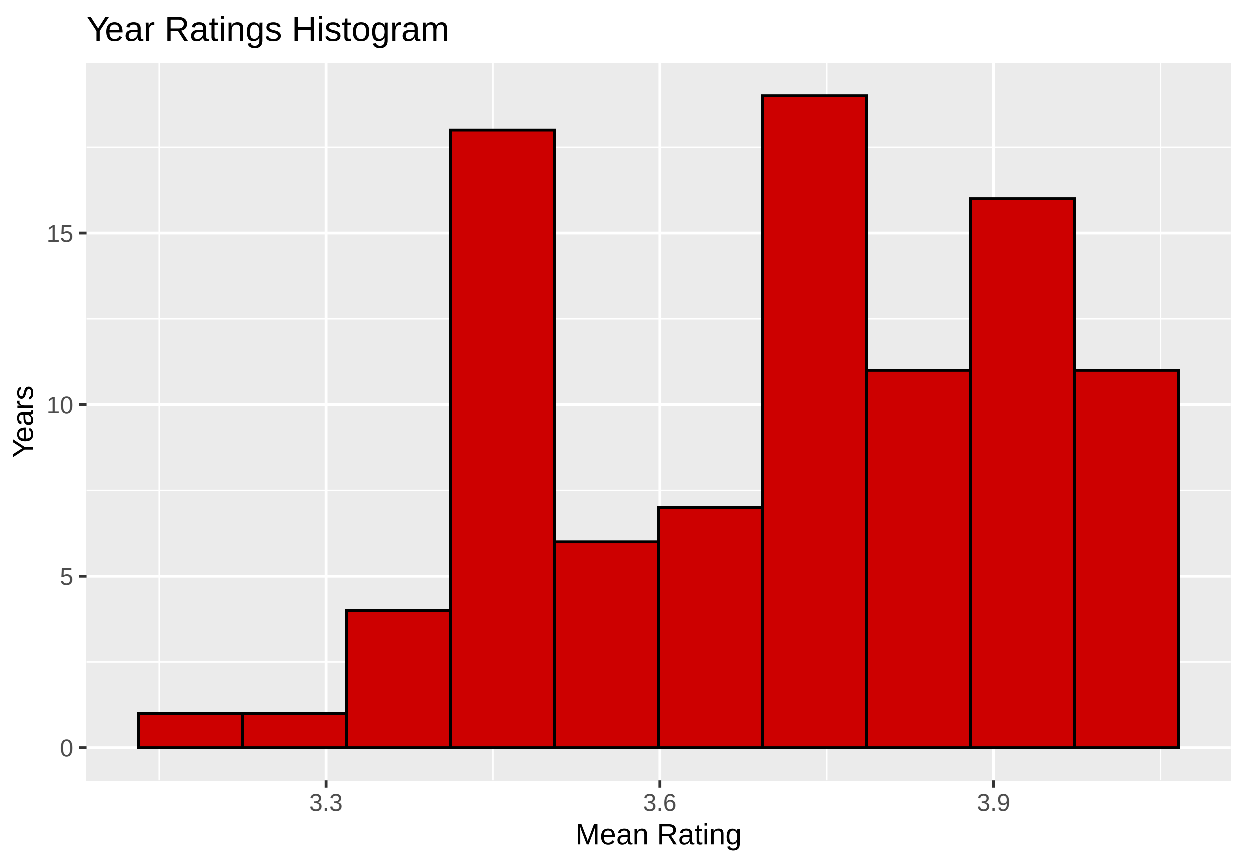
As expected, this model improves on the prior model which only considered the movie effects. The lowest RMSE 0.8663733 is observed at a weight of 0.955

- Please note, The Movie + User effect code above was derived from the "Introduction to Data Science" text by Rafael Irizarry, and is not entirely my original work. The addition of weight parameters is my modification to the code above. The remainder of this model represents my original work toward solving this problem.

Year Effect

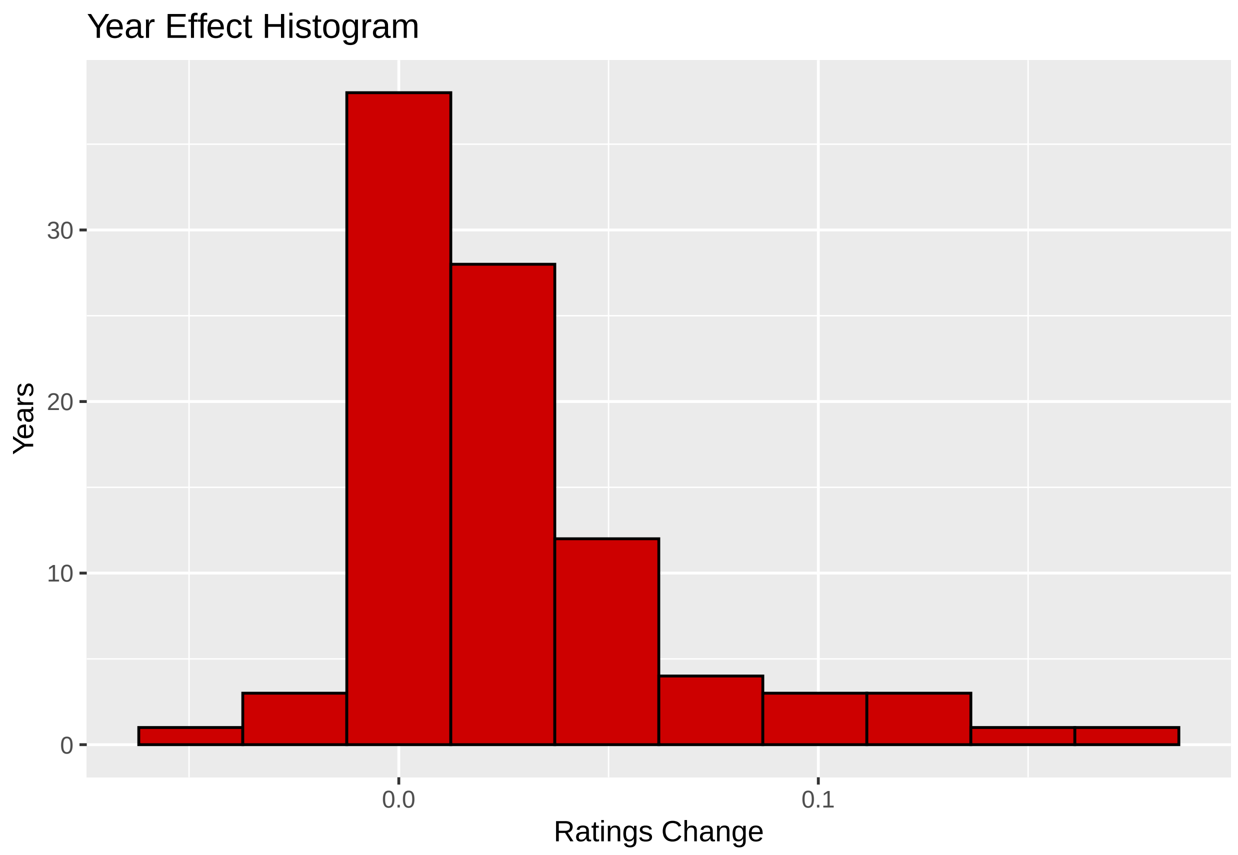
It's clear that adding additional predictors improves the accuracy of our models. Now we will also consider how the release year of the movie affects the rating.

```r
#setup variables
yearweight <- 1

#determine the mean movie rating by year
trainDS %>%
     group_by(movieyear) %>%
     summarize(b_y = mean(rating))  %>%
     ggplot(aes(b_y)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["year"]]) +
     labs(title="Year Ratings Histogram",
        x="Mean Rating",
        y="Years")
```



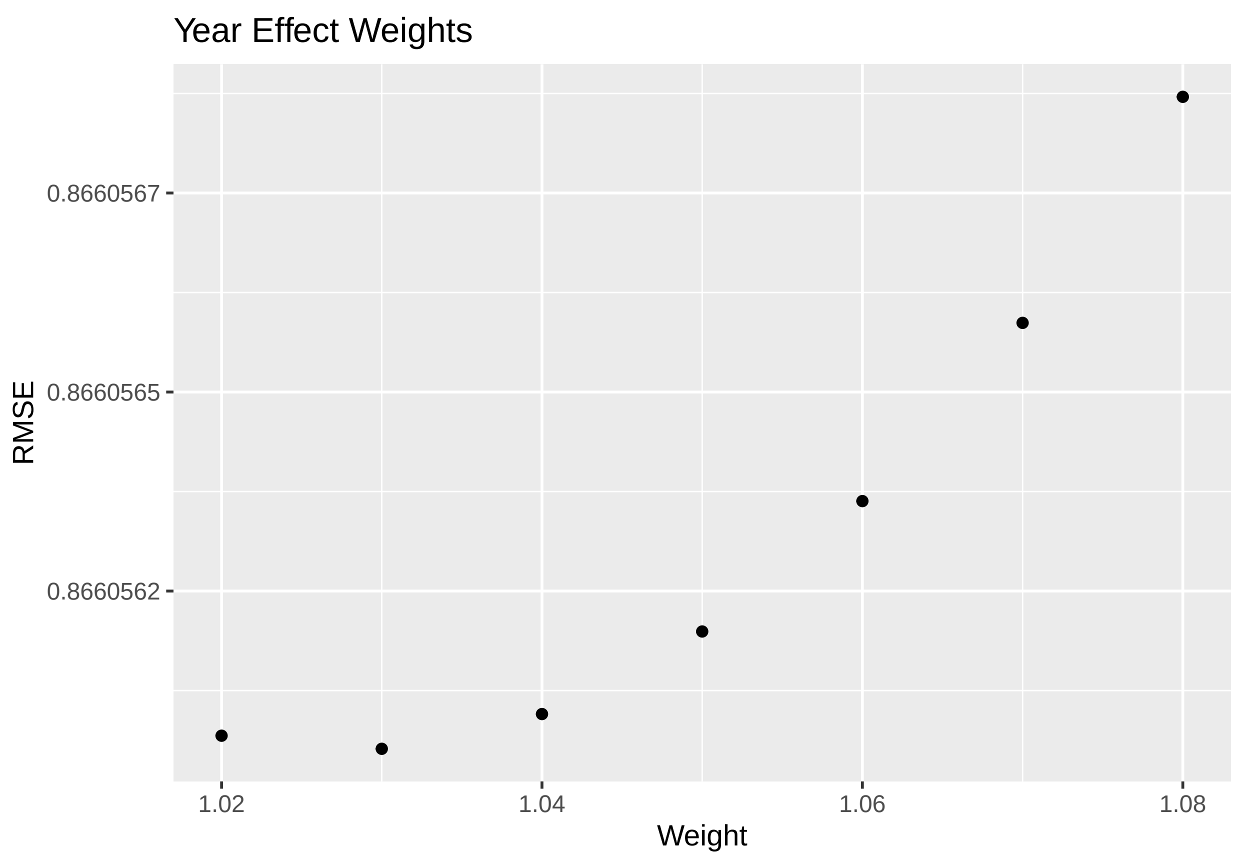
This chart (also shwon in the EDA section) shows that many years have above or below average ratings between 3.2 and 4.0. We will remove the movie and user effects from the year effect to prevent confounding.

```r
#determine the mean movie rating by year adjusted for prior effects
year_avgs <- trainDS %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     group_by(movieyear) %>%
     summarize(b_y = mean(rating - mu - b_i - b_u) * yearweight)

#plot a histogram
year_avgs %>% ggplot(aes(b_y)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["year"]]) +
     labs(title="Year Effect Histogram",
        x="Ratings Change",
        y="Years")
```



After removing the user and movie effects, the ratings will be affected by the year effect (b_y) as shown above. It appears that at least 50 of the years would see a slight gain of ~ .05 in rating estimate, and fewer than 5 years will see a small reduction in the ratings given of .05 or less.

```
#setup variables
yearweights <- seq(1.02,1.08,by=.01)
year_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
year_avgs_best <- c()
last_RMSE <- 1000

for (w in yearweights){

modstart <- Sys.time()
year_avgs <- trainDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     group_by(movieyear) %>%
     summarize(b_y = mean(rating - mu - b_i - b_u) * w)

#make predictions on the test set
predicted_ratings <- testDS %>%
     mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs, by='movieyear') %>%
     mutate(pred = mu + b_i + b_u + b_y) %>%
     .$pred

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#calculate the RMSE
model_3_rmse <- RMSE(testDS$rating, predicted_ratings)

if(model_3_rmse < last_RMSE){
      year_avgs_best <- c()
      year_avgs_best <- year_avgs
      last_RMSE <- model_3_rmse
   }

modelname <- paste("Year Effect Model")

#show the results
year_results <- bind_rows(year_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_3_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0 ))

}

#plot the RMSE curve
year_results %>%
  ggplot(aes(x=weight,y=RMSE)) +
     geom_point() +
     labs(title="Year Effect Weights",
        x="Weight",
        y="RMSE")
```



```
#keep the best RMSE
year_best <-  year_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, year_best)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0

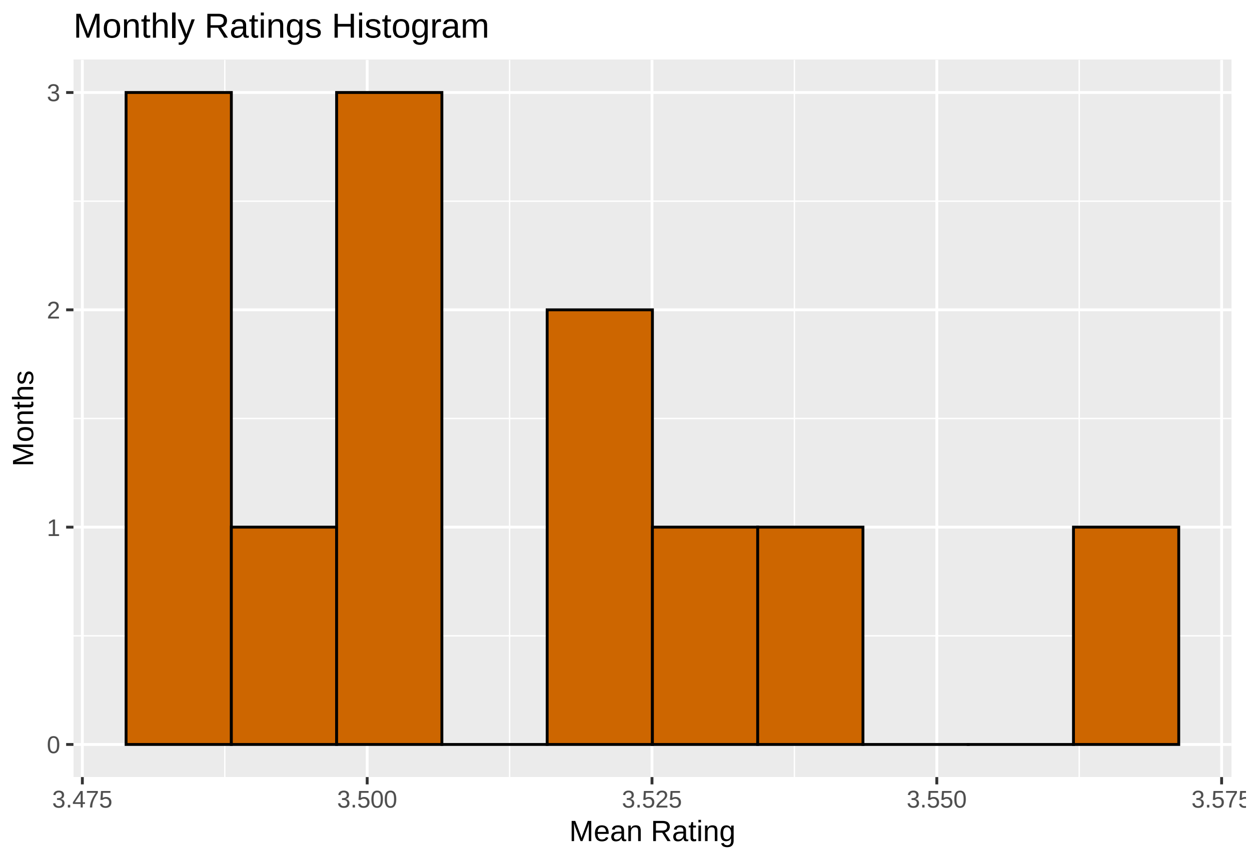
```
year_best_RMSE <- year_best$RMSE
yearweight = year_best$weight
```

The movie year provides a minor improvement in RMSE to 0.8660561 when the year bias is assigned a relative weight of 1.03.

Rating Month Effect

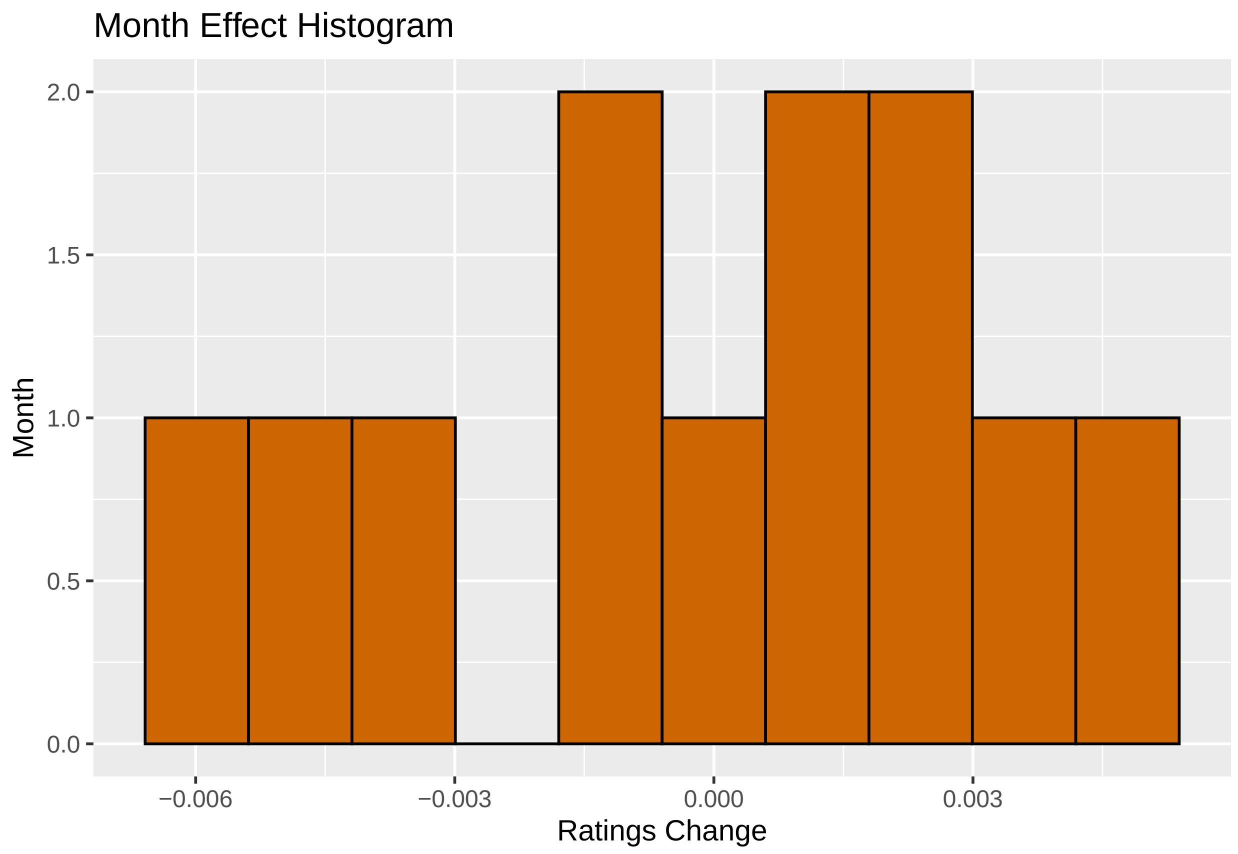
As we explored above, the month in which the movie is rated may give a clue to its rating. Let's consider how this effect will impact the overall ratings given:

```r
#setup variables
monthweight <- 1

#determine the mean movie rating by year
trainDS %>%
     group_by(ratemonth) %>%
     summarize(b_m = mean(rating))  %>%
     ggplot(aes(b_m)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["month"]]) +
     labs(title="Monthly Ratings Histogram",
        x="Mean Rating",
        y="Months")
```



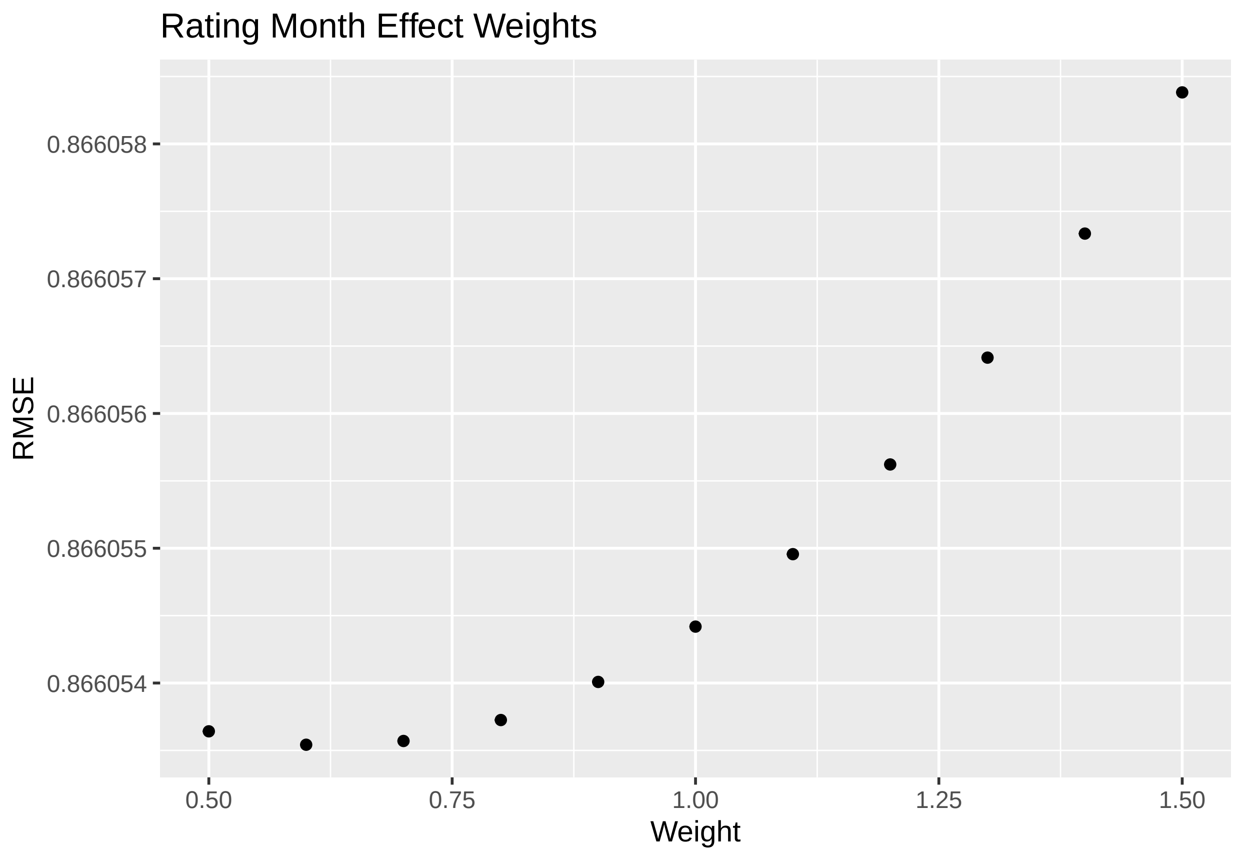
The distribution is fairly even - a similar number of months will see a rating reduction as will see an increase.

```r
#determine the mean movie rating by month adjusted for prior effects
month_avgs <- trainDS %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     left_join(year_avgs, by='movieyear') %>%
     group_by(ratemonth) %>%
     summarize(b_m = mean(rating - mu - b_i - b_y - b_u) * monthweight)

#plot a histogram
month_avgs %>% ggplot(aes(b_m)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["month"]]) +
     labs(title="Month Effect Histogram",
        x="Ratings Change",
        y="Month")
```



After removing the user and movie effects, the ratings will be affected by the month effect (b_m) as shown above. It appears that each month would receive a small adjustment +/- of up to approximately .006.

```
#setup variables
monthweights <- seq(.5,1.5,by=.1)
month_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
month_avgs_best <- c()
last_RMSE <- 1000

for (w in monthweights){

modstart <- Sys.time()

month_avgs <- trainDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     group_by(ratemonth) %>%
     summarize(b_m = mean(rating - mu - b_i - b_u - b_y) * w)
#make predictions on the test set
predicted_ratings <- testDS %>%
     mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     left_join(month_avgs, by='ratemonth') %>%
     mutate(pred = mu + b_i + b_u + b_y + b_m) %>%
     .$pred

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#calculate the RMSE
model_3_rmse <- RMSE(testDS$rating, predicted_ratings)

if(model_3_rmse < last_RMSE){
      month_avgs_best <- c()
      month_avgs_best <- month_avgs
      last_RMSE <- model_3_rmse
   }

modelname <- paste("Rate Month Effect Model")

#show the results
month_results <- bind_rows(month_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_3_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0 ))
}

#plot the RMSE curve
month_results %>%
  ggplot(aes(x=weight,y=RMSE)) +
     geom_point() +
     labs(title="Rating Month Effect Weights",
        x="Weight",
        y="RMSE")
```



```
#keep the best RMSE
month_best <-  month_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, month_best)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0

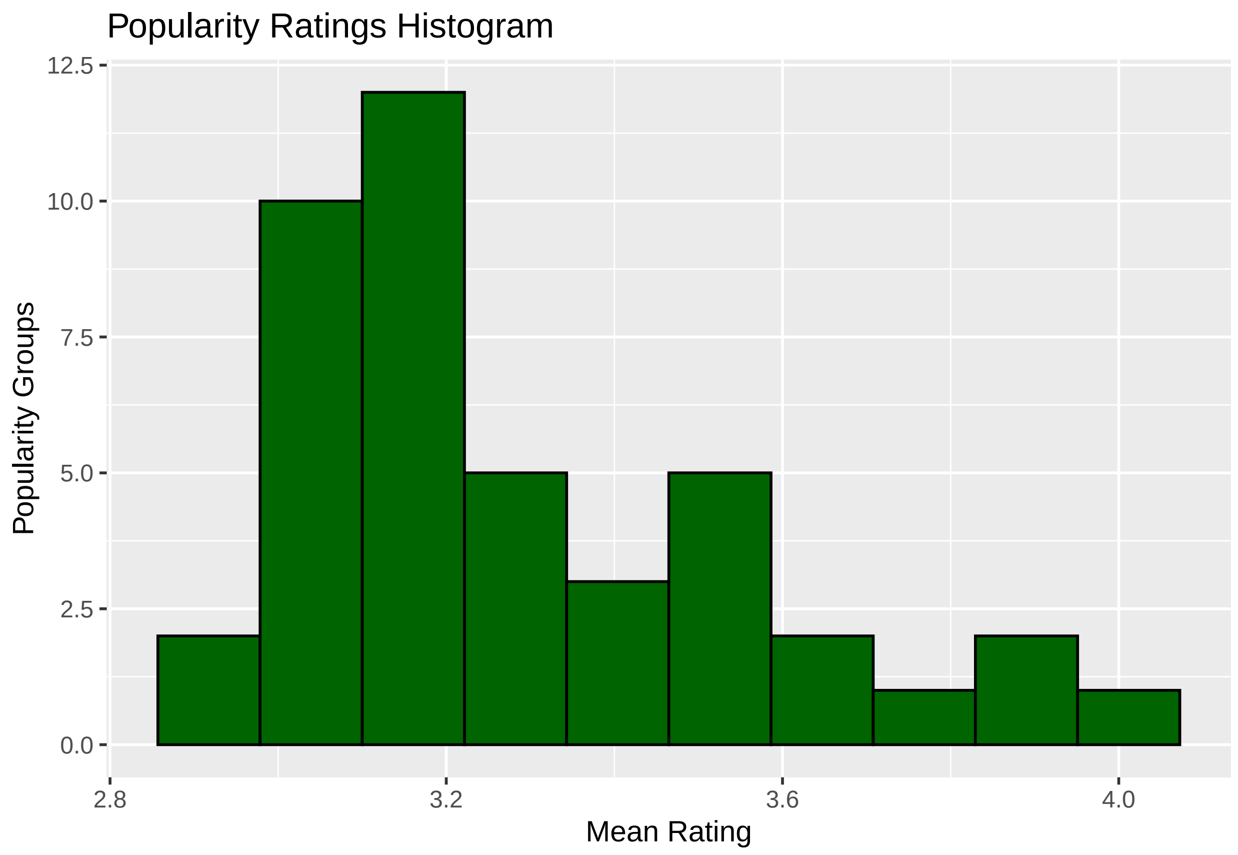
```
month_best_RMSE <- month_best$RMSE
monthweight = month_best$weight
```

The movie month bias provides another minor improvement in RMSE to 0.8660535, when the month bias is assigned a relative weight of 0.6.

Popularity Effect

The popularity rating is designated as a log10 transformed count of the total number of ratings received for that movie, rounded to the nearest tenth. As demonstrated in the previous analysis, the popularity has an impact on ratings, with more popular movies receiving generally better ratings across every era.

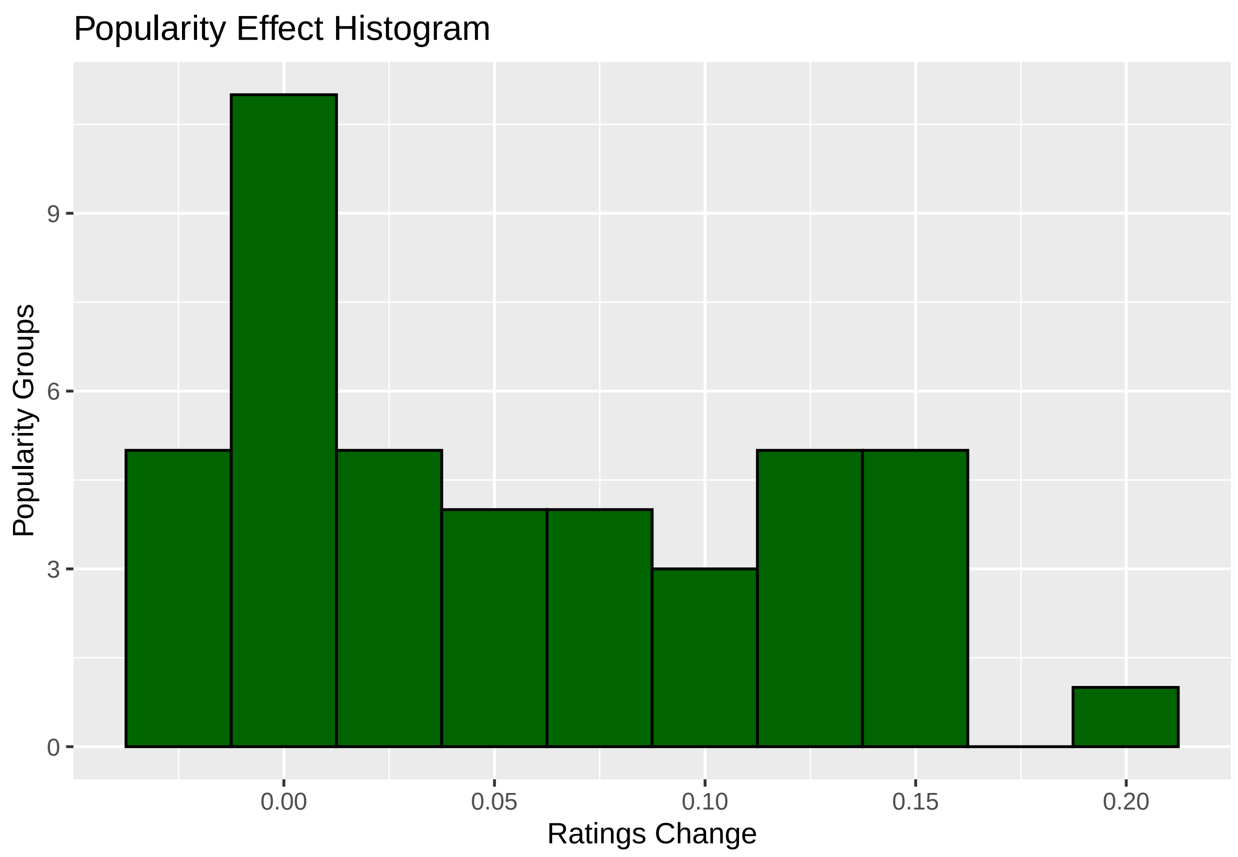
```
#setup variables
popweight <- 1

#determine the mean movie rating by popularity group
trainDS %>%
     left_join(popgroups, by="movieId") %>%
     group_by(popgroup) %>%
     summarize(b_p = mean(rating))  %>%
     ggplot(aes(b_p)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["popular"]]) +
     labs(title="Popularity Ratings Histogram",
        x="Mean Rating",
        y="Popularity Groups")
```



A fair number of popularity groups are rated slightly above or below the mean.

```
#determine the mean movie rating by popularity group adjusted for prior effects
pop_avgs <- trainDS %>%
     left_join(popgroups, by="movieId") %>%
     left_join(user_avgs, by='userId') %>%
```

```r
     left_join(movie_avgs, by='movieId') %>%
     left_join(year_avgs, by='movieyear') %>%
     left_join(month_avgs, by='ratemonth') %>%
     group_by(popgroup) %>%
     summarize(b_p = mean(rating - mu - b_i - b_u - b_y - b_m) * popweight)

#plot a histogram
pop_avgs %>% ggplot(aes(b_p)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["popular"]]) +
     labs(title="Popularity Effect Histogram",
        x="Ratings Change",
        y="Popularity Groups")
```

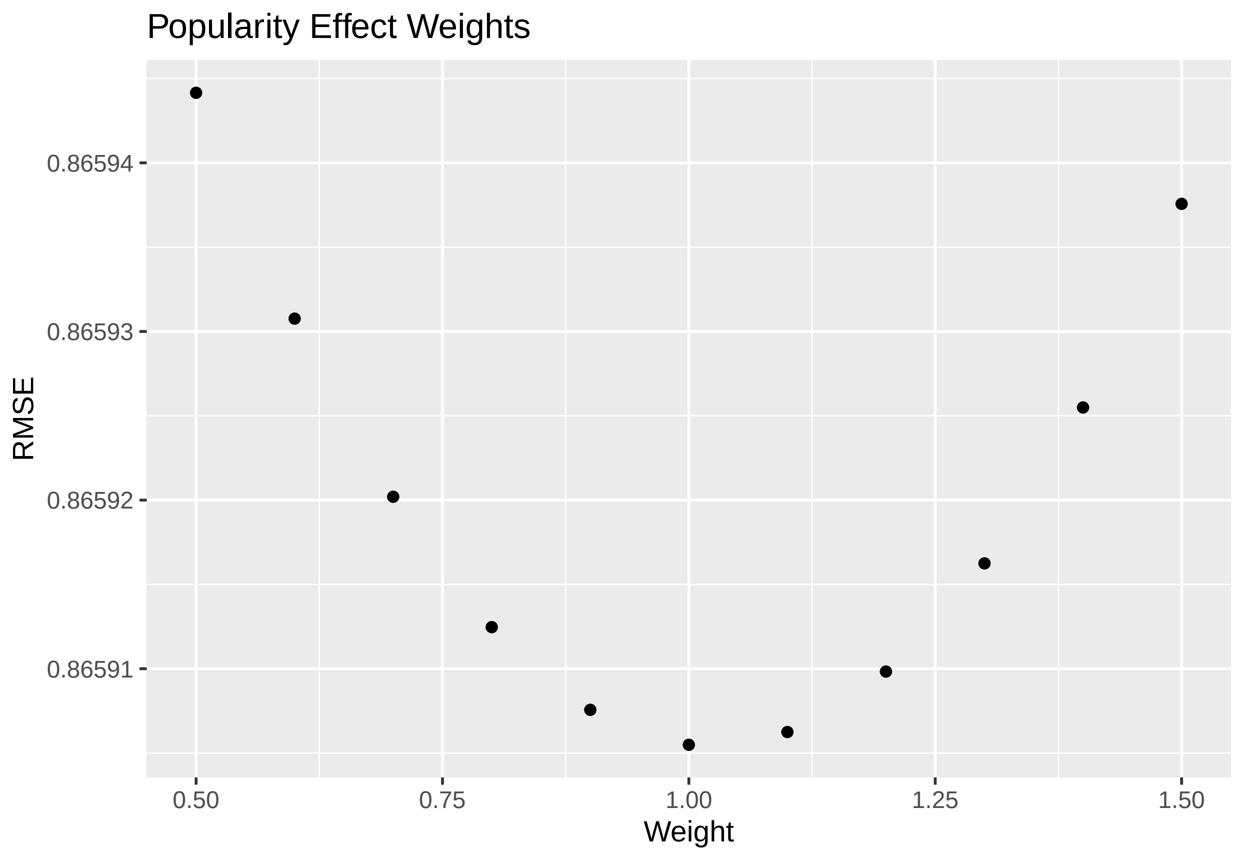


The plot above shows how much popularity effect (b__p), after removing all other bias factors, will influence the average rating up or down. Its clear that this will have a relatively small effect. It appears that our previous factors have accounted for most of the difference we are seeing in popularity.

```r
#setup variables
popweights <- seq(.5,1.5,by=.1)
pop_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
pop_avgs_best <- c()
last_RMSE <- 1000
```

```r
for (w in popweights){
#start benchmark timer
modstart <- Sys.time()

#determine the mean movie rating by popularity group adjusted for prior effects
pop_avgs <- trainDS %>%
     left_join(popgroups, by="movieId") %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     left_join(month_avgs_best, by='ratemonth') %>%
     group_by(popgroup) %>%
     summarize(b_p = mean(rating - mu - b_i - b_u - b_y - b_m) * w)

#make predictions on the test set
predicted_ratings <- testDS %>%
     left_join(popgroups, by="movieId") %>%
     mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     left_join(month_avgs_best, by='ratemonth') %>%
     left_join(pop_avgs, by='popgroup') %>%
     mutate(pred = mu + b_p + b_i + b_u + b_y + b_m) %>%
      .$pred

#end benchmark timer
modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#calculate the RMSE
model_5_rmse <- RMSE(testDS$rating, predicted_ratings)


if(model_5_rmse < last_RMSE){
      pop_avgs_best <- c()
      pop_avgs_best <- pop_avgs
      last_RMSE <- model_5_rmse
   }

modelname <- paste("Popularity Effect Model")

#show the results
pop_results <- bind_rows(pop_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_5_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0 ))


}
```

```r
#plot the RMSE curve
pop_results %>%
  ggplot(aes(x=weight,y=RMSE)) +
     geom_point() +
     labs(title="Popularity Effect Weights",
        x="Weight",
        y="RMSE")
```



```r
#keep the best RMSE
pop_best <-  pop_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, pop_best)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0

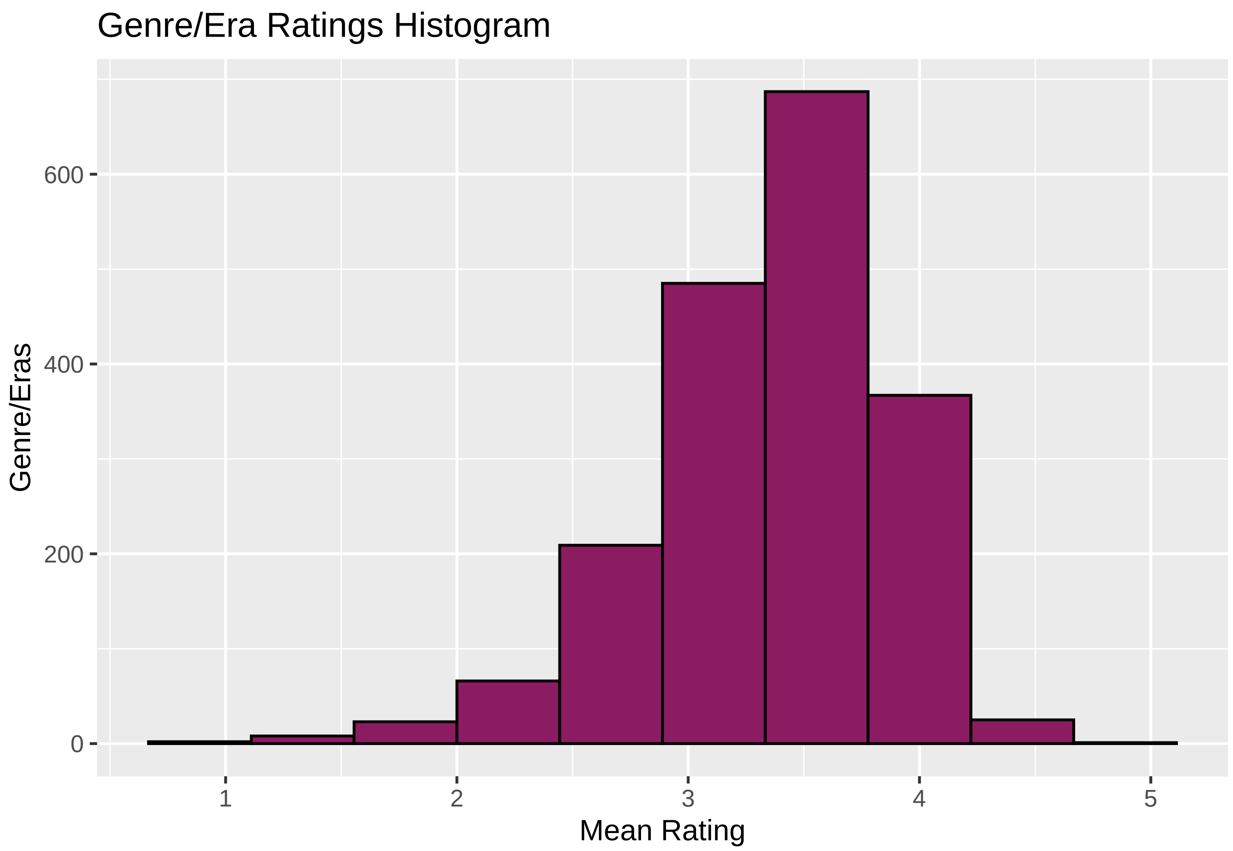
```
pop_best_RMSE <- pop_best$RMSE
popweight = pop_best$weight
```

Again, we can see another slight improvement to 0.8659055 when the popularity factor is weighted at 1.

Genre Effects

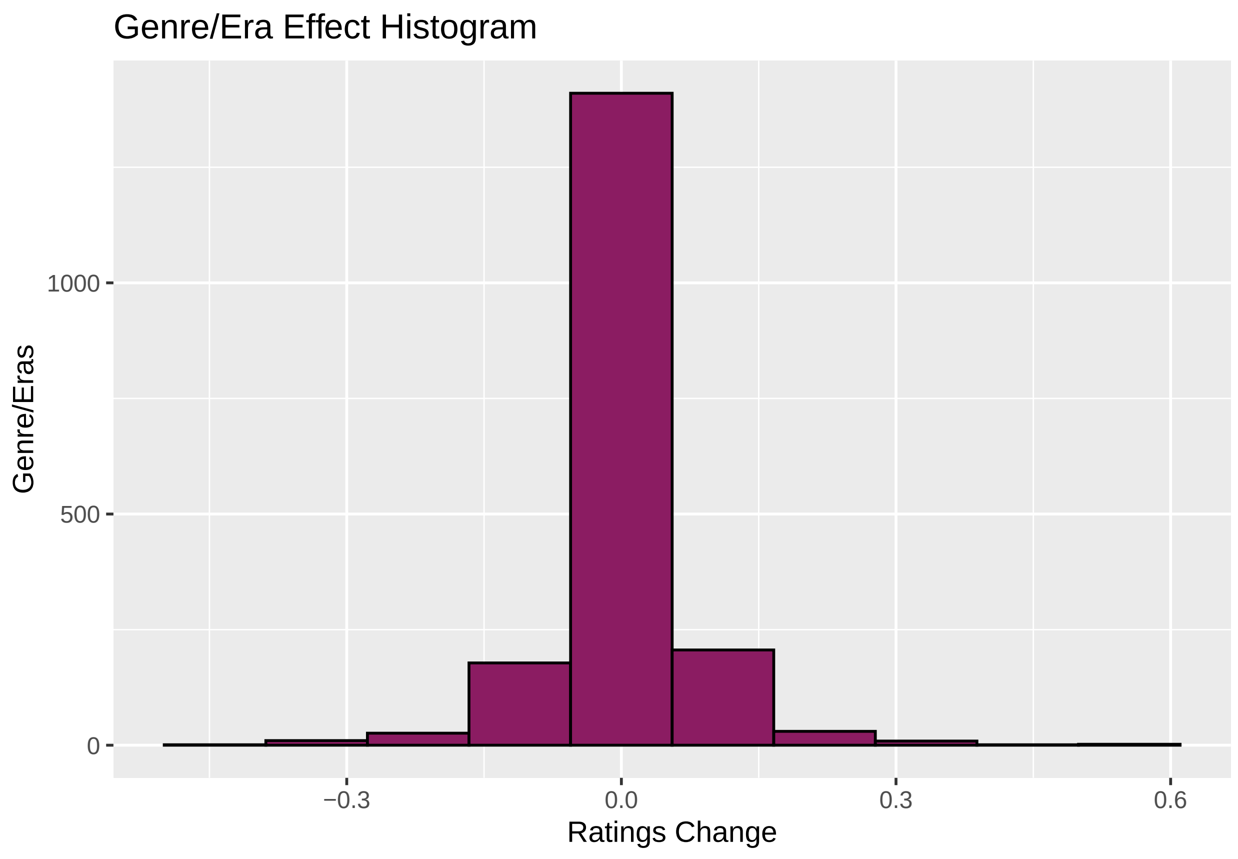
We will now examine how the RMSE is affected by joining the genre AND era/decade together, and using the average of movies of the *same genre, in the same era.*

```
#determine the mean movie rating by genre & era
trainDS %>%
    group_by(era,genres) %>%
     summarize(b_ge = mean(rating))  %>%
     ggplot(aes(b_ge)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["genreera"]]) +
     labs(title="Genre/Era Ratings Histogram",
        x="Mean Rating",
        y="Genre/Eras")
```



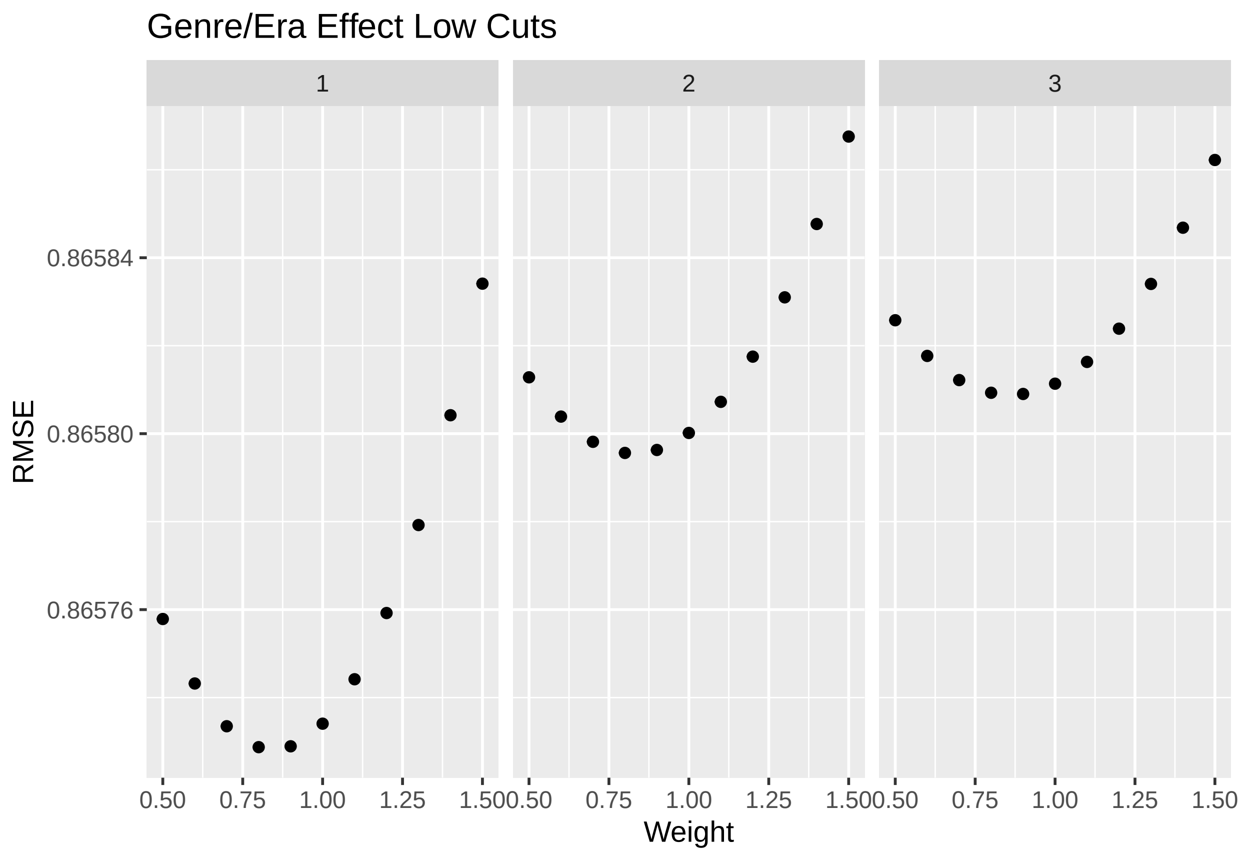
Many genre/era combinations are rated above or below the mean. We will create a dataset with the mean rating for movie in each era/genre combination. The dataset will also contain a count of ratings, count of users, and count of movies within each group. These will be used to filter out entries where there is not enough data to be considered a reliable predictor.

```
#determine the mean movie rating by year adjusted for prior effects
genreera_avgs <- trainDS %>%
     left_join(popgroups, by="movieId") %>%
     left_join(pop_avgs_best, by='popgroup') %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     left_join(year_avgs, by='movieyear') %>%
     left_join(month_avgs, by='ratemonth') %>%
     group_by(era,genres) %>%
     summarize(b_ge = mean(rating - mu - b_i - b_u - b_y - b_m  - b_p),
               rcount=n_distinct(rating),
               mcount=n_distinct(movieId),
               ucount=n_distinct(userId))


#plot a histogram
genreera_avgs %>%  ggplot(aes(b_ge)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["genreera"]]) +
     labs(title="Genre/Era Effect Histogram",
        x="Ratings Change",
        y="Genre/Eras")
```



The plot above shows that the genre/era bias (b_ge), after removing other effects, will have an effect on the model. Some 400 movies would see a slight bump of around .25. The impact of additional predictors is diminishing with each effect added beyond the user and movie effect. For genres, we will apply both a weight adjustment and a low-cut filter to zero out bias effects where we don't have enough data.

```r
#setup variables
genreera_results <- data.frame()
genreeraweights <- seq(.5,1.5,by=.1)
genreeralowcuts <- c(1,2,3)
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
predicted_ratings_genreera_best <- c()
last_RMSE <- 1000
for (w in genreeraweights){
  for (m in genreeralowcuts){

      #start benchmark timer
      modstart <- Sys.time()

      #apply a weight multiplier to the b_ge bias
      genreera_avgs_weighted <- genreera_avgs %>% mutate(b_ge = b_ge * w)

      #Set the bias to NA in cases where we feel there is not enough data in the training set to apply
      genreera_avgs_filtered <- genreera_avgs_weighted %>%
        mutate(b_ge = ifelse(mcount < m,0,b_ge))

      #make predictions on the test set
      predicted_ratings_genreera <- testDS %>%
          mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
          left_join(popgroups, by="movieId") %>%
          left_join(pop_avgs_best, by='popgroup') %>%
          left_join(movie_avgs_best, by='movieId') %>%
          left_join(user_avgs_best, by='userId') %>%
          left_join(year_avgs_best, by='movieyear') %>%
          left_join(month_avgs_best, by='ratemonth') %>%
          left_join(genreera_avgs_filtered, by=c('era','genres')) %>%
          mutate(pred = mu + b_i + b_u + b_y + b_m + b_p + b_ge) %>%
          .$pred

      #end benchmark timer
      modfinish <- Sys.time()
      benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

      modelname <- paste("Genre/Era Effects Model w/ Low Cut")

      #calculate the RMSE
      model_7_rmse <- RMSE(testDS$rating, predicted_ratings_genreera)

      #if this is the best RMSE - keep the table for use in next step
      if(model_7_rmse < last_RMSE){
        predicted_ratings_genreera_best <- c()
        predicted_ratings_genreera_best <- predicted_ratings_genreera
        last_RMSE <- model_7_rmse
      }

      #show the results
      genreera_results <- bind_rows(genreera_results,
                                    data_frame(step = modelstep,
                                               method=modelname,
                                               RMSE = model_7_rmse,
                                               benchmark = benchmark,
                                               weight = w,
                                              lowcut = m))

  }
}

#plot the RMSE curves faceted by lowcuts
genreera_results %>%
  ggplot() +
     geom_point(aes(x=weight,y=RMSE)) +
     facet_wrap(~lowcut) +
      labs(title="Genre/Era Effect Low Cuts",
        x="Weight",
        y="RMSE")
```



```
#keep the best RMSE
genreera_best <-  genreera_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, genreera_best)

rmse_results %>% knitr::kable(label="Model Results")
```

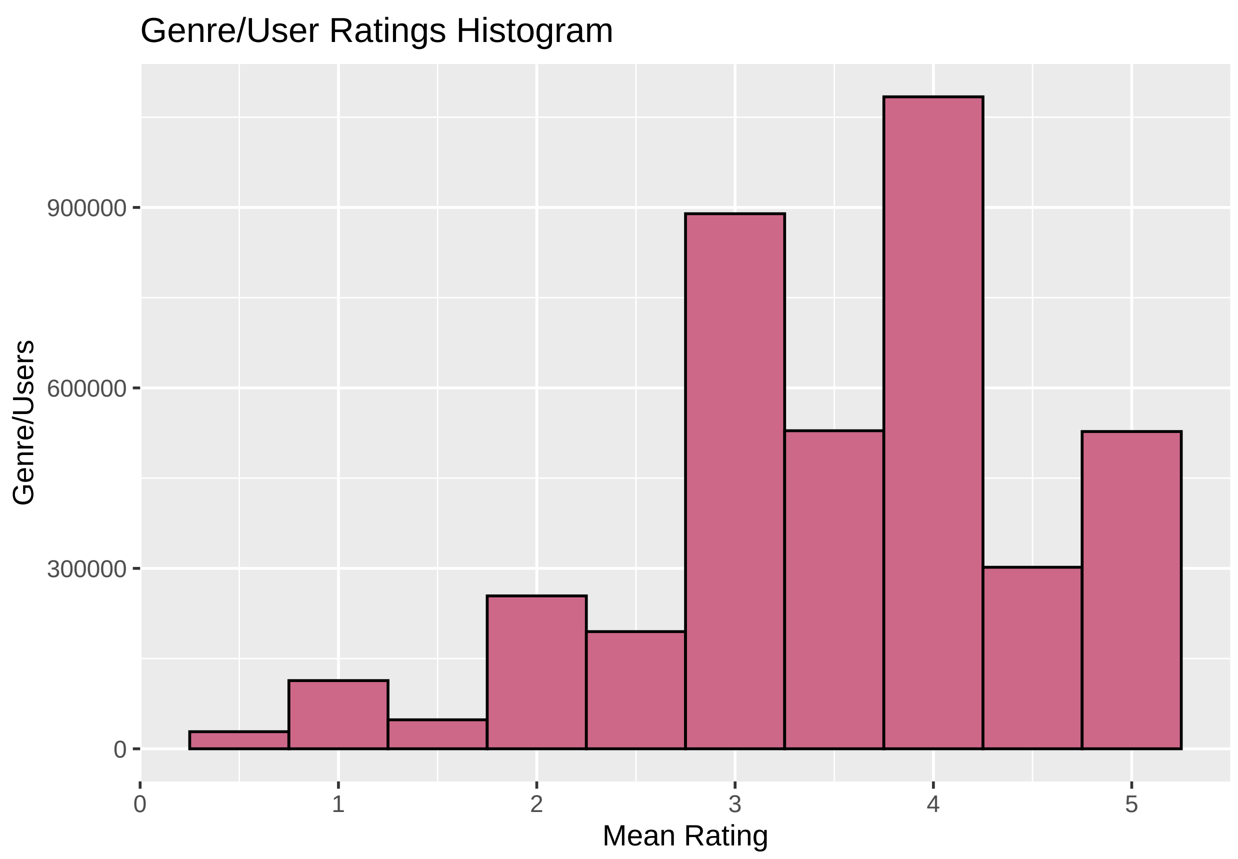
step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1

```
genreeralowcut = genreera_best$lowcut
genreeraweight = genreera_best$weight
```

The genre/era effect effect has reduced the RMSE again. The lowest RMSE is achieved with the lowcut of 1, so all ratings are being included/considered.

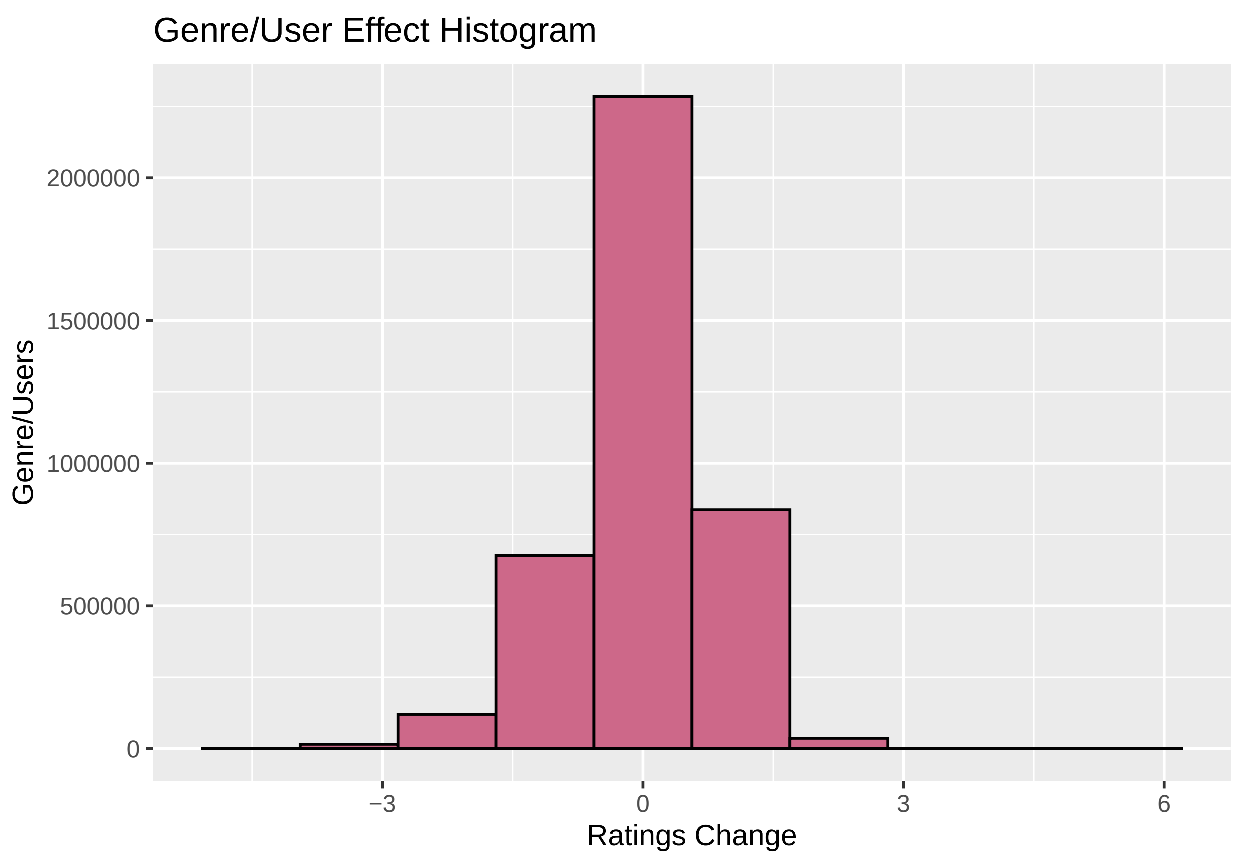
Now we will examine how a users' individual preference for a genre can be incorporated. This would require looking at the genres/userId pairs for any movie genres the user has rated, and applying similar bias if we find that userId and genre combination in the test data. It's logical to assume that not every user would have a previous rating submitted for every movie genre in the test set - so, in these cases we will have the predictions "fall back" to the genre/era bias defined in the previous step.

```
#determine the mean movie rating by genre & era
trainDS %>%
    group_by(userId,genres) %>%
     summarize(b_gu = mean(rating))  %>%
     ggplot(aes(b_gu)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["genreuser"]]) +
     labs(title="Genre/User Ratings Histogram",
        x="Mean Rating",
        y="Genre/Users")
```



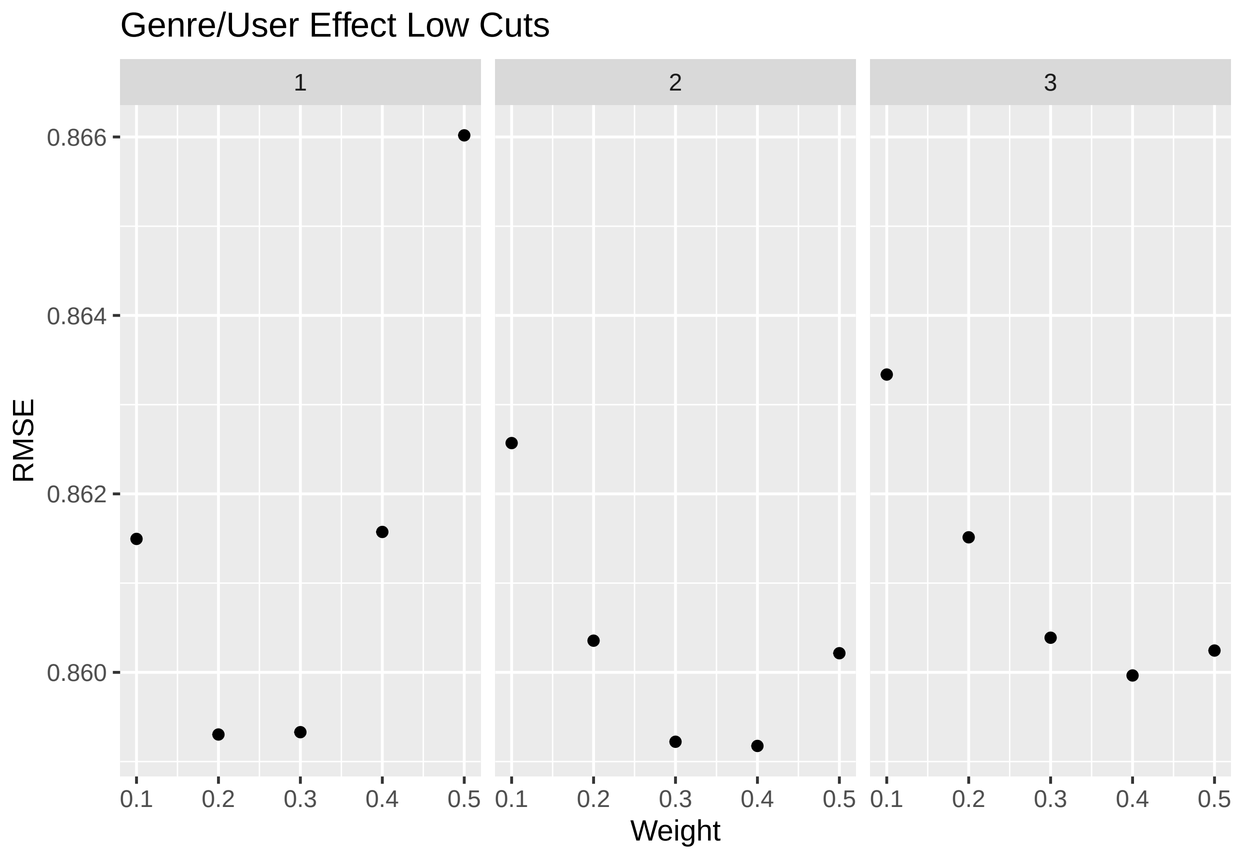
A large number of genre/user combinations have raw ratings below or above the mean.

```r
#determine the mean movie rating by year adjusted for prior effects
genreuser_avgs <- trainDS %>%
        left_join(popgroups, by="movieId") %>%
        left_join(pop_avgs_best, by='popgroup') %>%
        left_join(movie_avgs_best, by='movieId') %>%
        left_join(user_avgs_best, by='userId') %>%
        left_join(year_avgs_best, by='movieyear') %>%
        left_join(month_avgs_best, by='ratemonth') %>%
        group_by(userId,genres) %>%
        summarize(b_gu = mean(rating - mu - b_i - b_u - b_y - b_m - b_p),
               ratecount=n(),
               mcount=n_distinct(movieId))


#plot a histogram
genreuser_avgs %>%  ggplot(aes(b_gu)) +
     geom_histogram(bins = 10, color = "black", fill=palette[["genreuser"]]) +
     labs(title="Genre/User Effect Histogram",
        x="Ratings Change",
        y="Genre/Users")
```



The plot above shows how the genre/user bias (b_gu), after removing other effects, will impact the model. There are nearly 4 million combinations of these two factors. The distribution is roughly even with 800,000 of these pairs receiving a reduction and a similar number seeing an increase.

```r
#setup variables
genreuser_results <- data.frame()
genreuserweights <- seq(.1,.5,by=.1)
modelstep <- modelstep + 1

#set some reasonable values for a low cut on the number of movies that must have been rated to count th
#any genres that don't have this number, will use the previously defined genre/era bias
genreuserlowcuts <- c(1,2,3)

#the best prediction will be assigned to this variable
predicted_ratings_best <- c()
last_RMSE <- 1000
for (w in genreuserweights){
 for (z in genreuserlowcuts){

    #start benchmark timer
     modstart <- Sys.time()

    #apply a weight multiplier to the b_gu bias
    genreuser_avgs_weighted <- genreuser_avgs %>%
      mutate(b_gu = b_gu * w)
    #Set the bias to NA in  cases where we feel there is not enough data in the training set to apply a
    genreuser_avgs_adjusted <- genreuser_avgs_weighted %>%
      mutate(b_gu = ifelse(mcount < z,NA,b_gu))

    #make predictions on the test set
    predicted_ratings_partial <- testDS %>%
         mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
         left_join(popgroups, by="movieId") %>%
         left_join(pop_avgs_best, by='popgroup') %>%
         left_join(movie_avgs_best, by='movieId') %>%
         left_join(user_avgs_best, by='userId') %>%
         left_join(year_avgs_best, by='movieyear') %>%
         left_join(month_avgs_best, by='ratemonth') %>%
         left_join(genreuser_avgs_adjusted, by=c('userId','genres')) %>%
         mutate(pred = mu + b_i + b_u + b_y + b_m + b_p + b_gu) %>%
         .$pred

    #fill in NA values with values from the genre_era predictions
    predicted_ratings <- coalesce(predicted_ratings_partial, predicted_ratings_genreera)

    #end benchmark timer
    modfinish <- Sys.time()
    benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

    #calculate the RMSE
    model_8_rmse <- RMSE(testDS$rating, predicted_ratings)

     if(model_8_rmse < last_RMSE){
      predicted_ratings_best <- c()
      predicted_ratings_best <- predicted_ratings
      last_RMSE <- model_8_rmse
    }

    modelname <- paste("Genre/User Effects Model w/ Low Cut")

    #show the results
    genreuser_results <- bind_rows(genreuser_results,
                                   data_frame(step = modelstep,
                                              method=modelname,
                                              RMSE = model_8_rmse,
                                              benchmark = benchmark,
                                              weight = w,
                                              lowcut = z))

 }
}

#plot the RMSE curves faceted by lowcuts
genreuser_results %>%
  ggplot(aes(x=weight,y=RMSE)) +
     geom_point() +
    facet_wrap(~lowcut) +
     labs(title="Genre/User Effect Low Cuts",
        x="Weight",
        y="RMSE")
```



```
#keep the best RMSE
genreuser_best <-  genreuser_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, genreuser_best)

rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1
8	Genre/User Effects Model w/ Low Cut	0.8591753	6.22 secs	0.400	2

```
genreuserlowcut = genreuser_best$lowcut
genreuserweight = genreuser_best$weight
```

A low-cut setting of 2 on the genre-user bias produces the lowest RMSE. This means that a user with fewer than 2 ratings in a genre, will not have the effect applied, and instead the genre-era bias will be used. This method will produce the lowest RMSE when weighted at 0.4.

One last relatively easy change to increase precision, would be to round off predictions that are outside the range of possible values. Since the highest rating is 5 and lowest is .5, we will add a "bracket" function to round off ratings that fall out of that range.

```
#this helper function will round any values above 5 or below .5 since these values can only increase th
bracket <- function(v){
  v <- max(min(v,5),.5)
  v
}

#alter predictions with the bracket function
predicted_ratings <- sapply(predicted_ratings_best,bracket)

#calculate the RMSE
model_5R_rmse <- RMSE(testDS$rating,predicted_ratings)
modelstep <- modelstep + 1

#show the results
rmse_results <- bind_rows(rmse_results,
                          data_frame(step = modelstep,
                                     method="Multi-Effect Model w/ Bracket",
                                     RMSE = model_5R_rmse,
                                     benchmark = benchmark ))
rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1
8	Genre/User Effects Model w/ Low Cut	0.8591753	6.22 secs	0.400	2
9	Multi-Effect Model w/ Bracket	0.8590279	6.21 secs	NA	NA

Rounding off guesses that are known to be too high or low, produces a small improvement in overall accuracy. It will be important that we only apply the bracket function after all other effects are applied.

Multi-Effect Model

For the remainder of this report the effects described above will be referred to as the Multi-Effect Model.

Based on the data exploration, we will finally construct a function to cross-validate our model against our various partitions, incorporating all of the following effects:

- Movie Rating Average
- User Rating Average

- Rating Average of Movies made in the same Year
- Rating Average of Movies rated in the same Month of the Year
- Rating Average of Movies that are similar in Popularity
- Rating Average of Movies in the same Genre, per Era
- Rating Average of Movies in the same Genre, per User

A function called *trainingmodel* will incorporate the same processes we explored in the model construction above. This training model will be used to try out weights and filter settings, and a final, parameterized model called multieffectmodel will facilitate testing on the final hold out data.

The final *multieffectmodel* function will accept as parameters: a reference to the training and test sets, a set of weight parameters for each effect, and low-cut/high-pass filters for the genre effects. Finally, the model incorporates the penalty factor which we will optimize in the following step to further reduce over-fitting. This model will also apply our rounding bracket function to the final predictions. The function will then return a "modeltest" object containing the RMSE and a vector of the final predictions.

```
#this model will take a given training and test set and return parmeters for optimization.
#the average parameters will be plugged in to the final multieffect model

trainingmodel <- function(trainDS,
                          testDS,
                          movieweights,
                          userweights,
                          yearweights,
                          monthweights,
                          popweights,
                          genreeraweights,
                          genreuserweights,
                          genreeralowcuts,
                          genreuserlowcuts){

#IMPORTANT - Resetting mu here to the training set
#this is the baseline to which all effects will be added/subtracted
mu <- mean(trainDS$rating)
predictions <- rep(mu, nrow(trainDS))

#create a vector of predictions that are all the same (mu) value
#modstart and modfinish will time the routine for benchmarking
modstart <- Sys.time()
predictions <- rep(mu, nrow(trainDS))
modfinish <- Sys.time()

#get the RMSE of the naive model
naive_rmse <- RMSE(testDS$rating,predictions)

#determine the operation speed. This operation will be done for each model
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#modelstep will increment for each new test
modelstep <- 0
modelstep <- modelstep + 1

#show the RMSE results in a kable
rmse_results <- data_frame(step=modelstep,
                            method = "Baseline",
                            RMSE = naive_rmse,
                            benchmark = benchmark,
                                      weight = 1,
                                      lowcut = 0)


#-----MOVIE EFFECTS ------


#setup variables
movie_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
movie_avgs_best <- c()
last_RMSE <- 1000

for (w in movieweights){

modstart <- Sys.time()

movie_avgs <- trainDS %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu) * w)

#using the testDS set only to make a prediction.
predicted_ratings <- mu + testDS %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#check prediction vs the results in the testDS set
model_1_rmse <- RMSE(testDS$rating, predicted_ratings)

if(model_1_rmse < last_RMSE){
      movie_avgs_best <- c()
      movie_avgs_best <- movie_avgs
      last_RMSE <- model_1_rmse
    }


modelname <- paste("Movie Effect Model")

#checking my prediction vs the results in the testDS set
movie_results <- bind_rows(movie_results,
                           data_frame(step = modelstep,
                                      method=modelname,
                                      RMSE = model_1_rmse,
                                      benchmark = benchmark,
                                      weight = w,
                                      lowcut = 0
                                      ))
}

#keep the best RMSE
movie_best <-  movie_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, movie_best)

movieweight = movie_best$weight


#-----USER EFFECTS ------


#setup variables
user_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
user_avgs_best <- c()
last_RMSE <- 1000

for (w in userweights){
modstart <- Sys.time()

#determine the mean user rating after deducting prior effects
user_avgs <- trainDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_i) * w)

#predict ratings on the test set
predicted_ratings <- testDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     mutate(pred = mu + b_i + b_u) %>%
     .$pred

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

modelname <- paste("User Effect Model")

#calculate the RMSE
model_2_rmse <- RMSE(testDS$rating, predicted_ratings)


if(model_2_rmse < last_RMSE){
      user_avgs_best <- c()
      user_avgs_best <- user_avgs
      last_RMSE <- model_2_rmse
   }
```

```
#show the results
user_results <- bind_rows(user_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_2_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0
                                     ))
}

#keep the best RMSE
user_best <-  user_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, user_best)

userweight = user_best$weight


#-----YEAR EFFECTS ------

#setup variables
year_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
year_avgs_best <- c()
last_RMSE <- 1000

for (w in yearweights){

modstart <- Sys.time()

year_avgs <- trainDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     group_by(movieyear) %>%
     summarize(b_y = mean(rating - mu - b_i - b_u) * w)

#make predictions on the test set
predicted_ratings <- testDS %>%
     mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs, by='movieyear') %>%
     mutate(pred = mu + b_i + b_u + b_y) %>%
     .$pred

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#calculate the RMSE
```

```r
model_3_rmse <- RMSE(testDS$rating, predicted_ratings)

if(model_3_rmse < last_RMSE){
      year_avgs_best <- c()
      year_avgs_best <- year_avgs
      last_RMSE <- model_3_rmse
   }

modelname <- paste("Year Effect Model")

#show the results
year_results <- bind_rows(year_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_3_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0))

}


#keep the best RMSE
year_best <-  year_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, year_best)

yearweight = year_best$weight


#-----RATING MONTH EFFECTS ------


#setup variables
month_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
month_avgs_best <- c()
last_RMSE <- 1000

for (w in monthweights){

modstart <- Sys.time()

month_avgs <- trainDS %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     group_by(ratemonth) %>%
     summarize(b_m = mean(rating - mu - b_i - b_u - b_y) * w)

#make predictions on the test set
```

```
predicted_ratings <- testDS %>%
     mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     left_join(month_avgs, by='ratemonth') %>%
     mutate(pred = mu + b_i + b_u + b_y + b_m) %>%
     .$pred

modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#calculate the RMSE
model_4_rmse <- RMSE(testDS$rating, predicted_ratings)

if(model_4_rmse < last_RMSE){
      month_avgs_best <- c()
      month_avgs_best <- month_avgs
      last_RMSE <- model_4_rmse
   }

modelname <- paste("Rate Month Effect Model")

#show the results
month_results <- bind_rows(month_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_4_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0 ))
}

#keep the best RMSE
month_best <-  month_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, month_best)

monthweight = month_best$weight


#-----POPULARITY EFFECTS ------


#setup variables
pop_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
pop_avgs_best <- c()
last_RMSE <- 1000

for (w in popweights){
```

```
#start benchmark timer
modstart <- Sys.time()

#determine the mean movie rating by popularity group adjusted for prior effects
pop_avgs <- trainDS %>%
     left_join(popgroups, by="movieId") %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     left_join(month_avgs_best, by='ratemonth') %>%
     group_by(popgroup) %>%
     summarize(b_p = mean(rating - mu - b_i - b_u - b_y - b_m) * w)

#make predictions on the test set
predicted_ratings <- testDS %>%
     left_join(popgroups, by="movieId") %>%
     mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
     left_join(movie_avgs_best, by='movieId') %>%
     left_join(user_avgs_best, by='userId') %>%
     left_join(year_avgs_best, by='movieyear') %>%
     left_join(month_avgs_best, by='ratemonth') %>%
     left_join(pop_avgs, by='popgroup') %>%
     mutate(pred = mu + b_p + b_i + b_u + b_y + b_m) %>%
      .$pred

#end benchmark timer
modfinish <- Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

#calculate the RMSE
model_5_rmse <- RMSE(testDS$rating, predicted_ratings)


if(model_5_rmse < last_RMSE){
      pop_avgs_best <- c()
      pop_avgs_best <- pop_avgs
      last_RMSE <- model_5_rmse
   }

modelname <- paste("Popularity Effect Model")

#show the results
pop_results <- bind_rows(pop_results,
                          data_frame(step = modelstep,
                                     method=modelname,
                                     RMSE = model_5_rmse,
                                     benchmark = benchmark,
                                     weight = w,
                                     lowcut = 0 ))



}
```

```r
#keep the best RMSE
pop_best <-  pop_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, pop_best)

rmse_results %>% knitr::kable(label="Model Results")

popweight = pop_best$weight

#-----GENRE/ERA EFFECTS ------

#determine the mean movie rating by year adjusted for prior effects
genreera_avgs <- trainDS %>%
     left_join(popgroups, by="movieId") %>%
     left_join(pop_avgs_best, by='popgroup') %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     left_join(year_avgs, by='movieyear') %>%
     left_join(month_avgs, by='ratemonth') %>%
     group_by(era,genres) %>%
     summarize(b_ge = mean(rating - mu - b_i - b_u - b_y - b_m  - b_p),
               rcount=n_distinct(rating),
               mcount=n_distinct(movieId),
               ucount=n_distinct(userId))



#setup variables
genreera_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
predicted_ratings_genreera_best <- c()
last_RMSE <- 1000
for (w in genreeraweights){
  for (m in genreeralowcuts){

      #start benchmark timer
      modstart <- Sys.time()

      #apply a weight multiplier to the b_ge bias
      genreera_avgs_weighted <- genreera_avgs %>% mutate(b_ge = b_ge * w)

      #Set the bias to NA in cases where we feel there is not enough data in the training set to apply
      genreera_avgs_filtered <- genreera_avgs_weighted %>%
        mutate(b_ge = ifelse(mcount < m,0,b_ge))

      #make predictions on the test set
      predicted_ratings_genreera <- testDS %>%
           mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
           left_join(popgroups, by="movieId") %>%
           left_join(pop_avgs_best, by='popgroup') %>%
           left_join(movie_avgs_best, by='movieId') %>%
           left_join(user_avgs_best, by='userId') %>%
           left_join(year_avgs_best, by='movieyear') %>%
           left_join(month_avgs_best, by='ratemonth') %>%
           left_join(genreera_avgs_filtered, by=c('era','genres')) %>%
           mutate(pred = mu + b_i + b_u + b_y + b_m + b_p + b_ge) %>%
           .$pred

      #end benchmark timer
      modfinish <- Sys.time()
      benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)

      modelname <- paste("Genre/Era Effects Model w/ Low Cut")

      #calculate the RMSE
      model_6_rmse <- RMSE(testDS$rating, predicted_ratings_genreera)

      #if this is the best RMSE - keep the table for use in next step
      if(model_6_rmse < last_RMSE){
        predicted_ratings_genreera_best <- c()
        predicted_ratings_genreera_best <- predicted_ratings_genreera
        last_RMSE <- model_6_rmse
      }

      #show the results
      genreera_results <- bind_rows(genreera_results,
                                data_frame(step = modelstep,
                                           method=modelname,
                                           RMSE = model_6_rmse,
                                           benchmark = benchmark,
                                           weight = w,
                                          lowcut = m))

  }
}


#keep the best RMSE
genreera_best <-  genreera_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, genreera_best)

genreeralowcut = genreera_best$lowcut
genreeraweight = genreera_best$weight



#-----GENRE/USER EFFECTS ------



#determine the mean movie rating by year adjusted for prior effects
genreuser_avgs <- trainDS %>%
        left_join(popgroups, by="movieId") %>%
        left_join(pop_avgs_best, by='popgroup') %>%
        left_join(movie_avgs_best, by='movieId') %>%
        left_join(user_avgs_best, by='userId') %>%
        left_join(year_avgs_best, by='movieyear') %>%
        left_join(month_avgs_best, by='ratemonth') %>%
        group_by(userId,genres) %>%
        summarize(b_gu = mean(rating - mu - b_i - b_u - b_y - b_m - b_p),
               ratecount=n(),
               mcount=n_distinct(movieId))



#setup variables
genreuser_results <- data.frame()
modelstep <- modelstep + 1

#the best prediction will be assigned to this variable
predicted_ratings_best <- c()
last_RMSE <- 1000
for (w in genreuserweights){
 for (z in genreuserlowcuts){

    #start benchmark timer
     modstart <- Sys.time()

    #apply a weight multiplier to the b_gu bias
    genreuser_avgs_weighted <- genreuser_avgs %>%
      mutate(b_gu = b_gu * w)

    #Set the bias to NA in  cases where we feel there is not enough data in the training set to apply a
    genreuser_avgs_adjusted <- genreuser_avgs_weighted %>%
      mutate(b_gu = ifelse(mcount < z,NA,b_gu))

    #make predictions on the test set
    predicted_ratings_partial <- testDS %>%
         mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
         left_join(popgroups, by="movieId") %>%
         left_join(pop_avgs_best, by='popgroup') %>%
         left_join(movie_avgs_best, by='movieId') %>%
         left_join(user_avgs_best, by='userId') %>%
         left_join(year_avgs_best, by='movieyear') %>%
         left_join(month_avgs_best, by='ratemonth') %>%
         left_join(genreuser_avgs_adjusted, by=c('userId','genres')) %>%
         mutate(pred = mu + b_i + b_u + b_y + b_m + b_p + b_gu) %>%
         .$pred

    #fill in NA values with values from the genre_era predictions
    predicted_ratings <- coalesce(predicted_ratings_partial, predicted_ratings_genreera)

    #end benchmark timer
    modfinish <- Sys.time()
    benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)
    #calculate the RMSE
    model_7_rmse <- RMSE(testDS$rating, predicted_ratings)

     if(model_7_rmse < last_RMSE){
      predicted_ratings_best <- c()
      predicted_ratings_best <- predicted_ratings
      last_RMSE <- model_7_rmse
    }

    modelname <- paste("Genre/User Effects Model w/ Low Cut")

    #show the results
    genreuser_results <- bind_rows(genreuser_results,
                                   data_frame(step = modelstep,
                                              method=modelname,
                                              RMSE = model_7_rmse,
                                              benchmark = benchmark,
                                              weight = w,
                                              lowcut = z))

 }
}


#keep the best RMSE
genreuser_best <-  genreuser_results %>% slice(which.min(RMSE))

rmse_results <- bind_rows(rmse_results, genreuser_best)

genreuserlowcut = genreuser_best$lowcut
genreuserweight = genreuser_best$weight


#alter predictions with the bracket function
predicted_ratings <- sapply(predicted_ratings_best,bracket)

#calculate the RMSE
model_8_rmse <- RMSE(testDS$rating,predicted_ratings)
modelstep <- modelstep + 1


#show the results
rmse_results <- bind_rows(rmse_results,
                          data_frame(step = modelstep,
                                     method="Multi-Effect Model w/ Bracket",
                                     RMSE = model_8_rmse,
                                     benchmark = benchmark ))


#return the RMSE, prediction, penalty factor, & benchmark time
      returnobj <-list(table=rmse_results)

      class(returnobj) <- "resulttable"
      returnobj


}
```

The finalized training model is now ready for optimization of the weights and filter parameters.

```
#now we will use the three partitions already created and try to get the best
#weight/lowcut settings by averaging the results of three different splits of the data.
#this code takes a while to run.

train_A <- trainingmodel(part9010_A$trainDS,
                         part9010_A$testDS,
                          movieweights,
                          userweights,
                          yearweights,
                          monthweights,
                          popweights,
                          genreeraweights,
                          genreuserweights,
                          genreeralowcuts,
                          genreuserlowcuts)

train_B <- trainingmodel(part9010_B$trainDS,
                         part9010_B$testDS,
                          movieweights,
                          userweights,
                          yearweights,
                          monthweights,
                          popweights,
                          genreeraweights,
                          genreuserweights,
                          genreeralowcuts,
                          genreuserlowcuts)

train_C <- trainingmodel(part9010_C$trainDS,
                         part9010_C$testDS,
                          movieweights,
                          userweights,
                          yearweights,
                          monthweights,
                          popweights,
                          genreeraweights,
                          genreuserweights,
                          genreeralowcuts,
                          genreuserlowcuts)

#this function will return an average of the 3 parameters
getmean <- function(tbl1,tbl2,tbl3,row,col){
  v1 <- tbl1$table[row,col]
  v2 <- tbl2$table[row,col]
  v3 <- tbl3$table[row,col]
  mean <- (v1+v2+v3) / 3
}
```

```
#this block sets the final single parameter values to be used in the final model
avgmovieweight <- getmean(train_A,train_B,train_C,2,5)$weight
avguserweight <- getmean(train_A,train_B,train_C,3,5)$weight
avgyearweight <- getmean(train_A,train_B,train_C,4,5)$weight
avgmonthweight <- getmean(train_A,train_B,train_C,5,5)$weight
avgpopweight <- getmean(train_A,train_B,train_C,6,5)$weight
avggenreeraweight <- getmean(train_A,train_B,train_C,7,5)$weight
avggenreuserweight <- getmean(train_A,train_B,train_C,8,5)$weight
avggenreeralowcut <- round(getmean(train_A,train_B,train_C,7,6))$lowcut
avggenreuserlowcut <- round(getmean(train_A,train_B,train_C,8,6))$lowcut
```

We now have all the parameters (other than the regularization penalty) defined. These are all defined as the average of the three training models “best parameters”. We will now encapsulate the above model into our multieffectmodel function without all the optimization loops. This is the final model we will use for any additional work after this point.

```
#build the multi-effect model model function
#we have changed the code to use sum/n, as we will later attempt regularization
#the model will round off predictions over 5 and under .5 using the bracket function

multieffectmodel <- function(trainDS,
                             testDS,
                             movieweight,
                             userweight,
                             yearweight,
                             monthweight,
                             popweight,
                             genreeraweight,
                             genreuserweight,
                             genreeralowcut,
                             genreuserlowcut,
                             lambda){
    returnobj <- NULL
    returnobj <- data.frame()

    #start benchmark timer
    modstart <- Sys.time()

    #set mu baseline rating
    mu <- mean(trainDS$rating)

    #add the movie effect
    b_i <- trainDS %>%
              group_by(movieId) %>%
              summarize(b_i = sum(rating - mu) *
                          movieweight /(n()+lambda))

    #add the user effect
    b_u <- trainDS %>%
              left_join(b_i, by="movieId") %>%
              group_by(userId) %>%
              summarize(b_u = sum(rating - b_i - mu) *
                          userweight / (n()+lambda))
```

```r
#add the year effect
b_y <- trainDS %>%
          left_join(b_i, by="movieId") %>%
          left_join(b_u, by="userId") %>%
          group_by(movieyear) %>%
          summarize(b_y = sum(rating - mu - b_i - b_u) *
                      yearweight / (n()+lambda))

#add the month effect
b_m <- trainDS %>%
          left_join(b_i, by="movieId") %>%
          left_join(b_u, by="userId") %>%
          left_join(b_y, by="movieyear") %>%
          group_by(ratemonth) %>%
          summarize(b_m = sum(rating - mu - b_i - b_u - b_y) *
                      monthweight / (n()+lambda))

#add the popularity effect
b_p <- trainDS %>%
          left_join(popgroups, by="movieId") %>%
          left_join(b_i, by="movieId") %>%
          left_join(b_u, by="userId") %>%
          left_join(b_y, by="movieyear") %>%
          left_join(b_m, by="ratemonth") %>%
          group_by(popgroup) %>%
          summarize(b_p = sum(rating - mu - b_i - b_u - b_y - b_m) *
                      popweight / (n()+lambda))


# generate predictions for first set
b_ge <- trainDS %>%
          left_join(popgroups, by="movieId") %>%
          left_join(b_i, by="movieId") %>%
          left_join(b_u, by="userId") %>%
          left_join(b_y, by="movieyear") %>%
          left_join(b_m, by="ratemonth") %>%
          left_join(b_p, by = "popgroup") %>%
          group_by(era,genres) %>%
          summarize(b_ge = sum(rating - mu - b_i - b_u - b_y - b_m - b_p) *
                      genreeraweight / (n()+lambda),
           rcount=n_distinct(rating),
           mcount=n_distinct(movieId),
           ucount=n_distinct(userId))

  #Set the bias to 0 in  cases where there is not enough data
  b_ge <- b_ge %>% mutate(b_ge = ifelse(mcount < genreeralowcut,0,b_ge))

  #test using part of the data to predict the rest
  predicted_ratings_genreera <- testDS %>%
       left_join(popgroups, by="movieId") %>%
            left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>%
            left_join(b_y, by = "movieyear") %>%
            left_join(b_m, by = "ratemonth") %>%
            left_join(b_p, by = "popgroup") %>%
            left_join(b_ge, by=c('era','genres')) %>%
            mutate(pred = mu + b_i + b_u + b_y + b_m + b_p + b_ge) %>%
   .$pred

  # generate predictions for second set
  b_gu <- trainDS %>%
          left_join(popgroups, by="movieId") %>%
          left_join(b_i, by="movieId") %>%
          left_join(b_u, by="userId") %>%
          left_join(b_y, by="movieyear") %>%
          left_join(b_m, by="ratemonth") %>%
          left_join(b_p, by = "popgroup") %>%
          group_by(userId,genres) %>%
          summarize(b_gu = sum(rating - mu - b_i - b_u - b_y - b_m - b_p) *
                      genreuserweight / (n()+lambda),
          rcount=n(),
          mcount=n_distinct(movieId))

  #Set the bias to NA in  cases where there is not enough data
  b_gu <- b_gu %>% mutate(b_gu = ifelse(mcount < genreuserlowcut,NA,b_gu))


  #test using part of the data to predict the rest
  predicted_ratings_genreuser <- testDS %>%
       left_join(popgroups, by="movieId") %>%
            left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>%
            left_join(b_y, by = "movieyear") %>%
            left_join(b_m, by = "ratemonth") %>%
            left_join(b_p, by = "popgroup") %>%
            left_join(b_gu, by=c('userId','genres')) %>%
            mutate(pred = mu + b_i + b_u + b_y + b_m + b_p + b_gu) %>%
   .$pred

  #combine the genre/era predictions with the genre/user predictions where genre/user combos are NA
  predicted_ratings <- coalesce(predicted_ratings_genreuser, predicted_ratings_genreera)

  #run RMSE and return RMSE as numeric
  pred_rate <- as.numeric(predicted_ratings)
  pred_comp <- as.numeric(testDS$rating)

  #sapply the bracket function
  pred_raterounded <- sapply(pred_rate,bracket)

  #Model is finished/predictions are produced, compute benchmark
  modfinish <- Sys.time()
  bm <- round(difftime(modfinish, modstart, units = "secs"),2)

  #calculate the RMSE
  model_OA_rmse <- RMSE(pred_comp,pred_raterounded)
```

```
      #return the RMSE, prediction, penalty factor, & benchmark time
      returnobj <-list(RMSE=model_0A_rmse,
                  pred=pred_raterounded,
                  penalty = lambda,
                  benchmark = bm
                  )

      class(returnobj) <- "modeltest"
      returnobj

}



#cross validate against 3 training partitions

#10% training 90% test
trainDS <-part9010_A$trainDS
testDS <- part9010_A$testDS

#check the RMSE on this partition
multimodel_result <- multieffectmodel(trainDS,
                                  testDS,
                                  avgmovieweight,
                                  avguserweight,
                                  avgyearweight,
                                  avgmonthweight,
                                  avgpopweight,
                                  avggenreeraweight,
                                  avggenreuserweight,
                                  avggenreeralowcut,
                                  avggenreuserlowcut,
                                  0)

edx_9010_rmse <- multimodel_result$RMSE
```

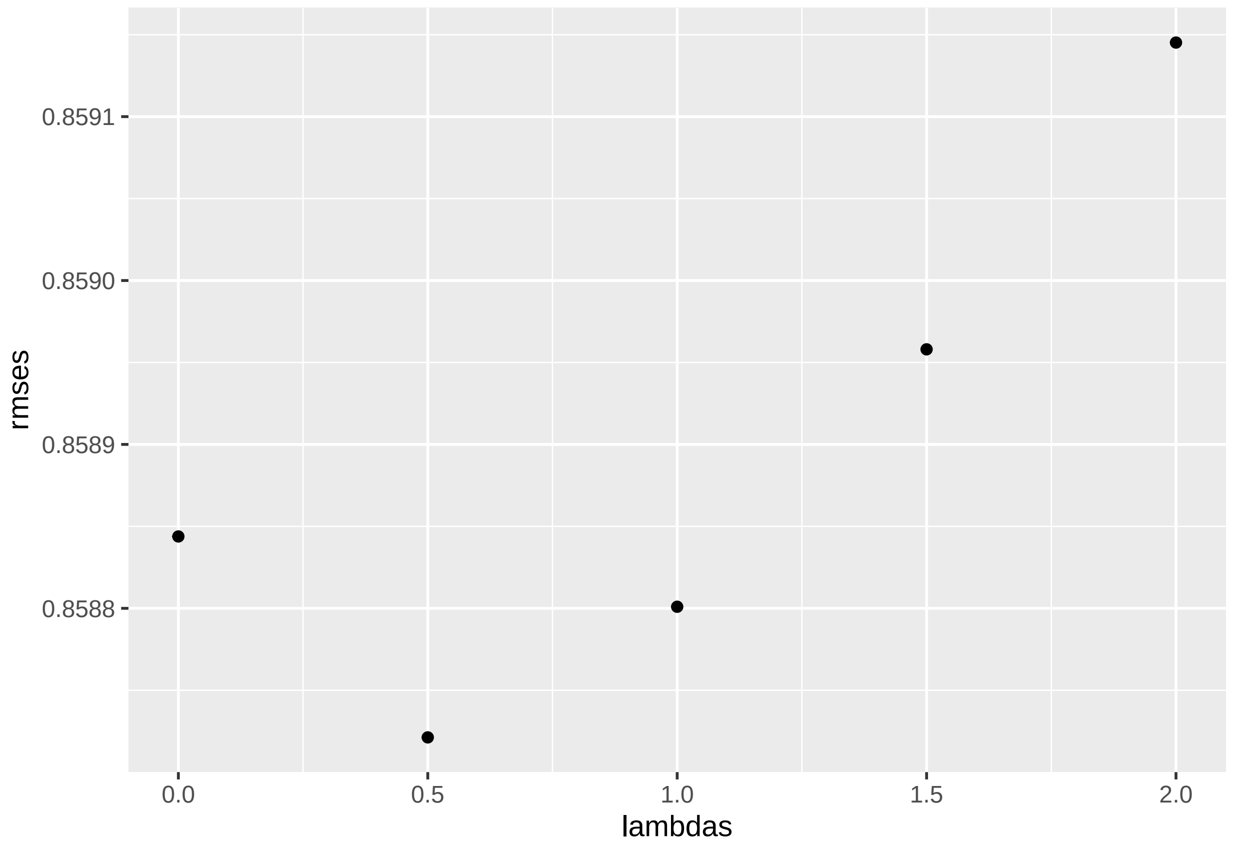
Now that we have encapsulated the model function, we can test the model more easily. The model RMSE score is down to 0.8588439 on the 90% training data partition "A" which we have used throughout the model construction process.

Regularization

We will now analyze performance of the multi effect model, trying some using different penalty values (λ) on the 90% training/10% test partition "A", to attempt further improve the RMSE and reduce over fitting through regularization. The partition has the same proportion of test and training data as the final edx and validation data; hopefully the lambda it produces will provide a good approximate value for use on the final model. This can be a slow operation as it must run the linear regression model for each iteration of the penalty value we are testing.

```
#try lambdas between 0 and 2 at .5 intervals
#this is a slow operation as it must run the model multiple times
lambdas <- seq(0, 2, .5)
```

```r
modstart <- Sys.time()

#interate different regularization settings & return the RMSE of each
rmses <- sapply(lambdas, function(lambda){
  return(multieffectmodel(trainDS,
                          testDS,
                          avgmovieweight,
                          avguserweight,
                          avgyearweight,
                          avgmonthweight,
                          avgpopweight,
                          avggenreeraweight,
                          avggenreuserweight,
                          avggenreeralowcut,
                          avggenreuserlowcut,
                          lambda)$RMSE)
})

#plot the RMSE w/ different lambda values
qplot(lambdas, rmses)
```



```r
#get the lowest RMSE lambda
lambda <- lambdas[which.min(rmses)]
```

A penalty of $\lambda = 0.5$ produces the lowest RMSE on our 90% training / 10% test partition "A".

```
#for consistency - run the model again (once) with the best lambda passed in
#the RMSE should match the regularization loop above
rmse_regularized <- multieffectmodel(trainDS,
                                  testDS,
                                  avgmovieweight,
                                  avguserweight,
                                  avgyearweight,
                                  avgmonthweight,
                                  avgpopweight,
                                  avggenreeraweight,
                                  avggenreuserweight,
                                  avggenreeralowcut,
                                  avggenreuserlowcut,
                                  lambda)

modelstep <- modelstep + 1

#show the results
rmse_results <- bind_rows(rmse_results,
                          data_frame(step = modelstep,
                                     method="Regularized Multi Effect Model - Test",
                                     RMSE = rmse_regularized$RMSE,
                                     benchmark=rmse_regularized$benchmark))
rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1
8	Genre/User Effects Model w/ Low Cut	0.8591753	6.22 secs	0.400	2
9	Multi-Effect Model w/ Bracket	0.8590279	6.21 secs	NA	NA
10	Regularized Multi Effect Model - Test	0.8587213	75.36 secs	NA	NA

Running the model with the selected penalty produces our closest estimate of the final model performance. All of the parameters we will use to tune the model have now been set. We may now proceed to testing the model performance against the final hold out data.

Multi Effect Model Final Test

Now that we've completed the construction, training, and tuning of the multi-effect model, we may finally check the performance against the hold-out validation set provided. To do so, we must re-run the model using the FULL edx dataset as the training set, and the reserved validation/hold-out set as the test set. We will use the weights, filters and penalty value which were approximated by our previous exercises. The result of this model should be considered one of the two final RMSE measures.

```r
# the final test uses the weight, lowcut and lambda values
# derived from the previous code blocks which did not use the validation/hold out data

# using the original 9,000,055 edx data to train the model
trainDS <- edx %>%
           mutate(ratedate = as.Date(as.POSIXct(timestamp, origin="1970-01-01"))) %>%
           mutate(ratemonth = format(ratedate, "%m")) %>%
           mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
           left_join(mv_era, by = "movieyear")

# the 999,999 row validation data is used only to make predictions
# the ratings in this dataset are used only for comparison with predictions to calculate RMSE
testDS <- validation %>%
           mutate(ratedate = as.Date(as.POSIXct(timestamp, origin="1970-01-01"))) %>%
           mutate(ratemonth = format(ratedate, "%m")) %>%
           mutate(movieyear=as.numeric(str_extract(title,"(?<=\\()\\d{4}?(?=\\))"))) %>%
           left_join(mv_era, by = "movieyear")

#run the final model
rmse_validate <- multieffectmodel(trainDS,
                                  testDS,
                                  avgmovieweight,
                                  avguserweight,
                                  avgyearweight,
                                  avgmonthweight,
                                  avgpopweight,
                                  avggenreeraweight,
                                  avggenreuserweight,
                                  avggenreeralowcut,
                                  avggenreuserlowcut,
                                  lambda)

modelstep <- modelstep + 1

#show final results
rmse_results <- bind_rows(rmse_results,
                          data_frame(step = modelstep,
                                     method="Regularized Multi-Effect Model - FINAL",
                                     RMSE = rmse_validate$RMSE,
                                     benchmark=rmse_validate$benchmark))
rmse_results %>% knitr::kable(label="Model Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1
8	Genre/User Effects Model w/ Low Cut	0.8591753	6.22 secs	0.400	2
9	Multi-Effect Model w/ Bracket	0.8590279	6.21 secs	NA	NA
10	Regularized Multi Effect Model - Test	0.8587213	75.36 secs	NA	NA

step	method	RMSE	benchmark	weight	lowcut
11	Regularized Multi-Effect Model - FINAL	0.8571761	82.30 secs	NA	NA

```r
#set a variable for the linear model final RMSE
rmsefinal_lm <- rmse_validate$RMSE
```

When expanding the scope of the predictions to the final validation set, a very slight decrease in the RMSE is observed vs the same model applied to the internal data partitions. We are now training with 9 Mil. rows to predict 999,999 instead of training with 8 Mil. to predict 899,990. The number of predictions increased by 11.1%, and the RMSE decreased by ~ .0015. This provides a reasonable indication that we have not overfit the model.

Based on this result, it appears we have we have achieved the RMSE target for this analysis project. However, one of the requirements of the project was to produce at least two models, of which one must not be a linear regression approach. To meet this requirement, we have one more model to develop. In the following section, we will examine how RMSE could be reduced further using a recommender package called *Recosystem*.

Matrix Factorization with Recosystem

This package was selected for its simplicity of use and performance. Using just a few lines of code, we can apply *matrix factorization* on our training data and use the model to predict ratings on the test set. The package utilizes the LIBMF library, a fast C++ machine learning library which takes advantage of parallel computing to facilitate fast construction of matrices using parallel stochastic gradient descent.

Before we develop a full prediction using matrix factorization (MF) we'll examine how this process works on a smaller subset of the edx data. Our mini partition will randomly select 1 million rows, then perform the same anti-join logic to ensure the training and test data have matching members.

```r
#sample 100000 rows randomly
edx_mini <- sample_n(edx_fortify, 1000000,replace=F)

#90/10 Partition
set.seed(1, sample.kind="Rounding")
edx_mini_train_index <- createDataPartition(y = edx_mini$rating,
                                            times = 1, p = 0.5, list = FALSE)

edx_mini_train <- edx_mini[-edx_mini_train_index,]
edx_mini_temp <- edx_mini[edx_mini_train_index,]

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

# Add rows removed from validation set back into edx set
edx_mini_removed <- anti_join(edx_mini_temp, edx_mini_valid)
edx_mini_train <- rbind(edx_mini_train, edx_mini_removed)

rm(edx_mini_temp)
rm(edx_mini_removed)


#--------------------------------------------------------
```

In this section, we will examine how the matrix factorization approach works. To test this method, we will first create a "mini" dataset with just 10% of the rows.

The files used by this package (which will be created in the default directory) are: trysettest.txt and checksettest.txt which will send predicted ratings to the file: predicttest.txt

The data partitions contains tables with userId, movieId, and rating. Recosystem will produce a prediction list to a text file with the predicted ratings in the same order as the validation data is passed in. The prediction produced by the MF model will then be mutated using the same bracket function as above.

To generate ratings, Recosystem uses the product of a matrix decomposition and multiplication process. More information on the features of this package may be found in in the package vignette and documentation, which is linked from the references section of this report.

```r
#set which datasets to work with
trainDS <- edx_mini_train
testDS <- edx_mini_valid

#edx set is used top train the Recosystem model
trymatrix <- trainDS %>% select(userId, movieId, rating)
trymatrix <- as.matrix(trymatrix)


#validation set is used ONLY to make predicted ratings, thus the rating column may be removed
checkmatrix <- testDS %>% select(userId, movieId)
checkmatrix <- as.matrix(checkmatrix)


# create text files for analysis
write.table(trymatrix , file = "./trysettest.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

write.table(checkmatrix, file = "./checksettest.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

#set the random seed for operations below
# read the files
set.seed(1)
try_set <- data_file("./trysettest.txt")
check_set <- data_file("./checksettest.txt")



#instantiate reco object
obj_reco <- Reco()
modstart <- Sys.time()

# tune training set.
tuner <- obj_reco$tune(try_set,
                       opts = list(dim = c(10,20),
                                    lrate = c(0.1),
                                    costp_l1 = 0, costq_l1 = 0,
                                    nthread = 1, niter = 10))
```

```r
#train the model
obj_reco$train(try_set, opts = c(tuner$min, nthread = 1, niter = 10))
```

```
## iter      tr_rmse          obj
##    0       1.5890  1.7028e+006
##    1       0.9553  8.6640e+005
##    2       0.8966  8.0373e+005
##    3       0.8701  7.7539e+005
##    4       0.8537  7.5868e+005
##    5       0.8408  7.4481e+005
##    6       0.8295  7.3454e+005
##    7       0.8185  7.2579e+005
##    8       0.8073  7.1734e+005
##    9       0.7961  7.0947e+005
```

```r
# Making prediction on validation set and calculate RMSE
# Recosystem saves the data to file reduce memory usage
tf <- "./predicttest.txt"

 #send the prediction to file
obj_reco$predict(check_set, out_file(tf))
```

```
## prediction output generated at ./predicttest.txt
```

```r
modfinish <-Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)
modelstep <- modelstep + 1

#get the prediction
predict_rating <- scan(tf)

 #apply the bracket function
pred_raterounded_mini <- sapply(predict_rating,bracket)

#save the final RMSE + return it
rmsetest_mf <- RMSE(testDS$rating,pred_raterounded_mini)


#show RMSE Results
rmse_results <- bind_rows(rmse_results,
                          data_frame(step = modelstep,
                                     method="Recosystem - Test",
                                     RMSE = rmsetest_mf,
                                     benchmark=benchmark))

rmse_results %>% knitr::kable(label="MF Test Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0

step	method	RMSE	benchmark	weight	lowcut
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1
8	Genre/User Effects Model w/ Low Cut	0.8591753	6.22 secs	0.400	2
9	Multi-Effect Model w/ Bracket	0.8590279	6.21 secs	NA	NA
10	Regularized Multi Effect Model - Test	0.8587213	75.36 secs	NA	NA
11	Regularized Multi-Effect Model - FINAL	0.8571761	82.30 secs	NA	NA
12	Recosystem - Test	0.9402308	99.04 secs	NA	NA

The MF model performs well considering the much smaller amount of reference data available in this mini partition, producing an RMSE of: 0.9402308. This establishes that Recosystem is able to perform better than the baseline.

Now that we have examined the performance of the MF approach using a small subset of the data, we'll go ahead and try this out on the complete edx training & validation set. This process is very processor intensive and may take 20+ minutes to run.

For simplicity, we have applied a fairly standard set of tuning parameters suggested in the package vignette. This model will utilize full edx dataset for training, and then produce a vector of predictions to compare against our unknown validation / hold out data. Therefore, the result of this model should also be considered one of the two "final" RMSE measures.

The files used by this package (which will be created in the default directory) are: trysetfinal.txt and checksetfinal.txt, and the predictions will be written to the file: predictfinal.txt

```r
#set which datasets to work with - setting to FINAL HOLD OUT DATA
trainDS <- edx
testDS <- validation


#edx set is used top train the Recosystem model
trymatrix <- trainDS %>% select(userId, movieId, rating)
trymatrix <- as.matrix(trymatrix)


#validation set is used ONLY to compare to the predicted ratings
checkmatrix <- testDS %>% select(userId, movieId)
checkmatrix <- as.matrix(checkmatrix)


# create text files for analysis
write.table(trymatrix , file = "./trysetfinal.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

write.table(checkmatrix, file = "./checksetfinal.txt" ,
            sep = " " , row.names = FALSE, col.names = FALSE)

#set the random seed for operations below
# read the files
set.seed(1)
try_set <- data_file("./trysetfinal.txt")
check_set <- data_file("./checksetfinal.txt")
```

```r
#instantiate reco object
obj_reco <- Reco()
modstart <- Sys.time()

# tune training set.
tuner <- obj_reco$tune(try_set,
                       opts = list(dim = c(10,20),
                                   lrate = c(0.1),
                                   costp_l1 = 0, costq_l1 = 0,
                                   nthread = 1, niter = 10))


#train the model
obj_reco$train(try_set, opts = c(tuner$min, nthread = 1, niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9666  1.1882e+007
##    1       0.8714  9.8601e+006
##    2       0.8408  9.1871e+006
##    3       0.8218  8.8038e+006
##    4       0.8079  8.5432e+006
##    5       0.7974  8.3571e+006
##    6       0.7891  8.2132e+006
##    7       0.7823  8.1038e+006
##    8       0.7765  8.0121e+006
##    9       0.7716  7.9392e+006
```

```r
# Making prediction on validation set and calculate RMSE
# Recosystem saves the data to file reduce memory usage
tf <- "./predictfinal.txt"

 #send the prediction to file
obj_reco$predict(check_set, out_file(tf))
```

```
## prediction output generated at ./predictfinal.txt
```

```r
modfinish <-Sys.time()
benchmark <- round(difftime(modfinish, modstart, units = "secs"),2)
modelstep <- modelstep + 1

#get the prediction
predict_rating <- scan(tf)

 #apply the bracket function
pred_raterounded <- sapply(predict_rating,bracket)

#save the final RMSE + return it
rmsefinal_mf <- RMSE(testDS$rating,pred_raterounded)

#show RMSE Results
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(step = modelstep,
                                     method="Recosystem - FINAL",
                                     RMSE = rmsefinal_mf,
                                     benchmark=benchmark))

rmse_results %>% knitr::kable(label="MF Final Results")
```

step	method	RMSE	benchmark	weight	lowcut
1	Baseline	1.0615912	0.02 secs	1.000	0
2	Movie Effect Model	0.9440221	0.85 secs	0.995	0
3	User Effect Model	0.8663733	2.65 secs	0.955	0
4	Year Effect Model	0.8660561	3.61 secs	1.030	0
5	Rate Month Effect Model	0.8660535	6.37 secs	0.600	0
6	Popularity Effect Model	0.8659055	9.74 secs	1.000	0
7	Genre/Era Effects Model w/ Low Cut	0.8657287	1.43 secs	0.800	1
8	Genre/User Effects Model w/ Low Cut	0.8591753	6.22 secs	0.400	2
9	Multi-Effect Model w/ Bracket	0.8590279	6.21 secs	NA	NA
10	Regularized Multi Effect Model - Test	0.8587213	75.36 secs	NA	NA
11	Regularized Multi-Effect Model - FINAL	0.8571761	82.30 secs	NA	NA
12	Recosystem - Test	0.9402308	99.04 secs	NA	NA
13	Recosystem - FINAL	0.7979198	1048.50 secs	NA	NA

The Recosystem matrix factorization approach has produced the best yet RMSE yet, resulting in a substantial improvement and a final RMSE score of 0.7979198. This is a fairly dramatic improvement over the performance of the multi-effect model. The effectiveness of matrix factorization is clearly demonstrated by this simple, efficient package.

Results

The multi-effect model offers a relatively easy-to-understand approach, but lacks the power of the more computationally intensive matrix factorization method. The multi-effect linear regression model combines the rating effects of the movie, user, year, and month of the rating, the popularity, and the genres (in the same era), as well as users genre preferences, to reduce the RMSE of our guesses from 1.0615912 to 0.8571761. A matrix factorization approach using the Recosystem package yielded an even greater improvement in RMSE - with a lot less code - reducing the final RMSE to 0.7979198. This comes at some cost of performance however, taking ~10-20 minutes to run the final model depending on tuning parameters used.

Model Performance

This project was compiled and run on a 64-bit Windows 10 PC with an Intel Core i7-6700HQ CPU @ 2.60 GHz and 32 GB RAM using RStudio Version 1.2.5033 and R x64 version 3.6.2.

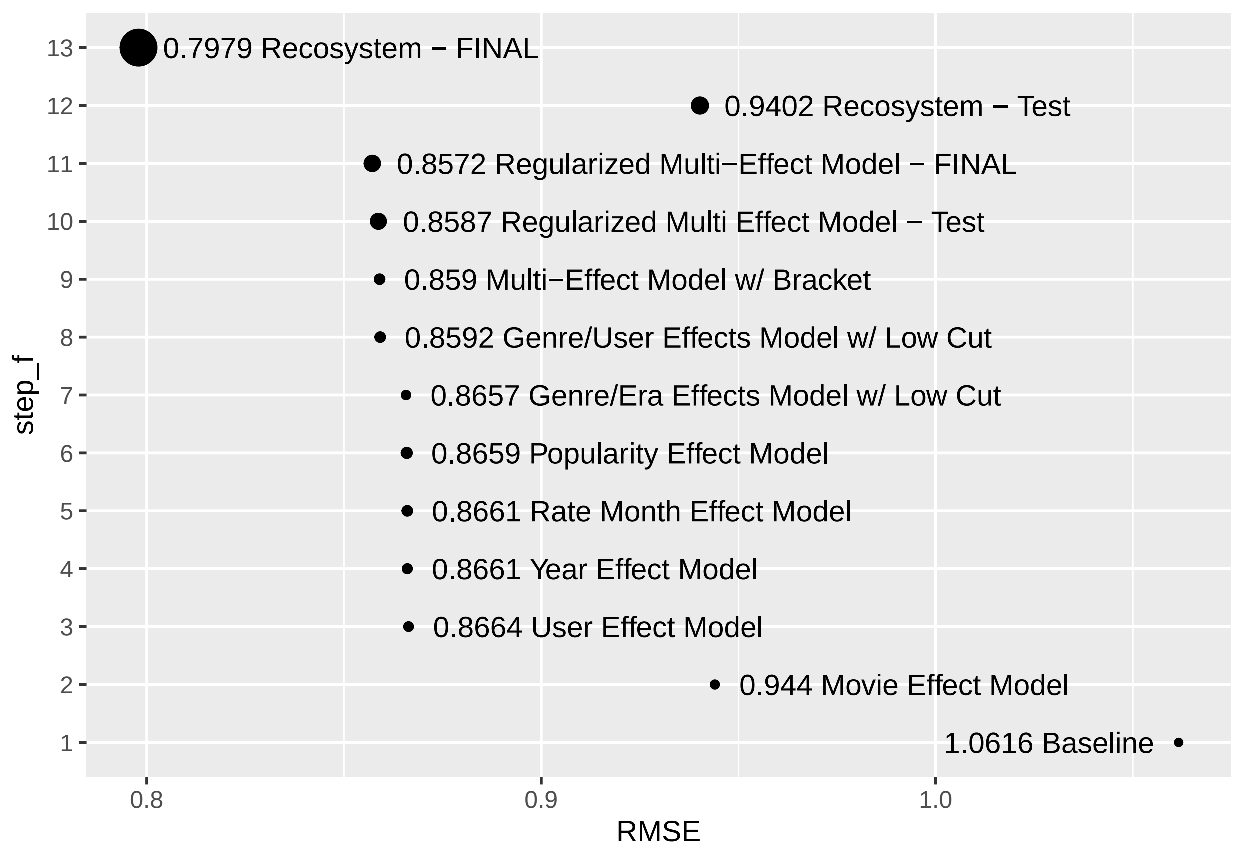
```r
#display version info
version
```

```
##                 _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.2
## year           2019
## month          12
## day            12
## svn rev        77560
## language       R
## version.string R version 3.6.2 (2019-12-12)
## nickname       Dark and Stormy Night
```

The following chart illustrates how RMSE changed with each successive step. Point size corresponds to processing time for each step. There is clearly some trade off between accuracy and performance, with some steps yielding relatively small gains in accuracy while increasing execution time substantially.

Note that the Recosystem steps incorporate cross-validation strategies, and take considerably longer as they are more computationally intensive and through many iterations. So these are not really *apples to apples* vs the regression model single-run times. However, they are informative as to the relative time cost of each approach.

```
#show table of RMSE results
rmse_results %>%
  mutate(step_f = factor(step, levels = c(1:modelstep))) %>%
  arrange(-step_f) %>%
  ggplot() +
  geom_point(aes(x=RMSE,
                 y=step_f,
                 size=as.double(benchmark,units="secs")/10)) +
  geom_text(aes(label = paste('  ',round(RMSE,4),method,'  '),
                x=RMSE,
                y=step_f,
                hjust = ifelse(RMSE<1,0,1)
                )) +
  theme(legend.position = "none")
```



Conclusion

After attempting various approaches to building a regression model, an RMSE reduction of about 20% was achieved. It is surprising to find that the Recosystem model - using only the userId, movieId and ratings as predictors - was able to produce a lower RMSE than a hand-made regression model. This demonstrates the advantage of matrix factorization (MF) for prediction of ratings. Recosystem can improve accuracy by uncovering latent factors and patterns in the data that may not be very evident to a human. This mathematical approach can out-perform a fairly complex linear regression model, even using in-depth data analysis and multiple, hand-selected predictors.

Limitations

There may be additional correlations and patterns to explore with further data analysis. Constructing additional data relations in the linear regression model could further optimize the RMSE. Adding more filters to reduce noise in the data and filtering out more outliers might also help. There may also be valuable data that could be gleaned from the words in the movie titles - an area which we felt was beyond the scope of this project.

As to the MF model - Recosystem is just one of the R packages available that concern recommendations, and some other packages may offer more different model options and algorithms. Recommender packages may provide better performance, but sometimes suffer from the *black box* effect - they can be difficult to understand or see how the model really operates. Some of these models also rely on having some data available to make predictions. This can be problematic in *cold start* scenarios, when new movies or users are added. Some models may require special handling for these scenarios.

Tuning Considerations

Several parameters we used throughout the regression model are candidates for further tuning. It may be possible to provide better results through more cross validation, data filtering, changing the genre encoding strategy, combining or altering the year ranges of our eras, or through further tuning of weights and penalty values.

For convenience, in the looping training operations, the range of values shown has been reduced and centered around the best RMSE - any new effects added should explore a broad range of values for weights and other parameters being tested.

The Recosystem package offers many built-in options for tuning and we experimented with several to get to a version that produced a fair RMSE improvement and didn't take too long to run - adding more iterations or changing penalty value settings in Recosystem may also further refine the model performance.

Commentary

Thank you for taking the time to review this report. This was my first formal data science report using R and R Markdown. I welcome all feedback, corrections, suggestions and ideas. Author: Ryan J. Cooper - info@ryancooper.com - All Rights Reserved

http://www.ryancooper.com

https://github.com/elaborative

References

Irizzary, Rafael, Introduction to Data Science, github page, https://rafalab.github.io/dsbook/, 2020.

The Netflix Prize, web site, https://www.netflixprize.com/rules.html, 2020.

Wikipedia contributors, Matrix Completion, Wikipedia, web site, https://en.wikipedia.org/wiki/Matrix_completion, 2020

Qiu, Yixuan, Recommender System Using Parallel Matrix Factorization, github page, https://github.com/yixuan/recosystem, 2020

Qiu, Yixuan, et. al., Recosystem Package, CRAN, https://cran.r-project.org/web/packages/recosystem/recosystem.pdf, 2020

Chin, Wei-Sheng et. al., A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf, 2015

University of Minnesota, Understanding Media and Culture: 8.2; The History of Movies, web site, https://open.lib.umn.edu/mediaandculture/chapter/8-2-the-history-of-movies/, 2020

Wikipedia contributors, Film Preservation, Wikipedia, web site, https://en.wikipedia.org/wiki/Film_preservation, 2020