MovieLens Report

Alex McIntosh

Introduction

GroupLens, a social computing research organization at the University of Minnesota, created and shared the rating data sets from the MovieLens website (https://movielens.org). The data set presented in the analysis to follow includes 10 million user ratings. The raw data set is not that useful. A human can not look at that quantity of information and draw many valuable conclusions. However, through the application of data science insights may be gleaned and tools may be built. For example, a movie recommendation system.

The goal of this projects is to develop a system that can predict how a user will rate a movie. The metric by which to measure success will be the Root Mean Squared Error (RMSE) between how the model predicts a user will rate a movie and how the user actually rated the movie. Success is considered to be a model that results in an RMSE of less than 0.86490.

The key steps are preparation, exploration, determination of the model, and validating the set to find the final RMSE. Preparing the data will involve downloading, cleaning, and tidying the data set. Exploring the data will include stratifying the data to see what predictors can be easily picked out. Determining the model will require taking into account the massive size of the data set, as some methods will not be physically possible, and narrowing to predictors that will have a significant impact. Validating the set will require having split off a portion of the data set before constructing the model to use as a clean to set to test the model against.

Methods and Analysis

The following analysis includes functions from the tidyverse, caret, data.table, lubridate, gridExtra, and randomForest packages. Five significant digits are displays in order to check proximity to the target RMSE, 0.86490.

Cleaning the Data

The MovieLens data set must be downloaded from https://files.grouplens.org/datasets/movielens/ml-10m.zip.

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

With the file the downloaded, it must be parsed and cleaned for easier condensation. A movies data frame holds the information with columns: userId(int), movieId(int), rating(float), timestamp(int), title(string), genres(string, separated by |). Further manipulation of this dataset will occur later.

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

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

colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>%
```

```
  mutate(movieId = as.numeric(movieId),
         title = as.character(title),
         genres = as.character(genres))

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

To generate tenable results from the model, the movie data must be split into an edx and validation set using createDataPartition. 10% of the data set will be preserved in the validation set.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

The following code ensures that userId and movieId in validation set are also present in the edx data set. This is important so that we have the ability to make predictions on all userId's and movieId's in the validation set.

```
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

Throughout, unused variables are removed to conserve RAM. It is not uncommon to spike up to 7 GB of RAM for calculations that run on the entire set.

Data Exploration

The data is local and clean. Let's take a look.

```
dim(edx)
```

```
## [1] 9000055       6
```

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

There are 9000055 observations in the set, (~90% of 10 million) with 6 variables each. The timestamp is saved as an int (the number of seconds since New Years 1970). The year a movie is released is included in its title. Genres are a string separated by |'s.

Categorical Data

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

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

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

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

```
n_distinct(edx$title)
```

```
## [1] 10676
```

```
n_distinct(edx$genres)
```

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

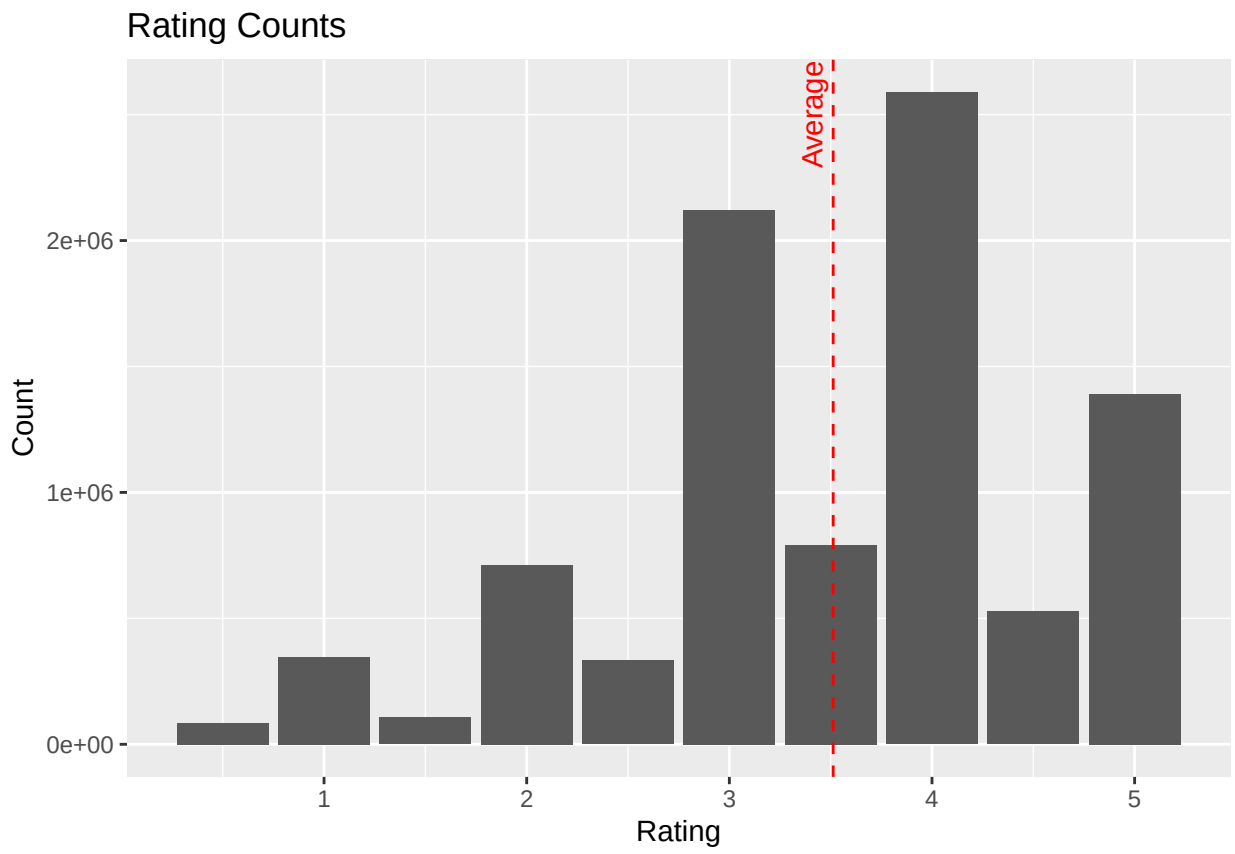
```
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  distinct(genres) %>%
  count() %>%
  pull(n)
```

```
## [1] 20
```

There are 69878 distinct users. 10677 distinct movies, but only 10676 distinct movie titles. 797 distinct combined genres, but only 20 distinct individual genres.

Distribution of Ratings

```
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot() +
  labs(title = "Rating Counts", x = "Rating", y = "Count") +
  geom_col(aes(x = rating, y = count)) +
  geom_vline(xintercept = mean(edx$rating), linetype = "dashed", color = "red") +
  annotate("text",
           x = mean(edx$rating),
           y = 2.5 * 10 ^ 6,
           label = "Average\n", color = "red",
           angle = 90)
```




```
mean(edx$rating)
```

```
## [1] 3.5125
```

Unsurprisingly, most user rate things as 3 or 4, with 4 being the most common rating. Many fewer people used a half rating, around 5 times fewer. The data does not follow a normal distribution. An average rating of about 3.5 and the distribtuion suggest users are averse to low and very high ratings.

Time

```
edx <- edx %>%
  mutate(date = as_datetime(timestamp))
range(edx$date)
```

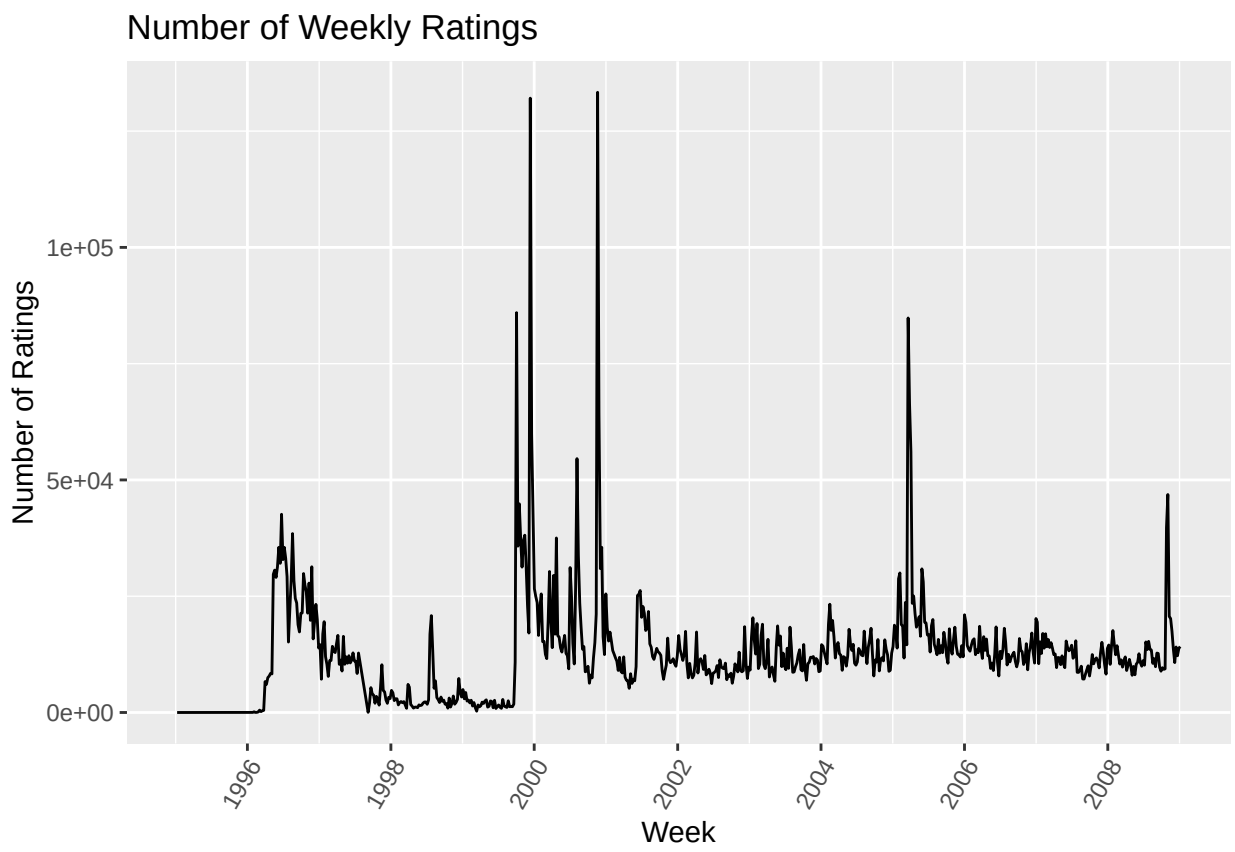
```
## [1] "1995-01-09 11:46:49 UTC" "2009-01-05 05:02:16 UTC"
```

Our observations range from January 9th, 1995 at 11:46 AM to January 5th, 2009, at 5:02 AM. Note that the date was saved to a new column in the data set, as it is often referenced and takes some time to run.

For a similar reason, a column is added to the edx data set to round the date to the nearest week. Below is a graph indicating the number of ratings the website received each week. Despite a week being an arbitrary unit to round to, it was valuable in building the model, shown later.

```
edx <- edx %>%
  mutate(week = round_date(date, unit = "week")) %>%
  mutate(week = as_date(week))
edx %>%
  group_by(week) %>%
```

```
  summarize(count = n()) %>%
  ggplot(aes(x = week, y = count)) +
  labs(title = "Number of Weekly Ratings",
       x = "Week",
       y = "Number of Ratings") +
  theme(axis.text.x=element_text(angle=60, hjust=1)) +
  scale_x_date(date_labels = "%Y",
                   date_breaks = "2 year") +
  geom_line()
```



The above graph reveals some interesting out-liers. There is a spike from 1996 to 1997 Then, a huge spike in late 1999, 2000, 2001, and 2005. What are these spikes? what movies are being rated during these spikes? What sort of users were being attracted to the site to submit these reviews? The data seem otherwise evenly distributed.

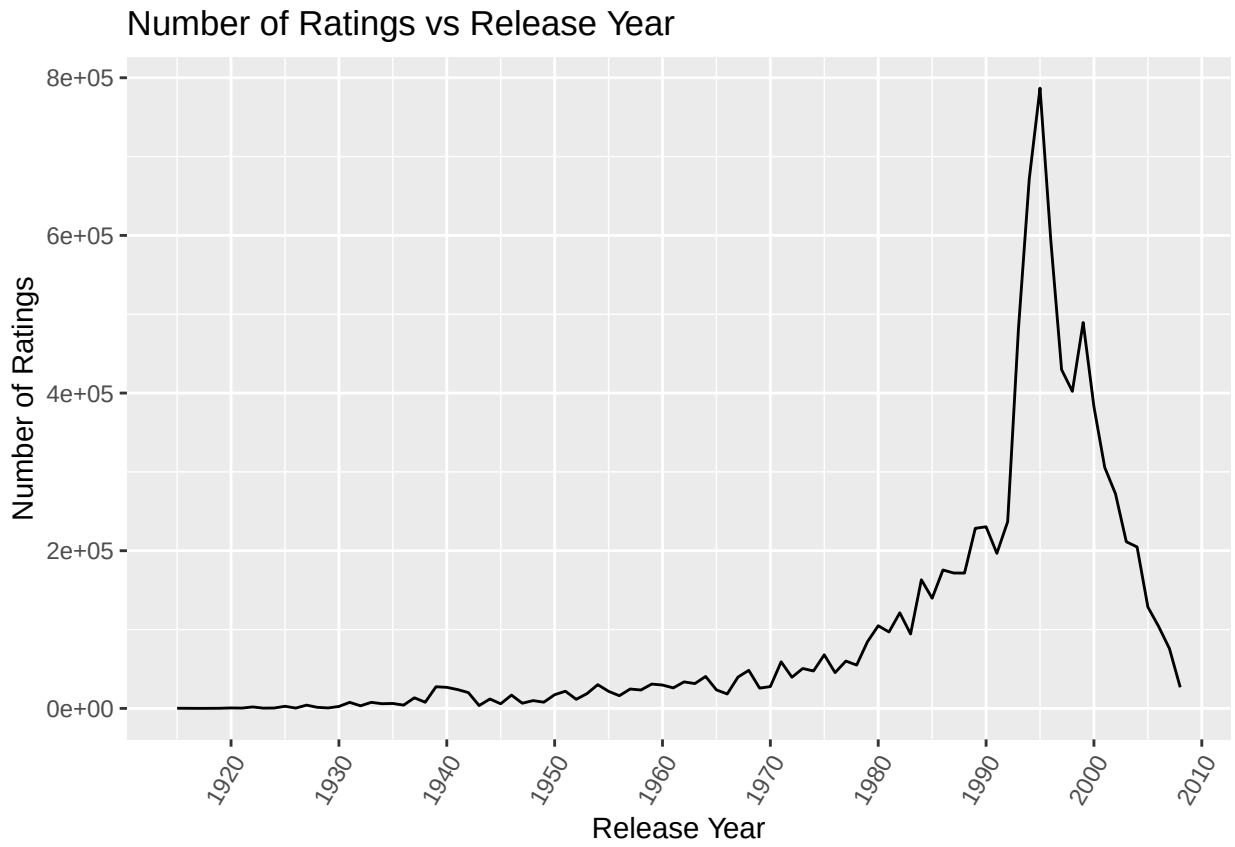
Now let us extract the release year from the title.

```
pattern <- " \\(\\d{4}\\)$"
edx <- edx %>%
  as_tibble() %>%
  mutate(release_year = str_extract(title, pattern),
         release_year = str_remove_all(release_year, " \\(|\\)"),
         release_year = as.integer(release_year),
         title = str_remove(title, pattern))
```

```
range(edx$release_year)
```

```
## [1] 1915 2008
```

```
edx %>%
  group_by(release_year) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = release_year, y = count)) +
  labs(title = "Number of Ratings vs Release Year",
       x = "Release Year",
       y = "Number of Ratings") +
  theme(axis.text.x=element_text(angle=60, hjust=1)) +
  scale_x_continuous(breaks = seq(1910, 2010, 10)) +
  geom_line()
```

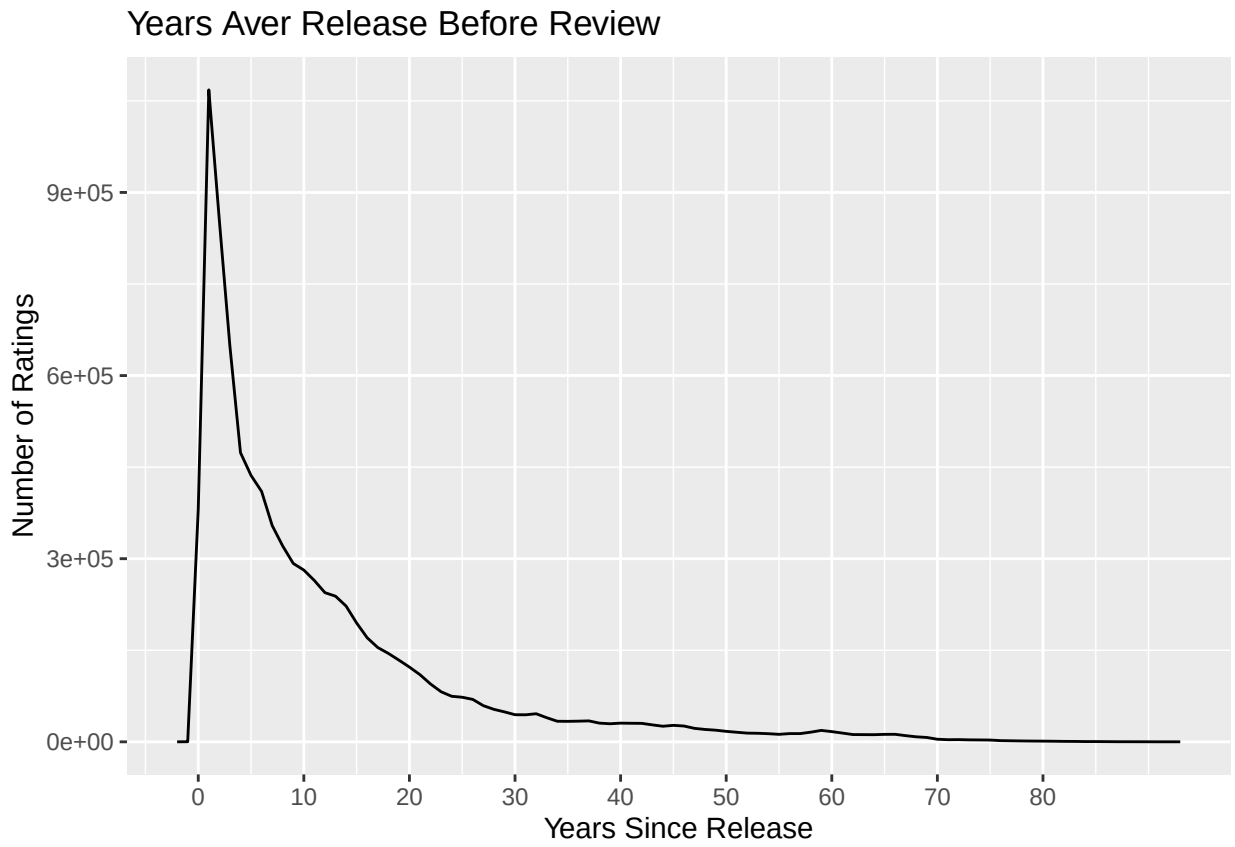


The movies were all released between 1915 and 2008. A lot of ratings were of movies released in the mid to late 90s. This may reveal some information about the average age of a user of this rating system.

Years After Release Before Review

```
edx <- edx %>%
  mutate(years_since_release = year(date) - release_year)
edx %>%
  group_by(years_since_release) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = years_since_release, y = count)) +
  labs(title = "Years Aver Release Before Review",
       x = "Years Since Release",
       y = "Number of Ratings") +
  scale_x_continuous(breaks = seq(0, 80, 10)) +
```

```
geom_line()
```



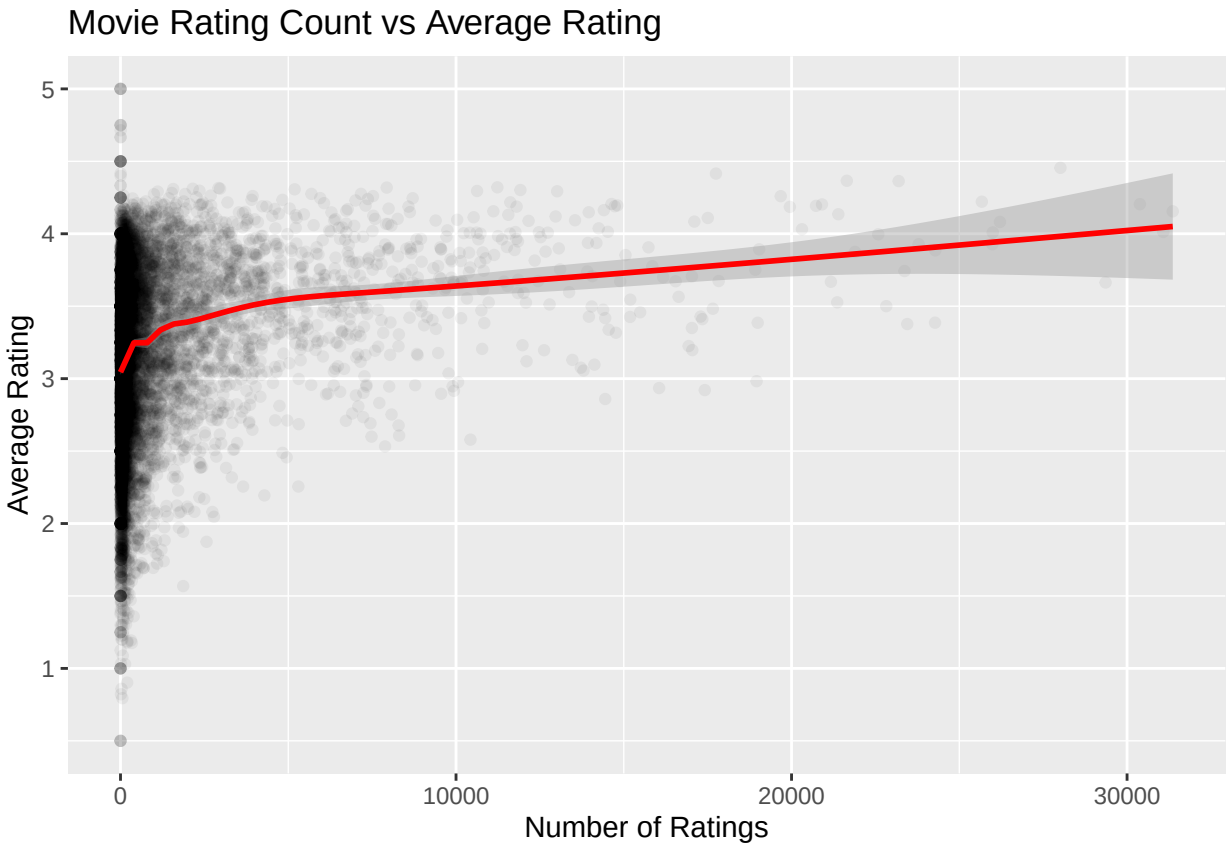
Most reviews occured within a few years of release. There also exist some ratings that occurred before release, assumedly by critics or those with early access to the movie.

Checking Relations

The previous data exploration focused on seeing the number of ratings and trying to build a picture of the types of users making reviews and the types of movies being reviewed. Now, lets see what effects these have on the ratings themselves.

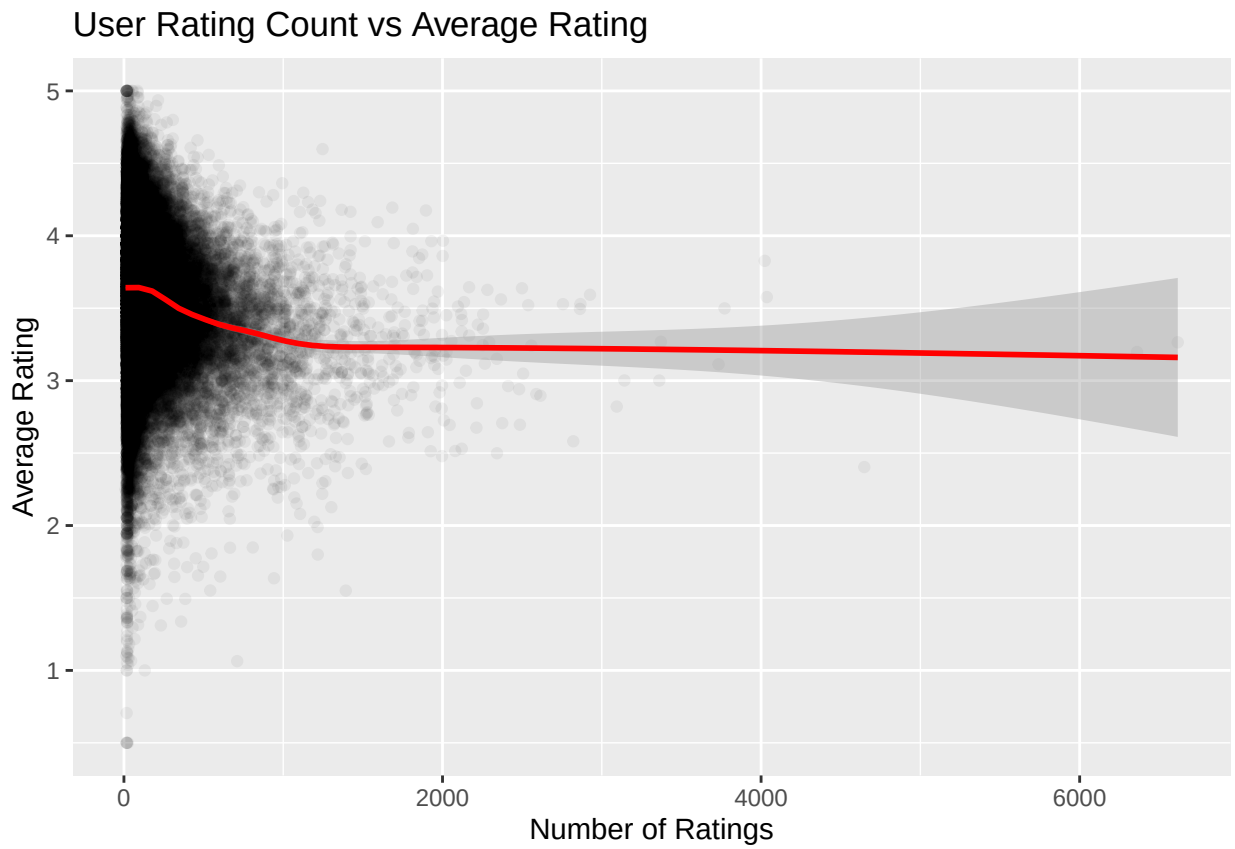
To reduce code, I will only show the plot outputs for the following checks.

Number of Movie Ratings vs Average Rating



The number of ratings a movie receives appears to positively correlate with its rating. Most movies have received fewer than 5000 ratings. Additionally, there are very few movies with a significant number of reviews which average over 4.5 or below 1.5

Number of User Ratings vs Average User Rating

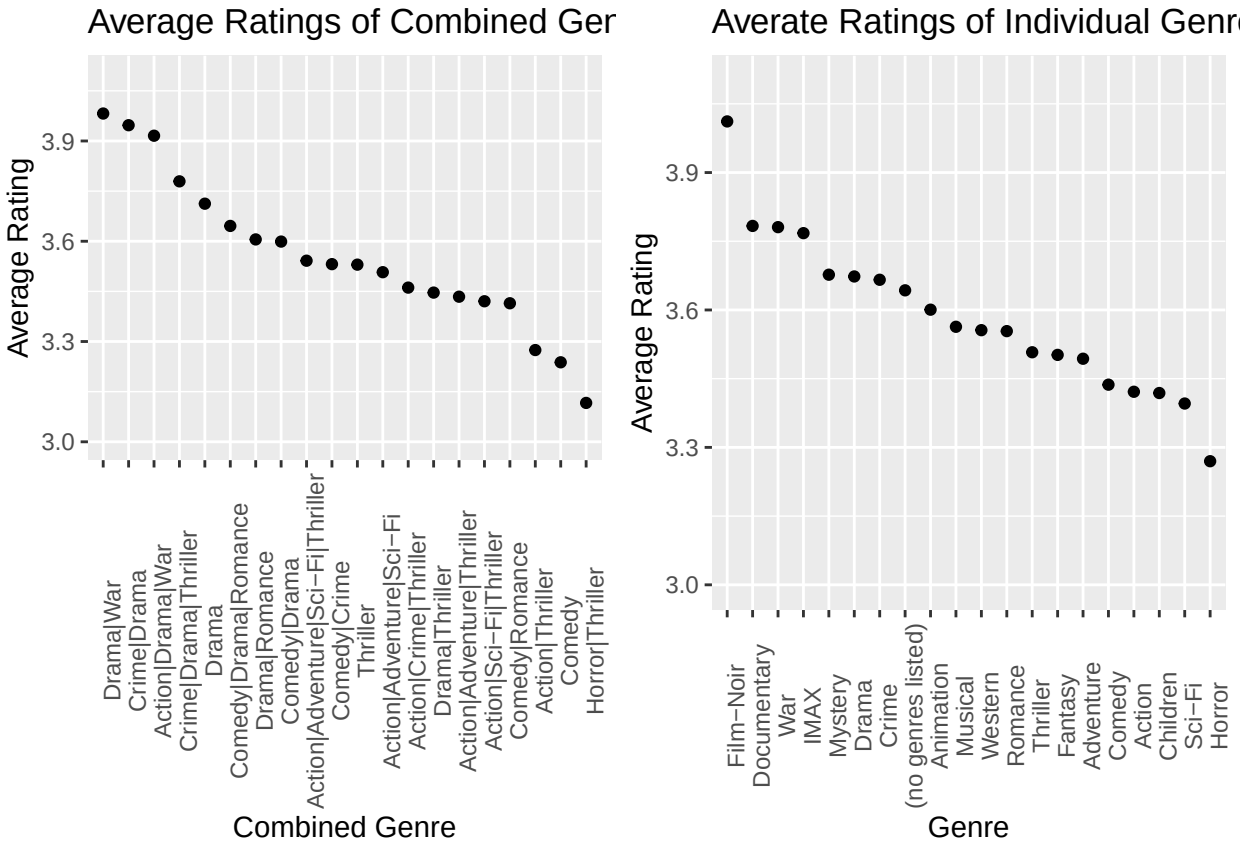


Interestingly, we see a regression towards the mean. Fewer ratings from users appear to correspond to higher average ratings. Those users who submitted more about 1000 votes vote, on average, float much closer to the overall average. This means we might suspect newer users to rate movies generally more highly but with much more variation.

There are also some out-liers (n_ratings $>$ 6000) that appear suspicious.

Genre Effect

For looking at the genre effect, we have to make some design decisions. Is it better to consider each genre as unique? Or is a combined genre to be considered distinct? For example, is an action|comedy just an action and a comedy, or does it become somethine else?

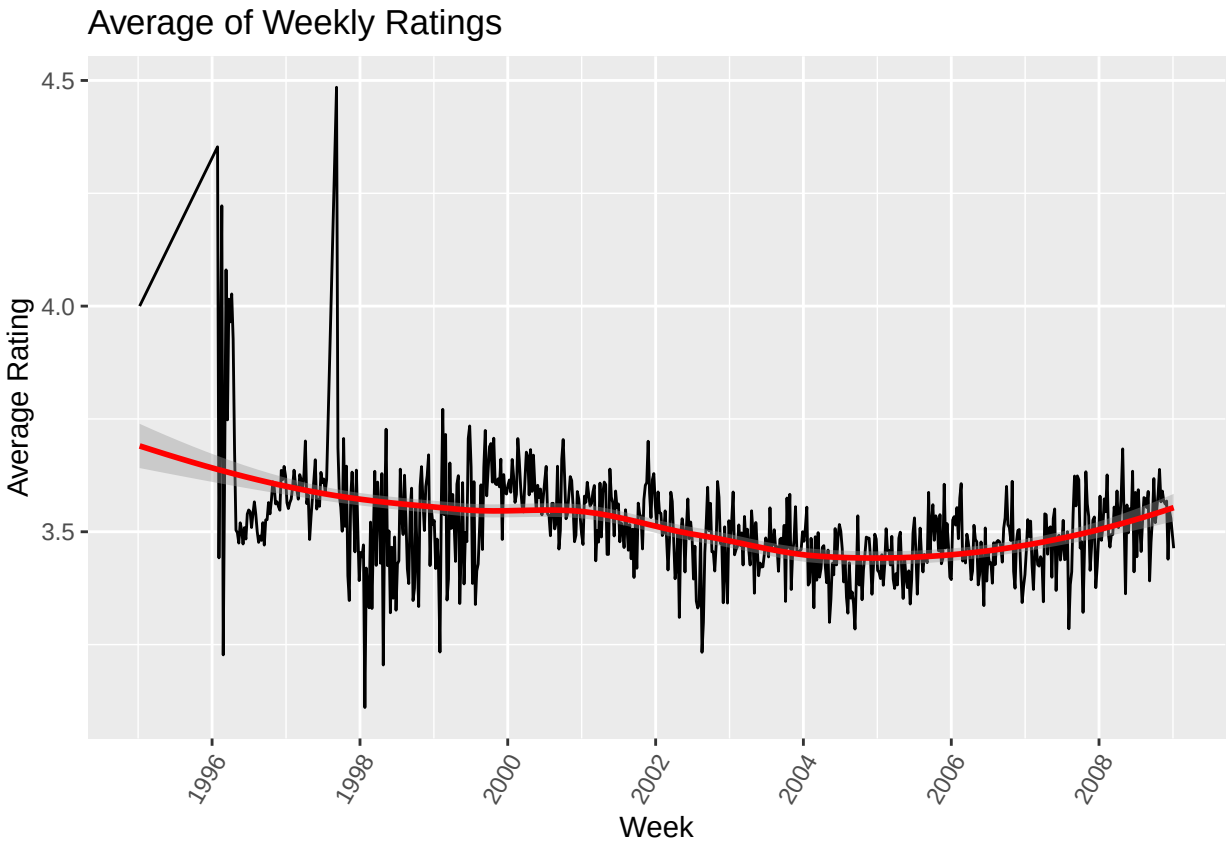


In the left graph, note how overlapping genres clump: Mystery/ Drama / Crime, Documentary / War.

The second graph suggests that the individual impact of each genre is less important than the combined genre, as noted by how little values can deviate from the mean, even for genres with lots of rating.

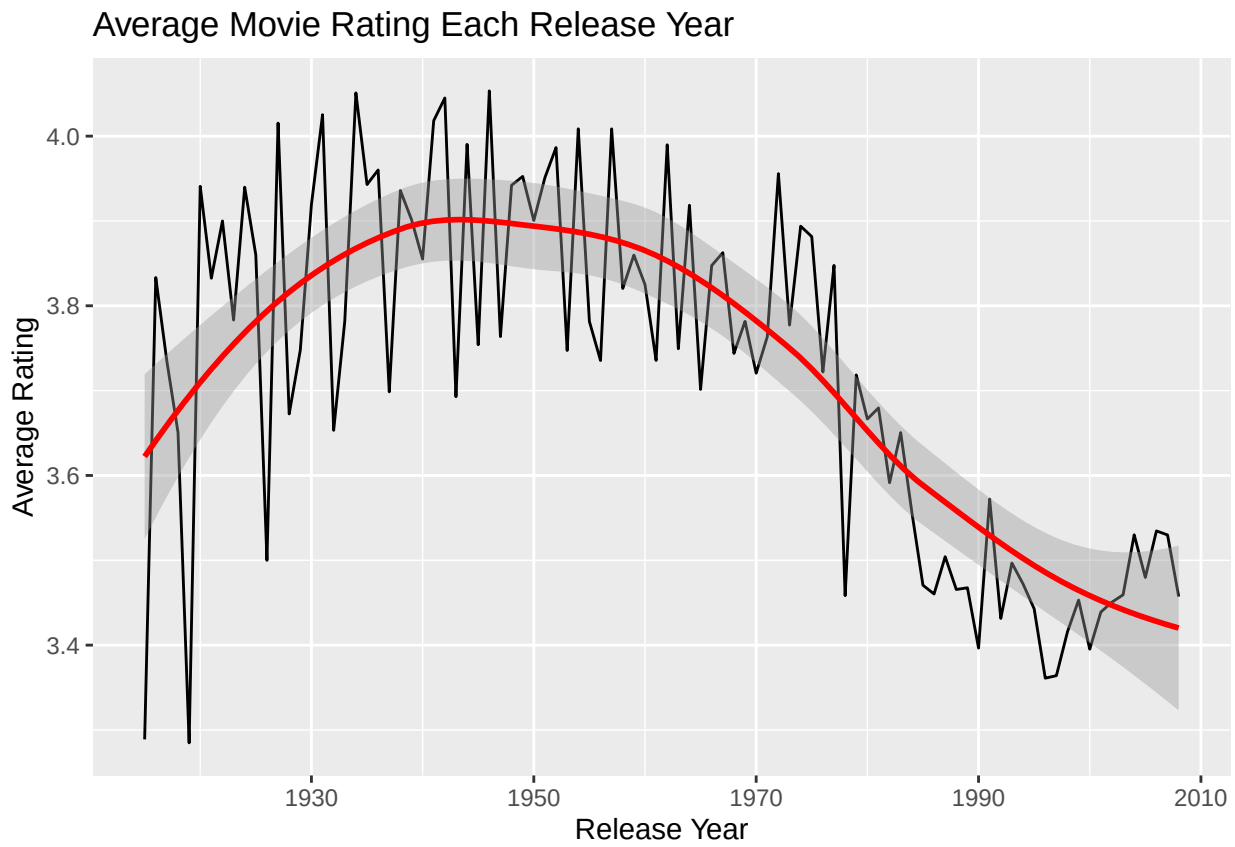
This makes sense if we suspect that genre is a relatively arbitrary categorization and is enhanced by dimensionality as opposed to categorization.

Timestamp vs Average Rating



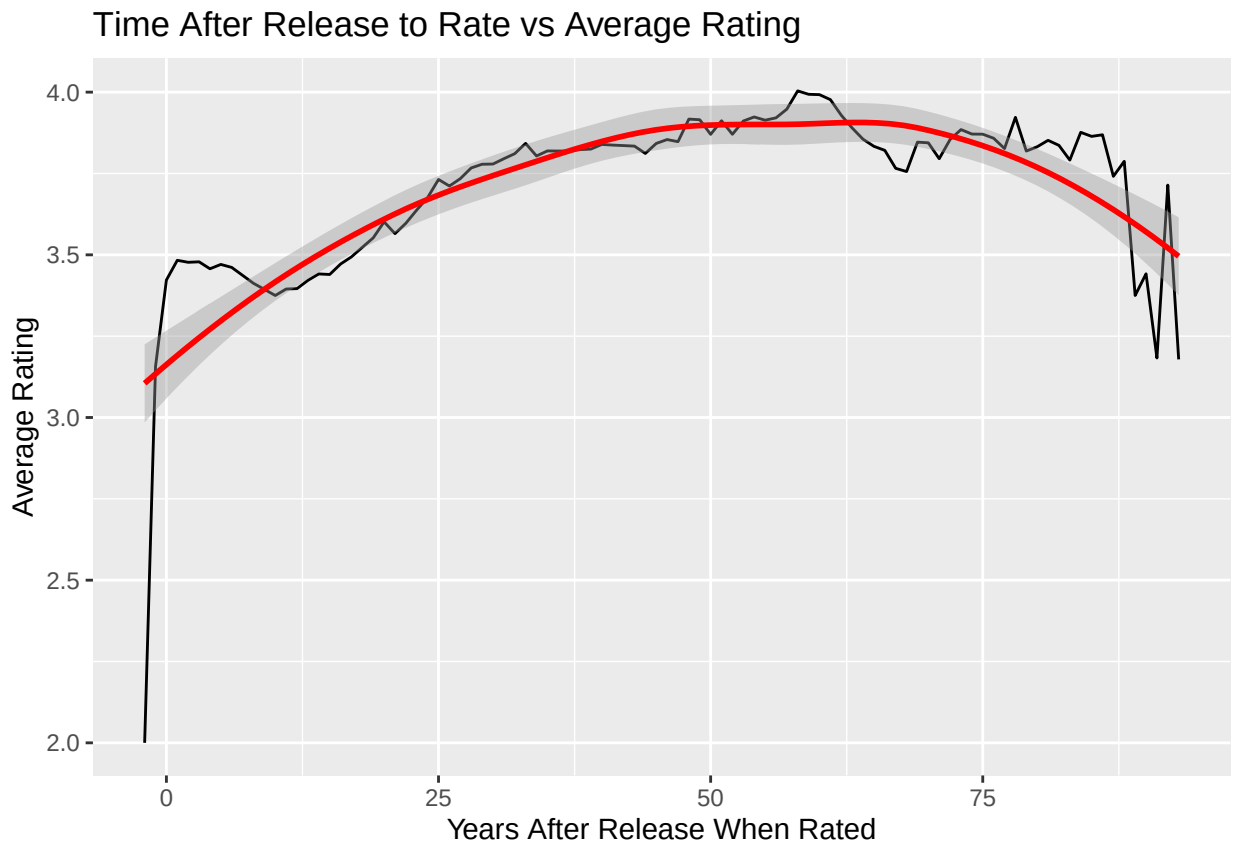
This data is harder to nail down. Ratings early in the existence of MovieLens skew very high, likely because people interested in rating movies tend to be movie fans and thus rate them higher. Within a few months, the data fall back to a pattern. There is not much distance from the mean. With the exception of 1997 to 1999, the data appears to be periodic.

Release Year vs Average Rating



The above graph reveals a near half-star increase in average rating for movies released around 1950. However, being released after about 1985 seems to have a largely negative impact on a movie's average rating. Are we seeing a nostalgia effect? Is it that the movies users are still watching from the 1950 must be better? Does it have to do with the shear quanitity of movies released in the last 30 years?

Years Since Release vs Average Rating



The previous two plots can also be considered in a different way: What if the impact is coming from the distance between release and review? The distance could be caused by a number of factors: Waiting to watch the movie after its release, waiting to review the movie. Regardless, this shows a much smoother relationship and deviation from the mean suggests it may be a stronger variable with which to make predictions.

Results

Now that the exploratory analysis is complete, lets look at the impact incorporating various effects (or biases) will have on the resulting RMSE. A possible solution to constructing a recommendation system is to base it on a predictive model. We assume ratings from other users, the user's own ratings of other movies, and more factors will impact a user's rating of a movie.

Because we expect to find a single variable, rating, to be dependent on many other variables, multiple linear regression is a modelling approach worth considering. However, because of the size and sparseness of the data set (most users have not rated most movies), the lm function will not be an effective strategy. But we know that by sweeping the expected value from the data set, what remains can be treated as an effect on the data. In the following section, we will determine enough such effects to reach the target RMSE.

Splitting EDX data into training and testing data

We do not want to use the validation set to construct our model. To evaluate the efficacy of our model, we want to compare it to a test set. So, the edx set is split into a training data set and a test data set. Splitting the data is accomplished by creating a data partition with the caret package. %20 of the edx data go to the test set. For reproduceability, the seed is set to 1 again (no rounding).

```
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx %>%
  slice(-test_index)
temp <- edx %>%
  slice(test_index)
```

Again, let us make sure userId and movieId present in test_set are also in train_set

```
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set)
train_set <- bind_rows(train_set, removed)
```

Metric of Success

For this analysis, the root mean squared estimate (RMSE) is be the metric to determine the effectiveness of the model. The goal is the minimize the RMSE. Overtraining is undesirable, as it will not translate to the final validation.

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

Base Line

First, I would like to establish a base line to make improvements to the model easier to measure. The following is based on 34.7.4 from the test, A First Model.

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

```
## [1] 3.5125
```

The average movie rating is 3.5125, similar to the value that was found above for the entire data set.

Naive RMSE

To get an understanding of the base line, lets check the effectiveness of simply predicting unknown ratings as the average of all (training) ratings. Mostly I just like the sound of "Naive RMSE."

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.0599
```

```
results <- tibble(method = "Average", RMSE = naive_rmse)
```

1.059904

Modeling the Movie Effect

Now, lets add the first effect to the model. The mean will be swept out so that we are determining the effect only from the residuals. The movie effect is based on the idea that if a given movie is rated highly, we can expect someone else to rate it highly, too.

```r
# Movie Effect after centering on the mean rating (mu_hat)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

# Make predictions by applying the movie effect to each movie in the test set
predicted_ratings <- mu_hat + test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)

movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_effect_rmse
```

```
## [1] 0.94374
```

```r
results <- results %>%
  add_row(method = "Movie Effect", RMSE = movie_effect_rmse)
```

With this one change, the RMSE has dropped over 0.1 to 0.94374. If every effect impacted the RMSE this much (it does not), then only a few effects would be sufficient to hit the target.

Adding the User Effect

A similar approach is taken to determine the user effect. This time, we only want to look at the effect on rating after the average and the movie effect have been removed.

```r
# User Effect after centering on the mean and sweeping the Movie Effect
user_avgs <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

# Make predictions by applying the movie effect and user effect across movies
# and users.
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)

movie_user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_user_effect_rmse
```

```
## [1] 0.86593
```

```r
results <- results %>%
  add_row(method = "Movie + User Effect", RMSE = movie_user_effect_rmse)
```

An RMSE of 0.86593 shows a noticeable improvement, but suggests we may already be running into diminishing returns. It is unsurprising that we are able to make predictions of users based on previous ratings.

The model is starting to look like something reasonable. Of course, this is only as far as the textbook went with the model. What other ingredients might be added to push it over the edge?

Choosing a Genre Effect

It seems reasonable to assume, from the exploratory analysis, that genre will have an impact on average ratings of a movie. But much of this effect may have been hidden within movie and user effects, so it is important to confirm the RMSE drops sufficiently with the addition of another effect. There are diminishing returns for each effect, so finding the right few to represent the model and draw conclusions is important.

Combined Genres

The genre data were presented as a string separated with |'s. The simplest model would be to assume that each combination of genres can be considered its own genre. That is, let us briefly assume that action|comedy and action|drama are completely orthogonal predictors.

```
# Mean rating of each genre after centering on the mean and sweeping movie
# effect and user effect
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))

# Make predictions based on movie, user, and genre effect
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)

movie_user_genre_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_user_genre_effect_rmse
```

```
## [1] 0.86559
```

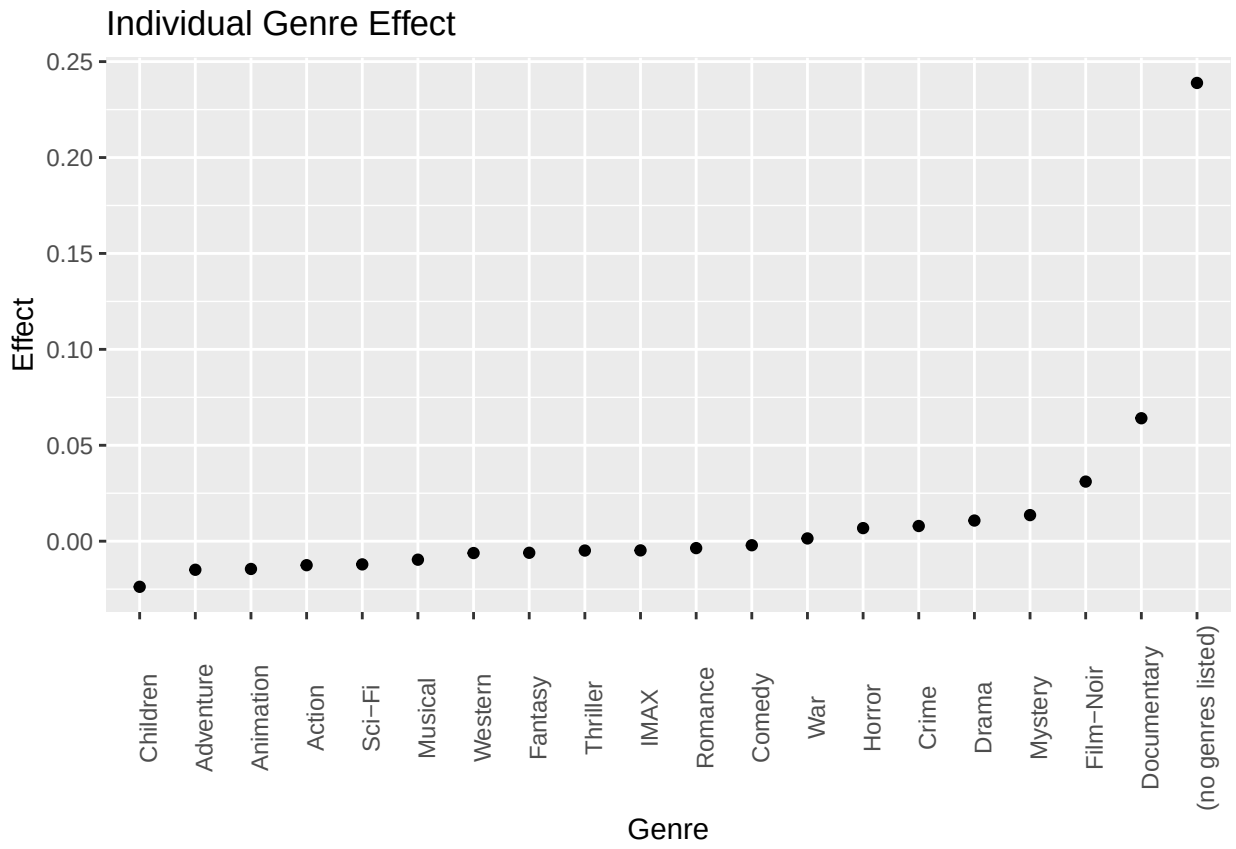
```
results <- results %>%
  add_row(method = "Movie + User + Genres Effect", RMSE = movie_user_genre_effect_rmse)
```

Can we do better with a more complex model?

Genre Splitting

The following code takes awhile to process because of the string operations, but it gives us a chance to see what effect each individual genre has.

```
split_genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))
split_genre_avgs %>%
  ggplot(aes(x = reorder(genres, b_g), y = b_g)) +
  labs(title = "Individual Genre Effect",
       x = "Genre",
       y = "Effect") +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90))
```




```
combined_genre_avgs <- train_set %>%
  distinct(genres) %>%
  mutate(b_g = map_dbl(genres, function(g){
    sum(split_genre_avgs$b_g[str_detect(split_genre_avgs$genres, g)])
  }))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(combined_genre_avgs, by = "genres") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)
movie_user_sep_genre_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_user_sep_genre_effect_rmse
```

```
## [1] 0.86585
```

```
results <- results %>%
  add_row(method = "Movie + User + Separated Genres Effect", RMSE = movie_user_sep_genre_effect_rmse)
```

This increase in error suggests that there is more to a combined genre than being simply the sum of its parts. For this reason, I will only keep the first genre effect going forward, as it had a larger impact on reducing the RMSE.

Effects of Timing

Thus far, the factors incorporated into the multiple linear regression model have been what might seem intuitive. They have also been categorical. But we are provided far more data than movie, user, and genre in a rating. We saw in our exploration that there seems to be an effect on ratings based on when they are made.

Week of Review

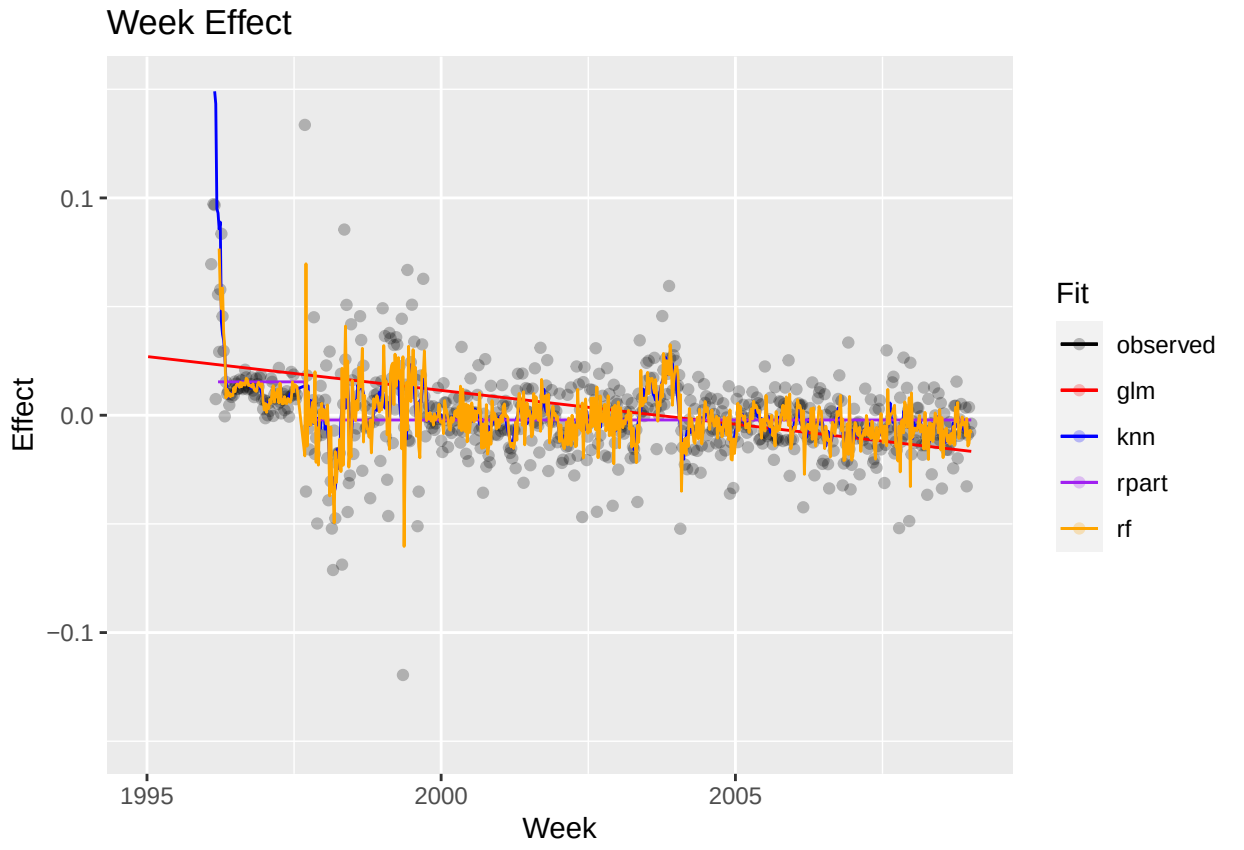
Let us start by teasing out any improvement on our model based on when a movie was rated and performing a time series analysis. We saw in exploration that most weeks have plenty of reviews, so we again group the data by the week of the review.

```
week_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(week) %>%
  summarize(week_num = as.numeric(week),
            b_d = mean(rating - mu_hat - b_i - b_u - b_g)) %>%
  distinct() %>%
  ungroup()
```

Because this is continuous data (time) rather than categorical, we expect that fitting to the data may be valuable to prevent over-fitting. Below is a collection of methods to smooth the time data, a linear model, k-nearest neighbors, regression trees, and random forest.

```
set.seed(1)
train_glm <- train(b_d ~ week_num,
                   method = "glm",
                   data = week_avgs)

train_knn <- train(b_d ~ week_num,
                     method = "knn",
                     data = week_avgs,
                     tuneGrid = data.frame(k = seq(1, 21, 2)))

train_rpart <- train(b_d ~ week_num,
                       method = "rpart",
                       tuneGrid = data.frame(cp = seq(0, 0.05, len = 25)),
                       data = week_avgs)

train_rf <- randomForest(b_d ~ week_num, data = week_avgs)

week_avgs <- week_avgs %>%
  mutate(fit_glm = predict(train_glm),
         fit_knn = predict(train_knn),
         fit_rpart = predict(train_rpart),
         fit_rf = predict(train_rf))

week_avgs %>%
  ggplot(aes(x = week)) +
  ylim(-0.15, 0.15) +
  geom_point(aes(y = b_d, color = "observed"), alpha = 0.25) +
  geom_line(aes(y = fit_glm, color = "glm")) +
  geom_line(aes(y = fit_knn, color = "knn")) +
  geom_line(aes(y = fit_rpart, color = "rpart")) +
```

```
  geom_line(aes(y = fit_rf, color = "rf")) +
  labs(title = "Week Effect",
       x = "Week",
       y = "Effect") +
  scale_colour_manual(name = "Fit",
      values = c(observed = "black", glm = "red", knn = "blue", rpart = "purple", rf = "orange"))
```




```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(week = round_date(date, unit = "week")) %>%
  left_join(week_avgs, by = "week") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_d,
         pred_fit_glm = mu_hat + b_i + b_u + b_g + fit_glm,
         pred_fit_knn = mu_hat + b_i + b_u + b_g + fit_knn,
         pred_fit_rpart= mu_hat + b_i + b_u + b_g + fit_rpart,
         pred_fit_rf = mu_hat + b_i + b_u + b_g + fit_glm)
week_results <-
  tibble(Method = c("Week", "GLM", "KNN", "RPart", "Random Forest"),
         RMSE = c(RMSE(pull(predicted_ratings, pred), test_set$rating),
                  RMSE(pull(predicted_ratings, pred_fit_glm), test_set$rating),
                  RMSE(pull(predicted_ratings, pred_fit_knn), test_set$rating),
                  RMSE(pull(predicted_ratings, pred_fit_rpart), test_set$rating),
                  RMSE(pull(predicted_ratings, pred_fit_rf), test_set$rating)))
knitr::kable(week_results)
```

Method	RMSE
Week	0.86549
GLM	0.86562
KNN	0.86556
RPart	0.86559
Random Forest	0.86562

```
results <- results %>%
  add_row(method = "Movie + User + Genre + Week Effect",
          RMSE = week_results %>% slice(1) %>% pull(RMSE))
```

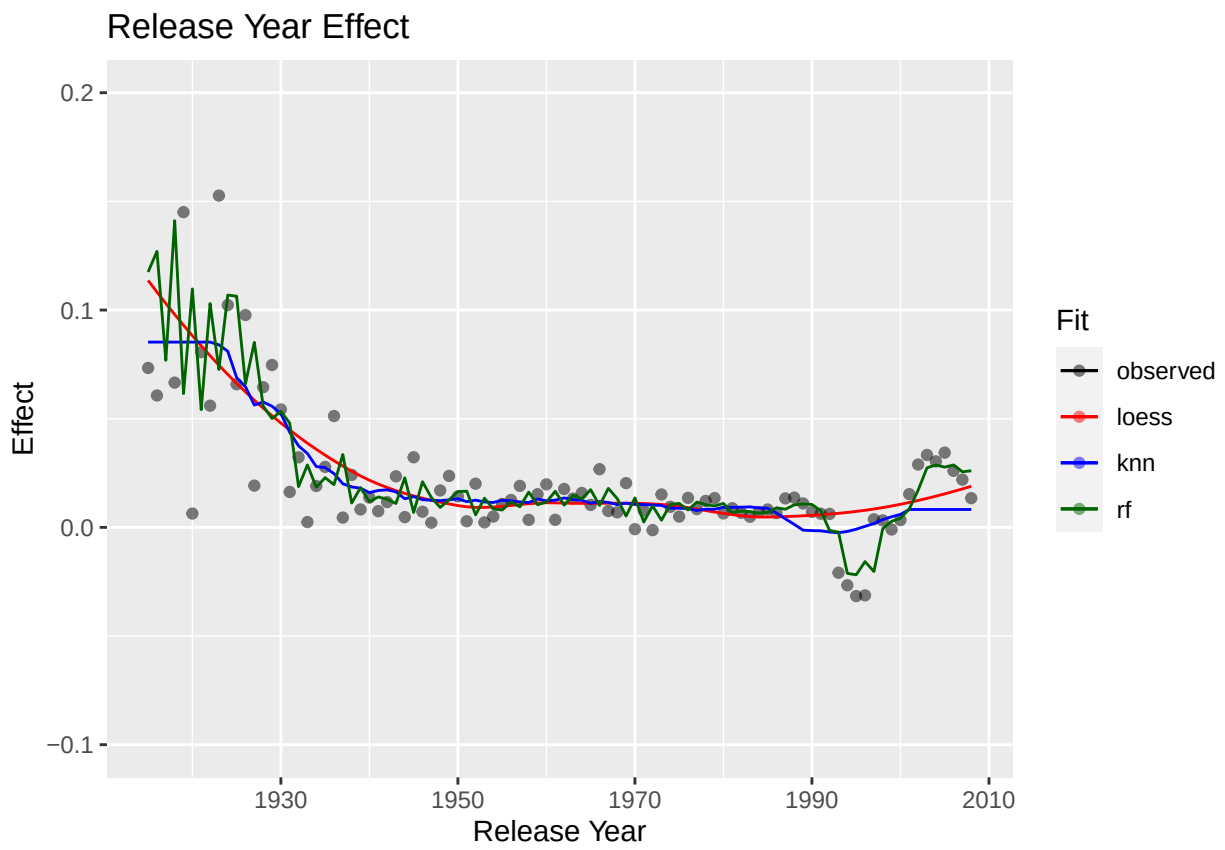
What a surprise! Incorporating a date of review effect seems to have almost no effect on the RMSE. This appears to differ from our intuition from the exploratory analysis. However, the relationship between the date of a rating and the average is likely folded into the user submitting ratings at that time.

Release Year

Similar to review date effect, we can check the effect of the year in which a movie was released. Again, we will sweep out other known effects before attempting to fit the line. Then, we will compare which fit results in the lowest RMSE against the test set.

```
year_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(week_avgs, by = "week") %>%
  group_by(release_year) %>%
  summarize(b_y = mean(rating - mu_hat - b_i - b_u - b_g - b_d))

set.seed(1)
train_loess = loess(year_avgs$b_y ~ as.numeric(year_avgs$release_year),
                              degree = 2)
train_knn = train(b_y ~ release_year,
                            method = "knn",
                            data = year_avgs,
                            tuneGrid = data.frame(k = seq(1, 21, 2)))
train_rf = randomForest(b_y ~ release_year, data = year_avgs)

year_avgs <- year_avgs %>%
  mutate(fit_loess = predict(train_loess),
         fit_knn = predict(train_knn),
         fit_rf = predict(train_rf))

year_avgs %>%
  ggplot(aes(x = release_year)) +
  ylim(-0.1, 0.2) +
  geom_point(aes(y = b_y, color = "observed"), alpha = 0.5) +
  geom_line(aes(y = fit_loess, color = "loess")) +
  geom_line(aes(y = fit_knn, color = "knn")) +
  geom_line(aes(y = fit_rf, color = "rf")) +
  labs(title = "Release Year Effect",
       x = "Release Year",
       y = "Effect") +
```

```
  scale_colour_manual(name = "Fit",
      values = c(observed = "black", loess = "red", knn = "blue", rf = "darkgreen"))
```



We see here that changing the method of smoothing greatly changes how the fit follows the data. Now lets see which method performs best with the test set.

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(week = round_date(date, unit = "week")) %>%
  left_join(week_avgs, by = "week") %>%
  left_join(year_avgs, by = "release_year") %>%
  mutate(other_effects = mu_hat + b_i + b_u + b_g + b_d,
         pred_year = other_effects + b_y,
         pred_loess = other_effects + fit_loess,
         pred_knn = other_effects + fit_knn,
         pred_rf = other_effects + fit_rf) %>%
  select(rating, pred_year, pred_loess, pred_knn, pred_rf)

year_results <- predicted_ratings %>%
  summarize(Year = RMSE(pred_year, rating),
            Loess = RMSE(pred_loess, rating),
            knn = RMSE(pred_knn, rating),
            rf = RMSE(pred_rf, rating))
knitr::kable(year_results)
```

Year	Loess	knn	rf
0.86525	0.86551	0.86544	0.86532

```
results <- results %>%
  add_row(method = "Movie + User + Genre + Week + Release Year Effect",
          RMSE = year_results$Year)
```

Though the Random Forest fit was close, it did not improve on treating the year as categorical rather than continuous. As a result, we will only use the effect based on the release year in the final model.

Years Between Release and Review

The relationship of time can be a tricky thing. Already, we have extracted the effect that the week in which a review was submitted effects its average rating. Just now, we extracted the effect of the year in which the movie was released on its average rating. But what of the effect of the relationship between the two?

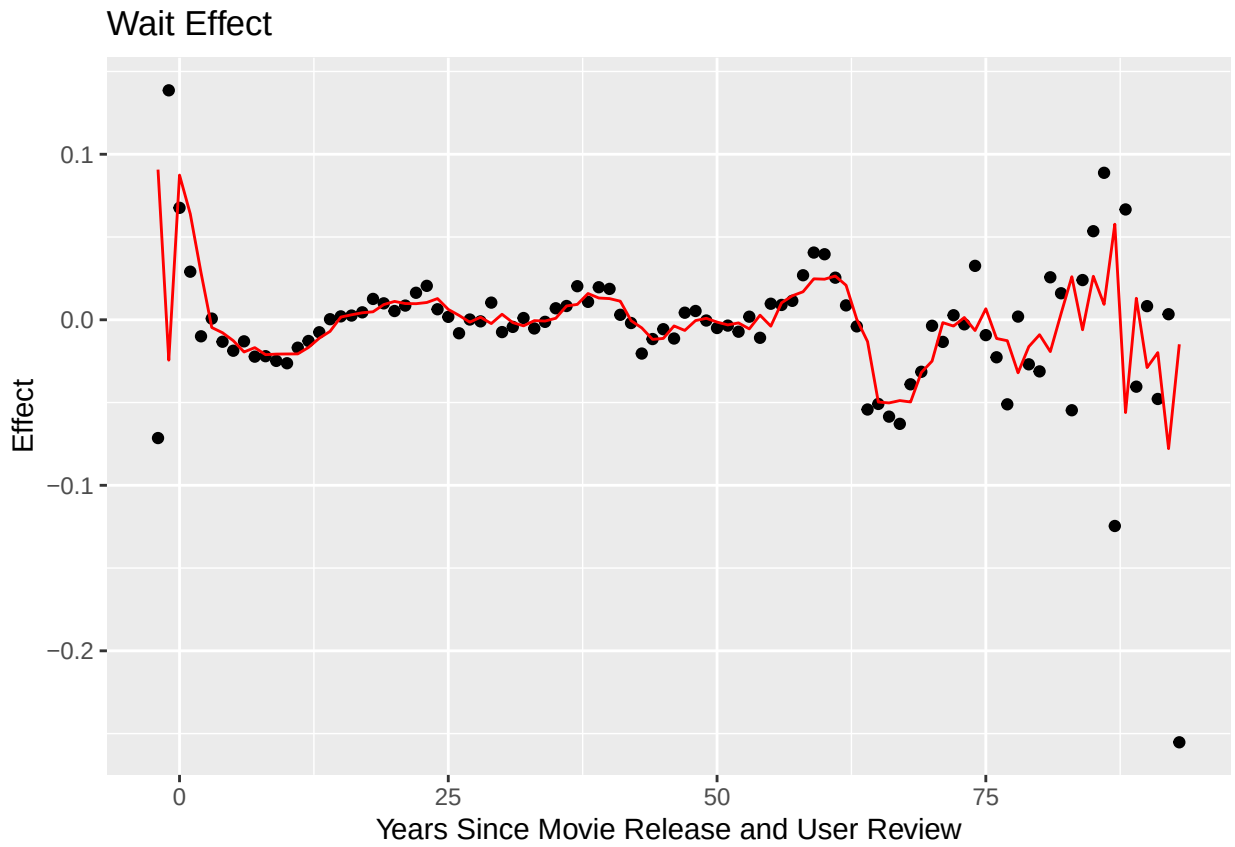
That is, what if we were to examine the effect of how long after a movie was released a user rated it on the average rating.

Recall that previously the years_since_release column was added to the edx data set, and it represents the number of years between the release year and the date of the review.

```
wait_avgs <-
  train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(week_avgs, by = "week") %>%
  left_join(year_avgs, by = "release_year") %>%
  group_by(years_since_release) %>%
  summarize(b_w = mean(rating - mu_hat - b_i - b_u - b_g - b_d - b_y))
```

Previously, the random forest method has been the best at producing a smoothed prediction, so only random forest will be used here to compare.

```
set.seed(1)
train_rf <- randomForest(b_w ~ years_since_release, data = wait_avgs)
wait_avgs <-
  wait_avgs %>%
  mutate(fit_rf = predict(train_rf))

wait_avgs %>%
  ggplot(aes(x = years_since_release)) +
  geom_point(aes(y = b_w)) +
  geom_line(aes(y = fit_rf), color = "red") +
  labs(title = "Wait Effect",
       x = "Years Since Movie Release and User Review",
       y = "Effect")
```




```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(week_avgs, by = "week") %>%
  left_join(year_avgs, by = "release_year") %>%
  left_join(wait_avgs, by = "years_since_release") %>%
  mutate(other_effects = mu_hat + b_i + b_u + b_g + b_d + b_y,
         pred = other_effects + b_w,
         pred_fit_rf = other_effects + fit_rf) %>%
  select(rating, pred, pred_fit_rf)

wait_results <- predicted_ratings %>%
  summarize(all_effects_rmse = RMSE(pred, rating),
            random_forrest_rmse = RMSE(pred_fit_rf, rating))
knitr::kable(wait_results)
```

all_effects_rmse	random_forrest_rmse
0.86498	0.86517

```
results <- results %>%
  add_row(method = "Movie + User + Genre + Date + Year + Wait Effect",
          RMSE = wait_results$all_effects_rmse)
```

Again, we find that attempting to smooth the data further does not improve the error when predicting the

test set.

It might be possible to go further with effects. However, we have run into serious diminishing returns. There are different tools we can now investigate to great effect.

Clamping and Half-Integer Rounding

At this point, we have gotten deeply into the weeds of the problem, trying to suss out any factors that could be having a further impact. But it is important to keep in mind that this is a real problem with boundaries. For example, users are only able to make ratings from 1 to 5 stars at half-integer intervals.

Below, we see the effect of merely clamping.

```
predicted_ratings <- predicted_ratings$pred
predicted_ratings[predicted_ratings > 5] <- 5
predicted_ratings[predicted_ratings < 1] <- 1

clamped_rmse <- RMSE(predicted_ratings, test_set$rating)
clamped_rmse
```

```
## [1] 0.86471
```

```
results <- results %>%
  add_row(method = "Clamping", RMSE = clamped_rmse)
```

Wow! An improvement. We have even surpassed the target RMSE. This is not surprising, as there were likely many predictions below 1 and above 5 that could not be "more right", only more wrong than they already were. In the context of a recommendation system, clamping does not make much sense. If we think a user would rate a movie higher than 5 (or lower than 1) if they could, then we are that much more confident in our recommendation. However, in the context of trying to reach a target RMSE, as in this problem, clamping to reduce error makes perfect sense.

Following this, does it make sense to round results to the nearest half-integer as well?

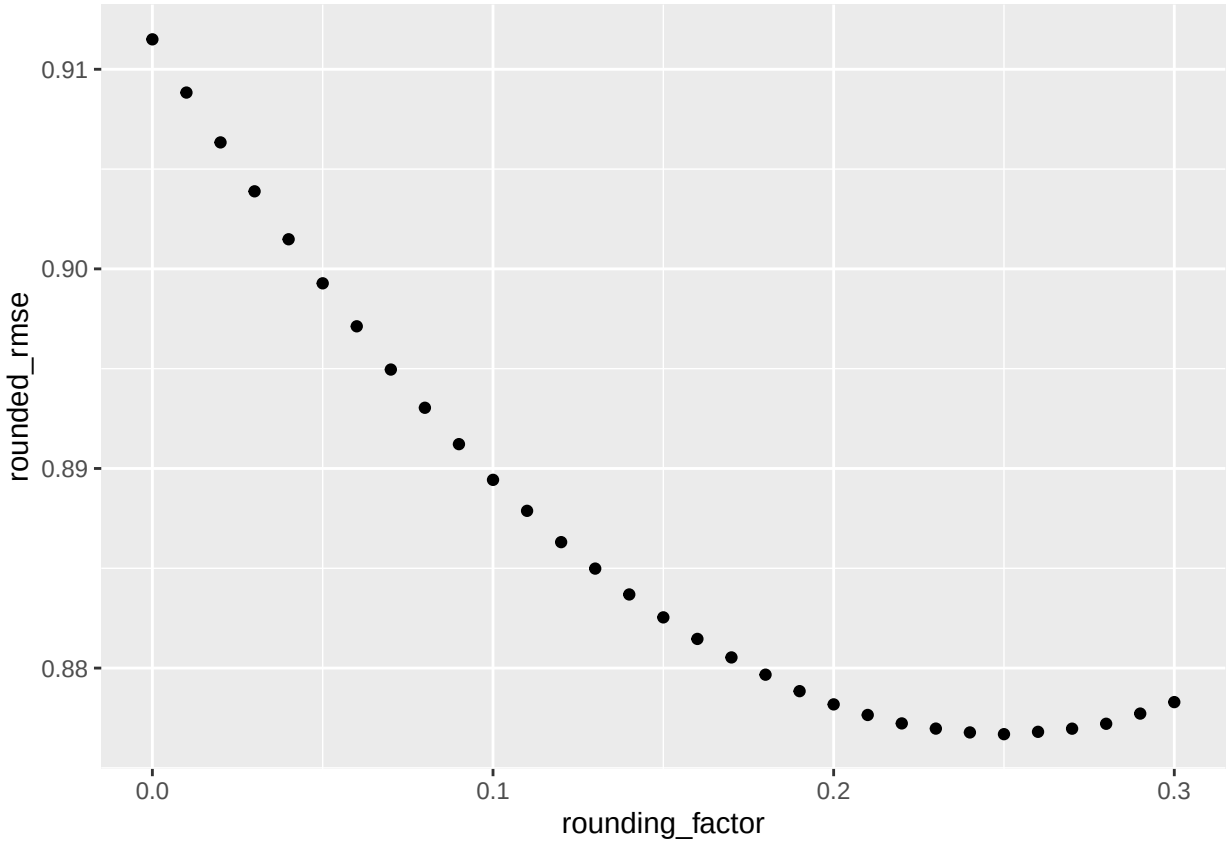
```
rounded_ratings <- 0.5 * round(2 * predicted_ratings)
rounded_rmse <- RMSE(rounded_ratings, test_set$rating)
rounded_rmse
```

```
## [1] 0.87669
```

```
results <- results %>%
  add_row(method = "Rounding", RMSE = rounded_rmse)
```

Oof, not the direction that we want to go. But wait! We saw in early exploratory analysis that users are far more likely to rate on whole-number values. Can we use a weighted rounding to eek out some improvement? Normally, rounding would occur at +/- 0.25. Lets see if there is any improvement as we get farther away.

```
rounding_factor <- seq(0, 0.3, 0.01)
rounded_rmse <- sapply(rounding_factor, function(r){
  base_prediction <- round(predicted_ratings)
  dec <- predicted_ratings - base_prediction
  base_prediction[dec > r] <- base_prediction[dec > r] + 0.5
  base_prediction[dec < -r] <- base_prediction[dec < - r] - 0.5
  return(RMSE(base_prediction, test_set$rating))
})
qplot(rounding_factor, rounded_rmse)
```




```
min(rounded_rmse)
```

```
## [1] 0.87669
```

```
rounding_factor[which.min(rounded_rmse)]
```

```
## [1] 0.25
```

We are already at the best as it can possibly be with basic rounding, and that significantly increases our error. Better to leave this out of the model.

Regularization

Nothing has yet been added to the model to account for movies and users with a low number of overall ratings. We saw in our data exploration that the numbers of ratings a movie received or a user submits can vary greatly.

We can correct the high effect of infrequently rated movies by adding a lambda value to our model in the denominator of the mean. That is, instead of dividing by the number of observations, we add a penalty term that will vanish for a high number of observations and force a lower effect based on fewer observations.

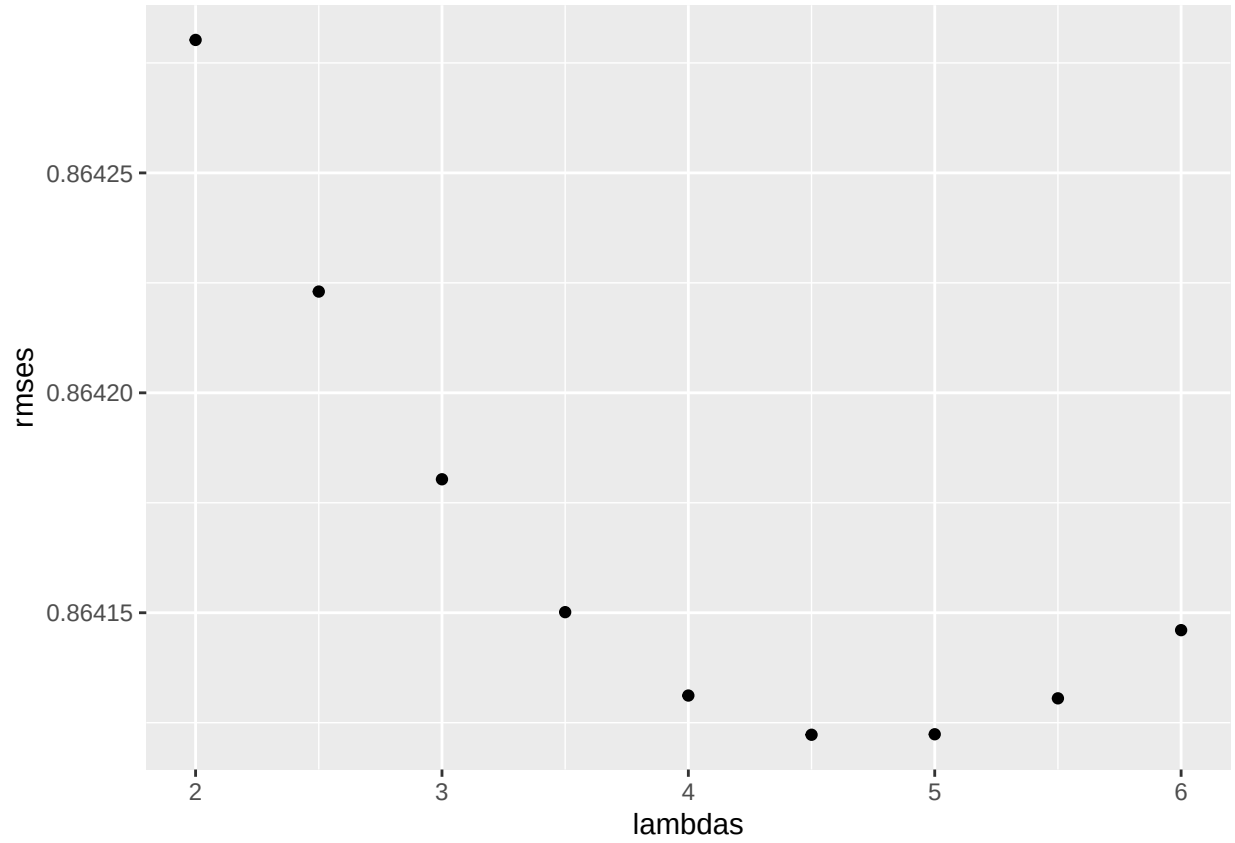
```
predict_ratings <- function(lambda, pred_against = test_set){
  # Movie effect
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat) / (n() + lambda))

  # User effect
  b_u <- train_set %>%
```

```
  left_join(b_i, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_hat) / (n() + lambda))

# Genre effect
b_g <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu_hat) / (n() + lambda))

# Date effect
b_d <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(week) %>%
  summarize(b_d = sum(rating - mu_hat - b_i - b_u - b_g) / (n() + lambda))

# Release Year Effect
b_y <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "week") %>%
  group_by(release_year) %>%
  summarize(b_y = sum(rating - mu_hat - b_i - b_u - b_g - b_d) / (n() + lambda))

# Years Since Release effect
b_w <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "week") %>%
  left_join(b_y, by = "release_year") %>%
  group_by(years_since_release) %>%
  summarize(b_w = sum(rating - mu_hat - b_i - b_u - b_g - b_d - b_y) / (n() + lambda))

# Combine all effects into a prediction for the set to predict against
predicted_ratings <-
  pred_against %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "week") %>%
  left_join(b_y, by = "release_year") %>%
  left_join(b_w, by = "years_since_release") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_d + b_y + b_w) %>%
  pull(pred)

# Clamping
predicted_ratings[predicted_ratings > 5] <- 5
predicted_ratings[predicted_ratings < 1] <- 1
```

```
  return(predicted_ratings)
}
```

Now, let us check a few values of lambda find where a minimum RMSE might be found.

```
lambdas <- seq(2, 6, .5)
predictions <- sapply(lambdas, predict_ratings)
rmses <- apply(predictions, 2, function(pred) {
  return (RMSE(test_set$rating, pred))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

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

```
min(rmses)
```

```
## [1] 0.86412
```

```
results <- results %>%
  add_row(method = "Regularized Effects", RMSE = min(rmses))
```

The RMSE bottoms out around 4.5. The minimum is 0.86412, well below the target RMSE. Overall, this is the improvement we have seen so far to the RMSE:

method	RMSE
Average	1.05990
Movie Effect	0.94374
Movie + User Effect	0.86593
Movie + User + Genres Effect	0.86559
Movie + User + Separated Genres Effect	0.86585
Movie + User + Genre + Week Effect	0.86549
Movie + User + Genre + Week + Release Year Effect	0.86525
Movie + User + Genre + Date + Year + Wait Effect	0.86498
Clamping	0.86471
Rounding	0.87669
Regularized Effects	0.86412

Over-training is a concern, because of the number of factors used and the use of cross-validation to tune regularization. Let us see how it checks out with the validation set that was set aside at the beginning of the analysis.

Validation

With the information we have so far, let us put together the final movie. We will include the movie effect, the user effect, the genre effect while treating all combinations of genre as distinct, the date of review effect rounded by weeks, the release year effect, and the difference between release and review effect in years. All of these effects will be regularized with the previously determined lambda (4.5), and the resulting predictions will be clamped between 1 and 5.

```
lambda <- 4.5
# Movie effect
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat) / (n() + lambda))

# User effect
b_u <- edx %>%
  left_join(b_i, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_hat) / (n() + lambda))

# Genre effect
b_g <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu_hat) / (n() + lambda))

# Date effect
b_d <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(week) %>%
  summarize(b_d = sum(rating - mu_hat - b_i - b_u - b_g) / (n() + lambda), n = n())

# Release Year effect
```

```
b_y <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "week") %>%
  group_by(release_year) %>%
  summarize(b_y = sum(rating - mu_hat - b_i - b_u - b_g - b_d) / (n() + lambda))

# Years Since Release Effect
b_w <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "week") %>%
  left_join(b_y, by = "release_year") %>%
  group_by(years_since_release) %>%
  summarize(b_w = sum(rating - mu_hat - b_i - b_u - b_g - b_d - b_y) / (n() + lambda))

predicted_ratings <-
  validation %>%
  mutate(release_year = str_extract(title, " \\(\\d{4}\\)$"),
         release_year = str_remove_all(release_year, " \\(|\\)"),
         release_year = as.integer(release_year),
         date = as_datetime(timestamp),
         week = as_date(round_date(date, unit = "week")),
         years_since_release = year(date) - release_year) %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "week") %>%
  left_join(b_y, by = "release_year") %>%
  left_join(b_w, by = "years_since_release") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_d + b_y + b_w) %>%
  pull(pred)

predicted_ratings[predicted_ratings > 5] <- 5
predicted_ratings[predicted_ratings < 1] <- 1

RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.86365
```

```
# 0.86365
```

We have flown below the target RMSE! Success!

Conclusion

In this report a data set from MovieLens containing 10 million movie ratings was downloaded and cleaned. Exploratory analysis was performed to evaluate possible effects to include in a predictive model. A multiple linear regression analysis was applied to develop the model. The model incorporated the movie effect, the user effect, the combined-genre effect, the date effect (rounded to weeks), the release date effect (rounded to years), and the effect from the time between release and review (rounded to year). The final model was able to predict unknown user ratings with a root mean squared error of less than 0.86490 and got as low as

0.86365 on a validation set.

Despite the success of the model, it appears that additional effects result in diminishing returns. It is possible that other high-impact effects have simply not yet been sussed out of the data. However, it is outside the scope of this analysis to perform an exhaustive exploration of alleffects.

Rather, it would be better to trust the data itself and perform principle component analysis in the form of single value decomposition (SVD). The flexible, math - based approach of SVD would results in a list of effects already ordered in their effectiveness. Enough effects could be applied to the model until over-fitting is observed.

However, the shear size of the 10 million reviews data set, with all its variable, requires a special approach to SVD. Examples of specialized SVD may be found online, in particular funkSVD.