HarvardX: PH125.9x: Movielens Recommendation System

Teri Duffie

3/30/2020

1.Introduction

Recommendation systems are used by companies to predict how well their customers will like a given item or service. They are built using massive datasets containing ratings provided by many users across many items. In the case of Netflix, the system predicts how many stars a specific user will give a specific movie. The star rating system ranges from 0.5-5 stars with 0.5 indicating extreme dislike and 5 suggesting the user will love it. If the algorithm predicts a high rating, the movie will be recommended to that user.

For the HarvardX: PH125.9x Data Science Capstone project, we will build a recommendation system in R using the Movielens 10M dataset provided by GroupLens research lab. We will apply many of the concepts and tools learned in the HarvardX Data Science Professional Certificate program. The goal of this project is to create a predictive model that minimizes the root mean squared error (RMSE) and accurately predicts whether a user will like or dislike a given movie.

Data

The data used for this project was collected and made available by GroupLens, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, and can be found here: https://grouplens.org/datasets/movielens/10m/

Details provided by GroupLens: MovieLens 10M movie ratings. Stable benchmark dataset. 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. Released 1/2009. Users were selected at random and all users had rated at least 20 movies. No demographic information is included. Each user is represented by an ID only.

The data are contained in three files: movies.dat, ratings.dat and tags.dat; however, code was provided to join these data into a single dataframe and split into an exploratory/training set and a validation set. This code is included in the Methods/Analysis section that follows.

Key steps

- Import/wrangle the data using the code provided and perform any cleaning necessary.
- Data exploration and visualization, noting insights and effects worth investigating in our model(s).
- The exploratory dataset (edx) is split into a training set and test set in order to evaluate and iterate our models.
- Once the final model is chosen we make predictions on the validation dataset and report the RMSE and accuracy.
- To conclude this report we summarize our findings and discuss limitations and potential improvements to the model.

2.Methods/Analysis

Download and initial review

Download MovieLens 10M data and create edx set and validation set using the code provided from HarvardX. Review the size and format of each set to ensure they were created as intended.

```r
#################################################################
# Create edx set, validation set with code provided from HarvardX
#################################################################

# Note: this process could take a couple of minutes

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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

```r
dim(edx)
```

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

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```r
dim(validation)
```

```
## [1] 999999      6
```

```r
head(validation)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                       title
## 1                                      Dumb & Dumber (1994)
## 2                                      Jurassic Park (1993)
## 3                                         Home Alone (1990)
## 4                                            Rob Roy (1995)
## 5                                     Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres
## 1                                 Comedy
## 2       Action|Adventure|Sci-Fi|Thriller
## 3                        Children|Comedy
## 4               Action|Drama|Romance|War
## 5                            Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

Both datasets are in tidy format and the edx data can easily be explored for outliers and missing information. We will pretend the validation set does not exist until we are ready to evaluate our final model.

Notice that “timestamp” is relatively meaningless to most people as it represents seconds since January 1, 1970. We’ll convert timestamp to date (Y-m-d) and add additional time intervals we will want to explore. Let’s also check the dataframe for NAs.

```
#add date/increments to edx
library(lubridate)
edx <- edx %>%
  transform(date = as.Date(as.POSIXlt(timestamp, origin = "1970-01-01",
    format ="%Y-%m-%d"), format = "%Y-%m-%d")) %>%
  mutate (year = format(as.Date(date), "%Y"),
          year_month = format(as.Date(date), "%Y-%m"),
          month = format(as.Date(date), "%m"),
          week = round_date(date, unit = "week"))

#check edx for missing values
any(is.na(edx))
```

```
## [1] FALSE
```

The quick summary shows that there are 10,677 unique movies and 69,878 unique users in the edx dataset. This means that not all users rate every movie as that would result in ~746 million rows and the edx dataset has only ~9 million.

We also note that the ratings were recorded between 1995-01-09 and 2009-01-05. The minimum rating was 0.5 and the maximum was 5, which is all in line with our expecations.

```
##     unique_movies      unique_users total_combination
##             10677             69878         746087406
```

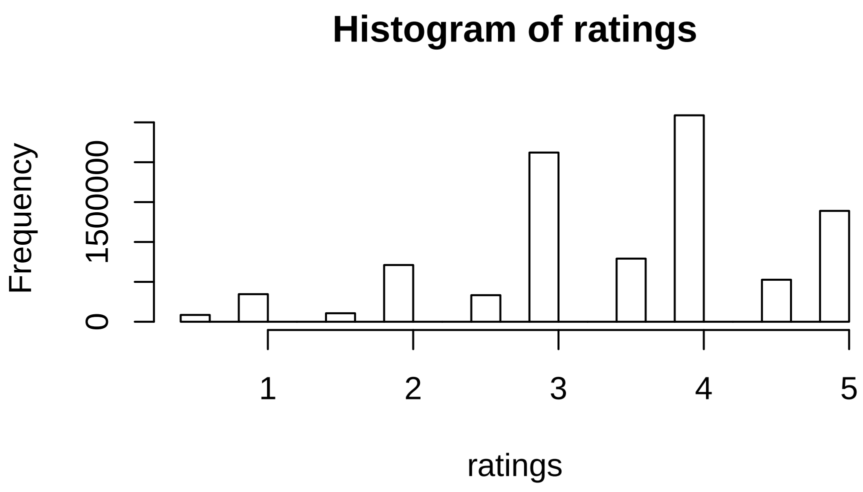
```
## min_rating max_rating
##        0.5        5.0
```

```
##     min_date     max_date
## "1995-01-09" "2009-01-05"
```

Exploration and statistics

Having confidence in our data, we can begin exploring.

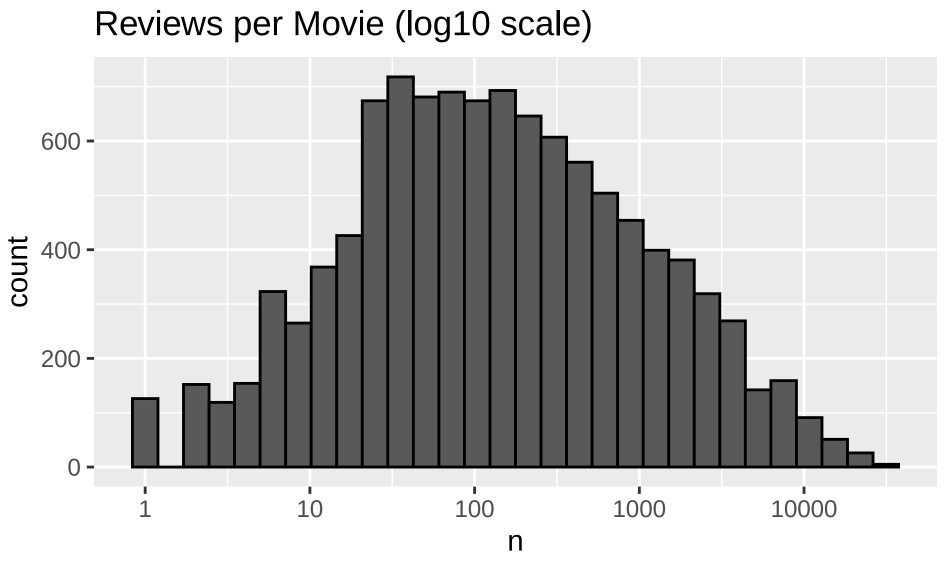
A histogram of all ratings shows users are able to give discreate 1/2 star ratings with 4 and 3 being the most common. Full star ratings are more common than 1/2 star. The overall mean rating of edx is 3.512.



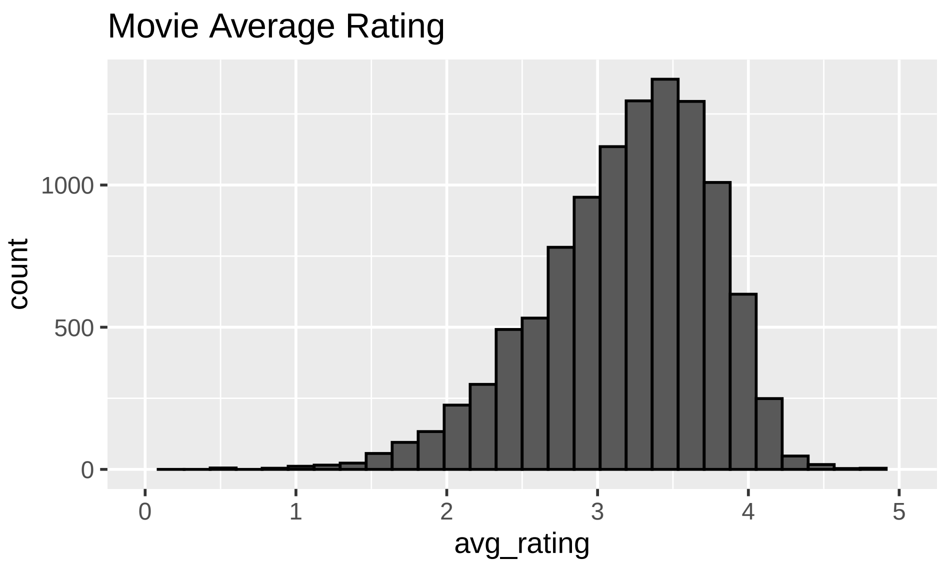
```r
mean(ratings)
```

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

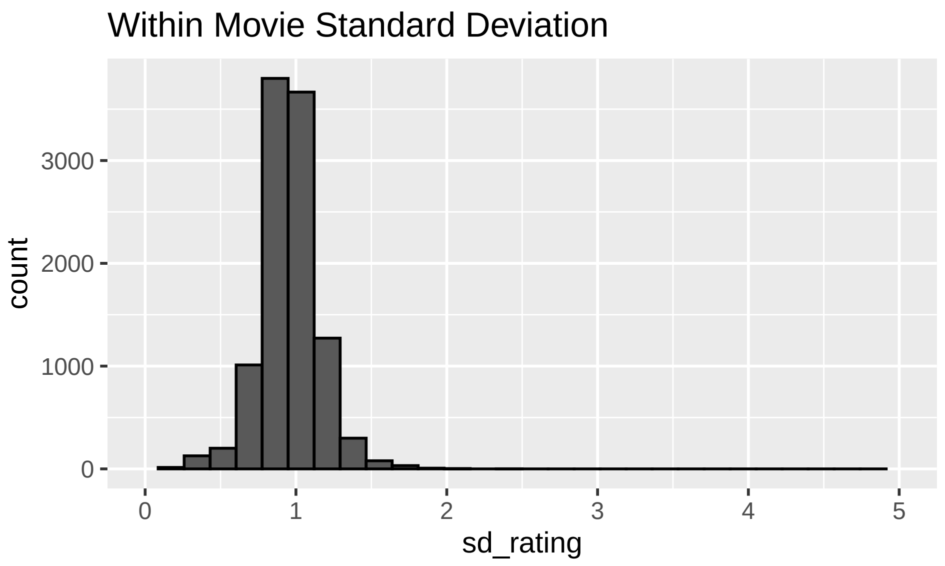
We can see that the number of ratings per movie has a very wide distribution with a range from 1 to over 10,000 and is skewed right. We note that this should be accounted for in our model as we have less confidence in ratings for movies that were rated only a few times.



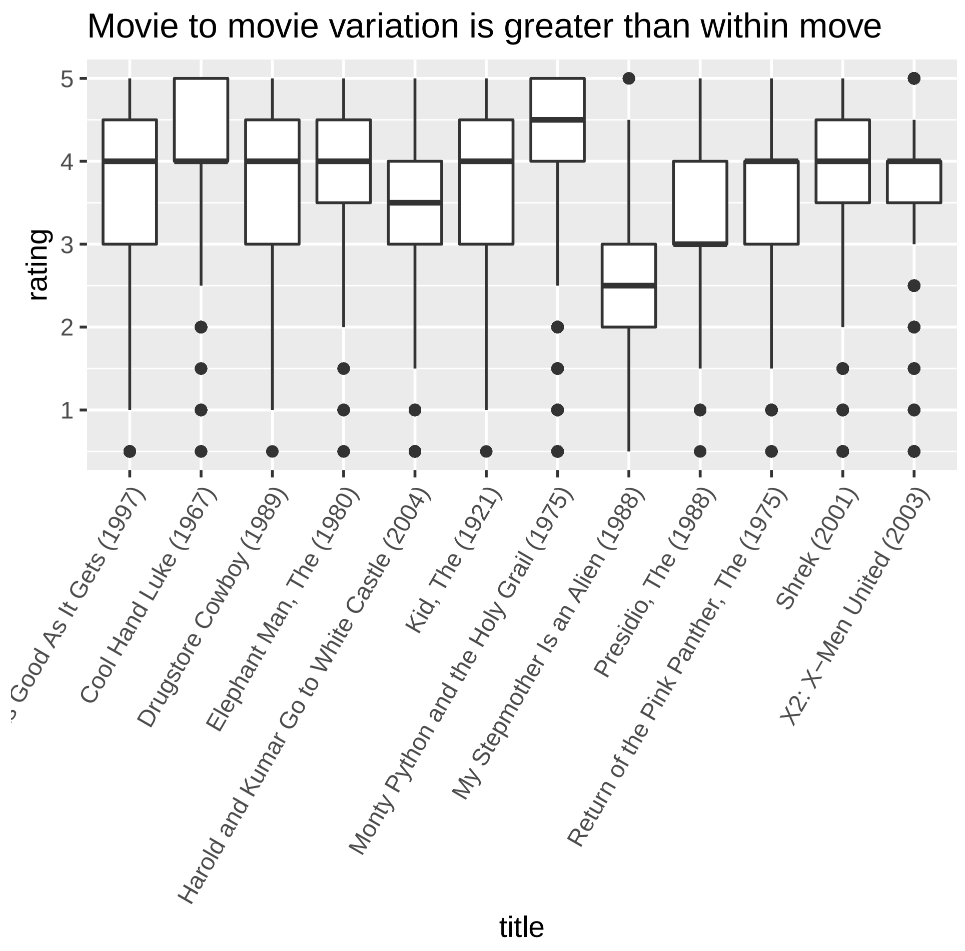
Now lets review the average rating by movie. We know intuitively that some movies are liked more than others and we can see that by examining the distribution of movie averages.



We should also note that the movie to movie variation (i.e. the spread of the distribution of averages above) is quite a bit greater than the spread of ratings within movie as seen below.

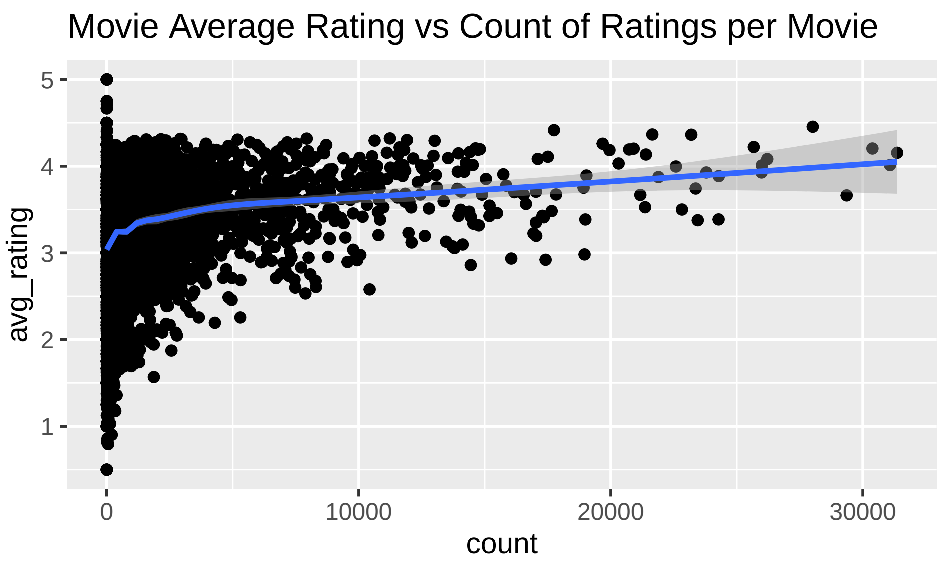


Here is a random sample of 12 movies which further illustrates this point:



The analysis above tells us with near certainty that the movie itself will be an important predictor of rating.

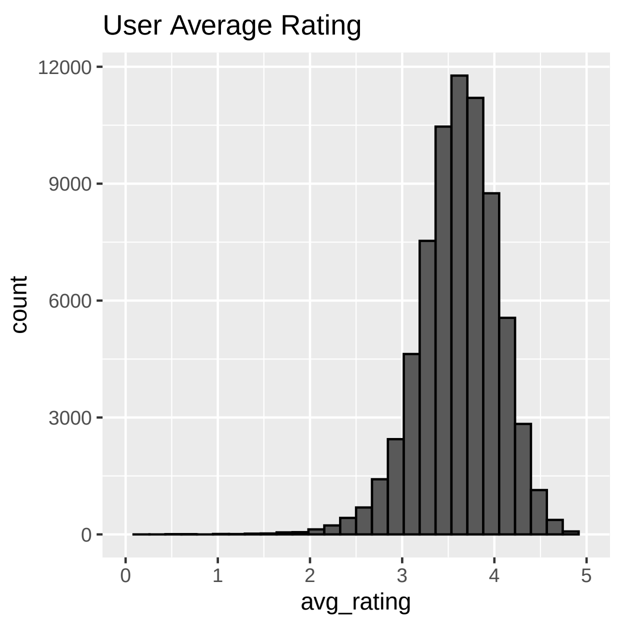
We can also see that movies with more ratings tend to be rated higher than those with fewer ratings. This makes sense as more people are likely to watch “better” movies. This is a trend we’ll want to revisit when building the model.

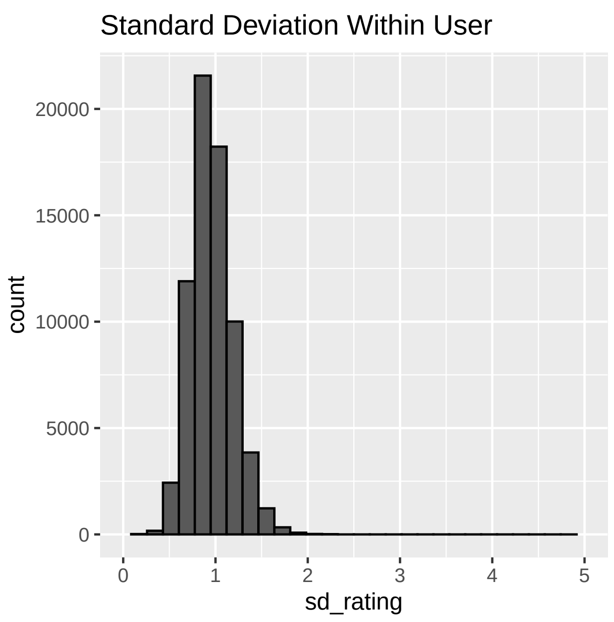


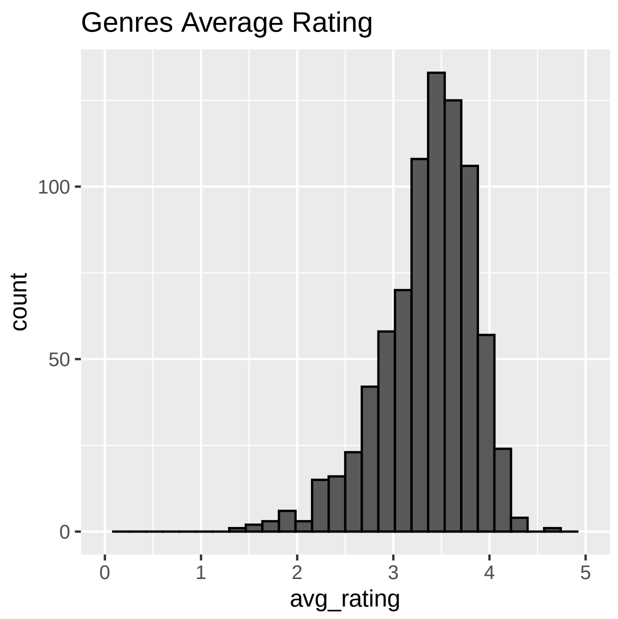
All but three of the top twenty most rated movies have an average rating above the overall average.

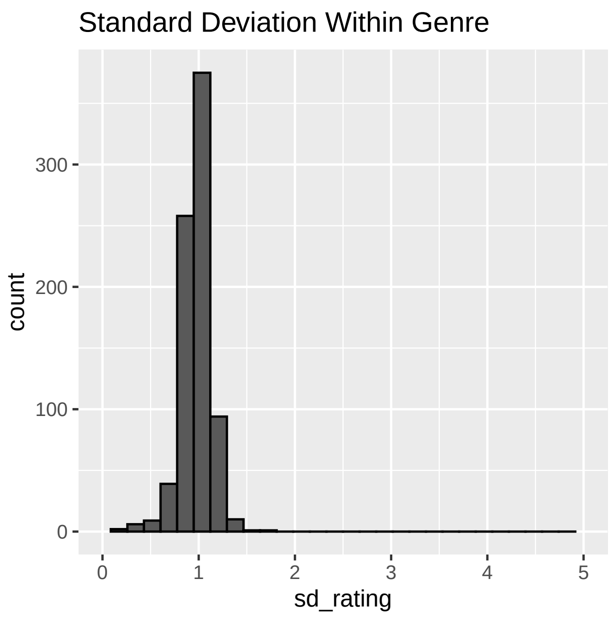
title	n	avg_rating
Pulp Fiction (1994)	31362	4.154789
Forrest Gump (1994)	31079	4.012822
Silence of the Lambs, The (1991)	30382	4.204101
Jurassic Park (1993)	29360	3.663522
Shawshank Redemption, The (1994)	28015	4.455131
Braveheart (1995)	26212	4.081852
Fugitive, The (1993)	25998	4.009155
Terminator 2: Judgment Day (1991)	25984	3.927859
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672	4.221311
Apollo 13 (1995)	24284	3.885789
Batman (1989)	24277	3.386292
Toy Story (1995)	23790	3.927638
Independence Day (a.k.a. ID4) (1996)	23449	3.376903
Dances with Wolves (1990)	23367	3.742628
Schindler's List (1993)	23193	4.363493
True Lies (1994)	22823	3.500285
Star Wars: Episode VI - Return of the Jedi (1983)	22584	3.996436
12 Monkeys (Twelve Monkeys) (1995)	21891	3.874743
Usual Suspects, The (1995)	21648	4.365854
Fargo (1996)	21395	4.134821

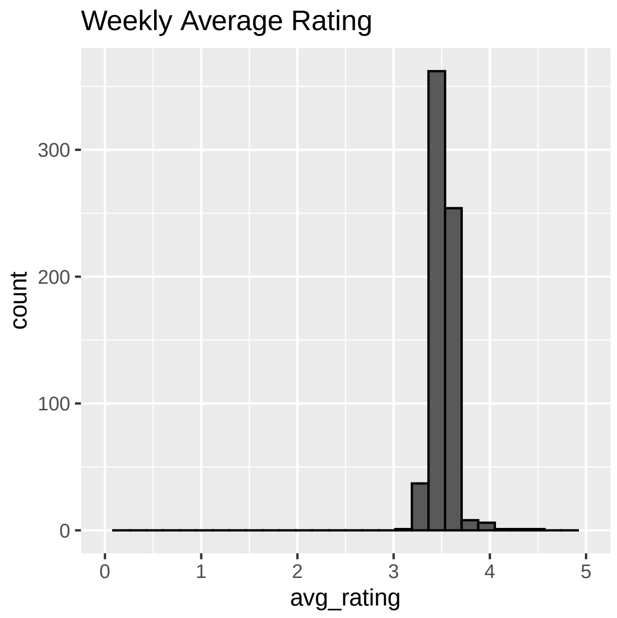
Now let's see how factors such as users, genres and week of review could explain varibility in ratings. For now we analyze review timeframe by week but we will examine the time factor in further detail later on. Similar to movies, we are also curious as to whether variation we see group to group, let's say genres, is greater than the variation we see within each genre. Again, we are looking at how wide the averages-by-group-distribution is compared to the distribution of standard deviations within each group. This understanding helps us to incorporate only meaningful features in the model.

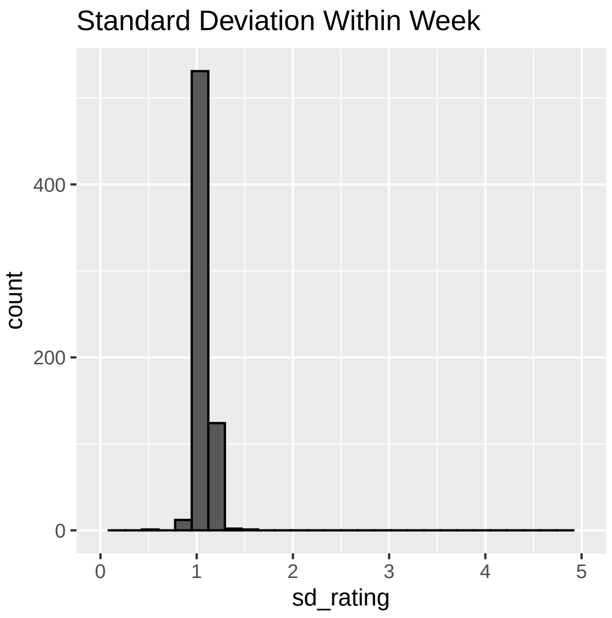

User Average Rating
count
12000
9000
6000
3000
0
0
1
2
3
4
5
avg_rating


Standard Deviation Within User
count
20000
15000
10000
5000
0
0
1
2
3
4
5
sd_rating


Genres Average Rating
count
100
50
0
0
1
2
3
4
5
avg_rating

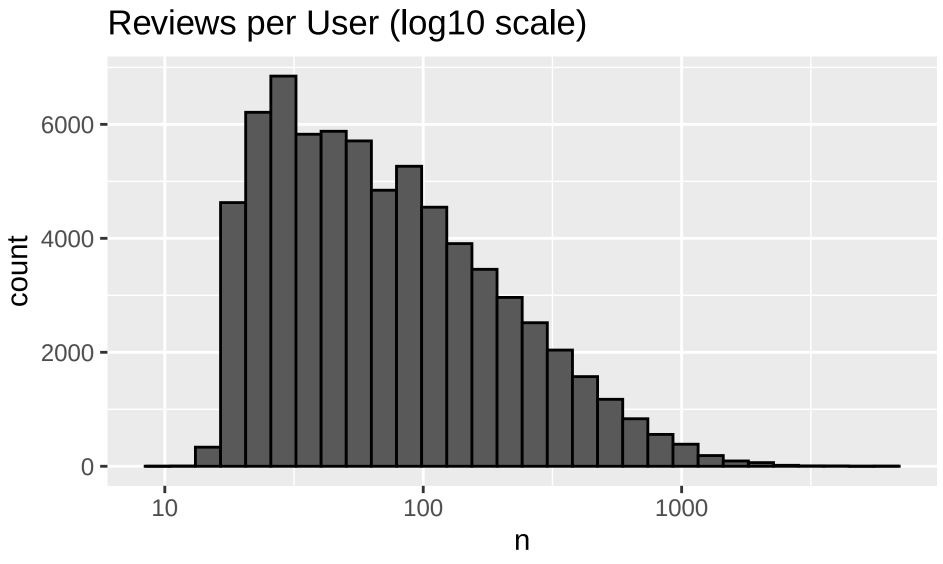

Standard Deviation Within Genre
count
300
200
100
0
0
1
2
3
4
5
sd_rating





Based on the plots above we expect that movie, user and genre will be important features in our model. We might still consider including a time feature but need to analyze this further as week to week average ratings explain much less variation than the other three factors and we want to be mindful of overfitting. We can also see that the spread of the ratings within a week are nearly as large as the spread of the week to week averages.

Let's review the number of ratings per user. Here we can see that this also varies greatly and we make a note that we'll want to account for this in our model.



Like movies and users, the summary below shows that the number of ratings per genre(s) is also widely distributed.

```
##     genres              count
##  Length:797         Min.   :     2
##  Class :character   1st Qu.:   185
##  Mode  :character   Median :  1459
##                     Mean   : 11292
##                     3rd Qu.:  7167
##                     Max.   :733296
```

Note that the genres column often inclues more than one genre separated by "|".

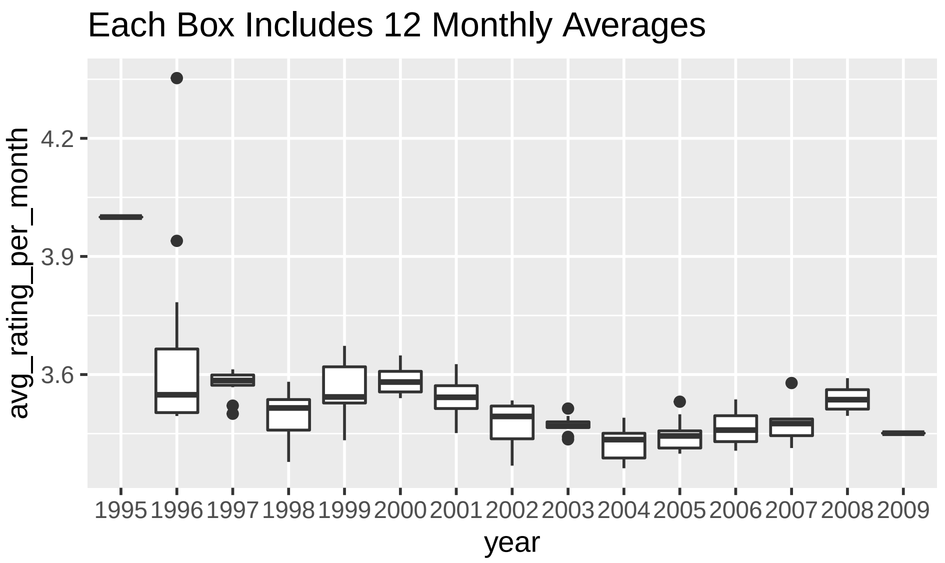
```
## [1] "Comedy|Romance"                "Action|Crime|Thriller"
## [3] "Action|Drama|Sci-Fi|Thriller"  "Action|Adventure|Sci-Fi"
## [5] "Action|Adventure|Drama|Sci-Fi" "Children|Comedy|Fantasy"
```

We will leave genres grouped as we prefer each line in our dataframe to represent one sample (rating) when we build our model. It is also suspected that the grouped genres have meaning. For instance, someone may like comedies very much, but not really be into romantic comedies. Or they could love action movies but hate Sci-Fi.Therefore splitting these apart and assigning the same ranking to each genre does not make sense intuitively. I will describe another possible approach when discussing potential improvements to the model in the Conclusion section. For now, we are satisfied that most grouped genres have enough ratings to be considered useful and we will use regularization when we build our model to account for small sample sizes. More details on regularization to come.

Now we're going to examine the timeframe factor in a few ways.

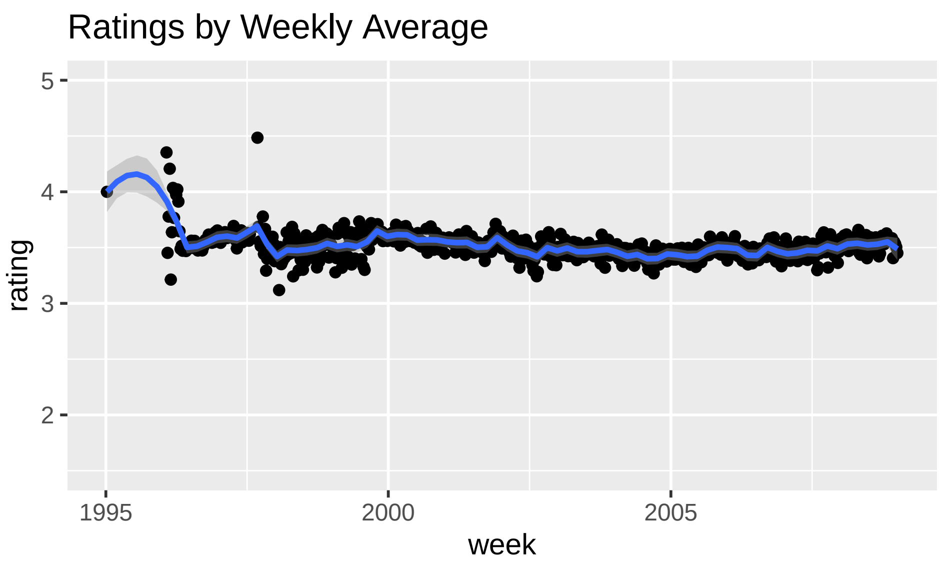
First let's review the year over year ratings. Instead of visualizing only the yearly average, let's make box plots using the average for each month of the year. This allows us to review the variability within each year as well as the general year to year trend.

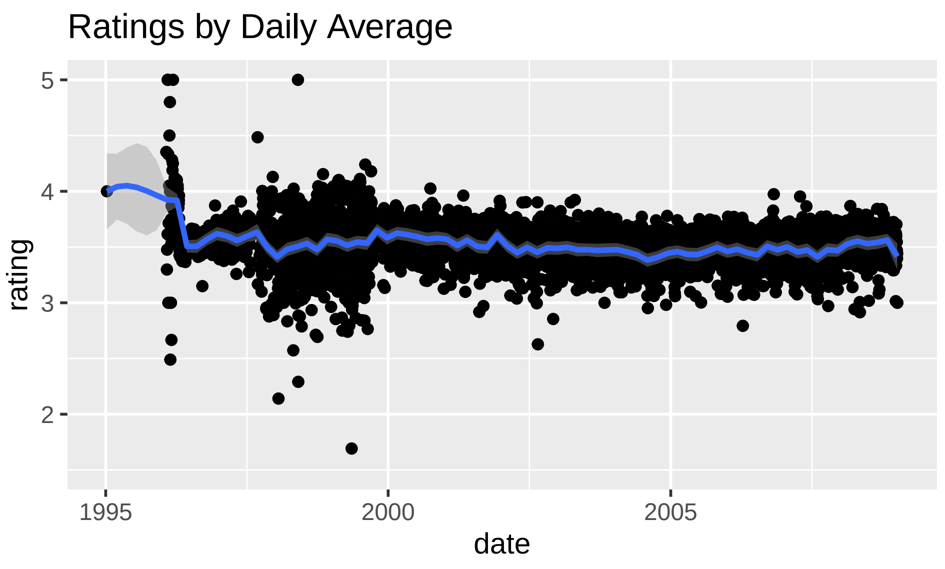
We can see that there is some variation in ratings explained by year, but most of the monthly averages in most years don't stray too far from the overall average rating of 3.51. The length of the boxes tends to be greater in the earlier years of rankings which points to more variance in those years. There are only two data points in 1995 and 1996 has the most variance of any year.



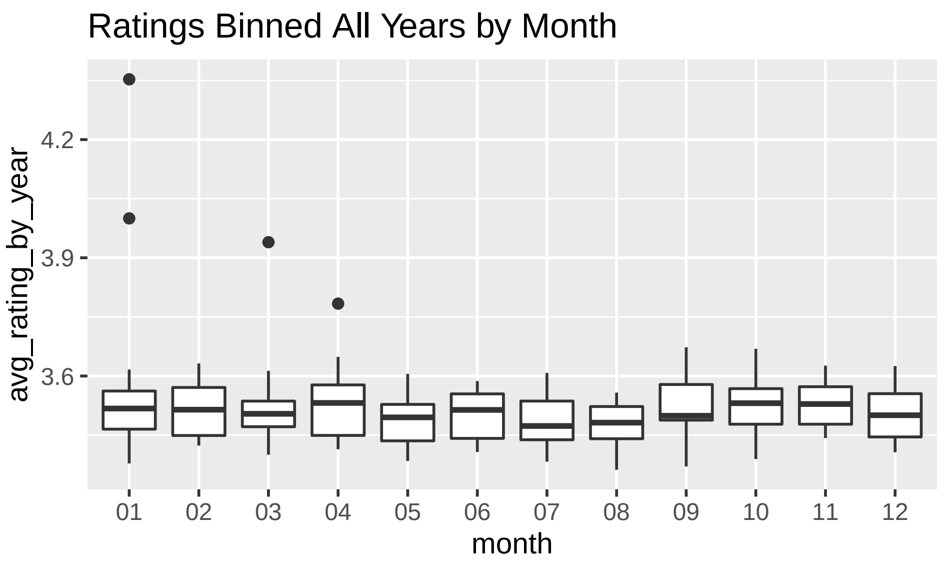
```
## The number of ratings in 1995 was 2
```

The plots by daily and weekly average tell the same basic story: High variation in the earlier years followed by less variation beyond the start of 2000. The smooth fit shows that after 1996, the average by week does not fluctuate far from the overall mean rating of 3.51, and the fit is not great prior to 2000 due to the high variation in ratings. We therefore decide not to complicate our model by adding a time factor that complicates the model and does not explain the ratings in a meaningful way.

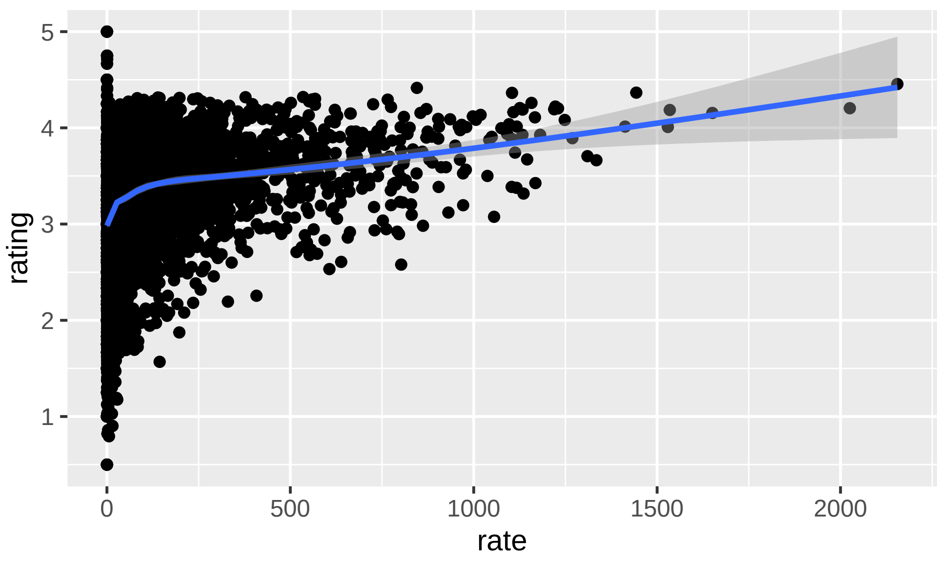




Note that there isn't a monthly effect. We also wanted to investigate a potential seasonal impact, though without geographic information this is not possible.



We observe a relationship between the number of times per year a movie gets rated (rate) and the average rating. This is very similar to the relationship we observed previously regarding the total number of reviews per movie vs rating.



Let's determine later if we should incorporate rate into our model after the more relavent factors of movie, user and genre have been accounted for.

Modeling approach

A penalized least squares approach is chosen given our findings during data exploration and the size of the dataset which will make most other approaches difficult. We will show that we are able to achieve low RMSE and reasonable accuracy using our approach.

We split the exploratory dataset (edx) into a training set and test set to evaluate and iterate our models. The training set is used to develop the model and we'll generate predictions on the test set using the model. We'll then report the RMSE of the predicted ratings vs. the true ratings.

```r
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

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

It is assumed our model can be defined by the following equation, though we will evaluate the RMSE on the test set with the addition of each feature to ensure we aren't overfitting. We will also analyze the residuals of our fitted model against the test set to determine if a relationship still exists with one of our time-based factors and should be added to the model.

$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

where b_i, b_u and b_g represent the movie, user and genres effect with genres remaining grouped.

The goal of this project is to mimimize the root mean squared error (RMSE) of the predicted ratings vs actual ratings. RMSE is defined by the equation:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

In R we create a function:

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

Once we choose the final model utilizing the edx train and test sets, we'll evaluate the RMSE of the validation set and check the accuracy by determining the rate at which it can correctly predict whether a user will enjoy a movie better than average.

3.Results

The simplest possible model is to predict the same rating for all movies regardless of any other factor. The rating that minimizes the error is the overall average, μ.

```r
#predict the average
mu <- mean(train_set$rating)
mu
```

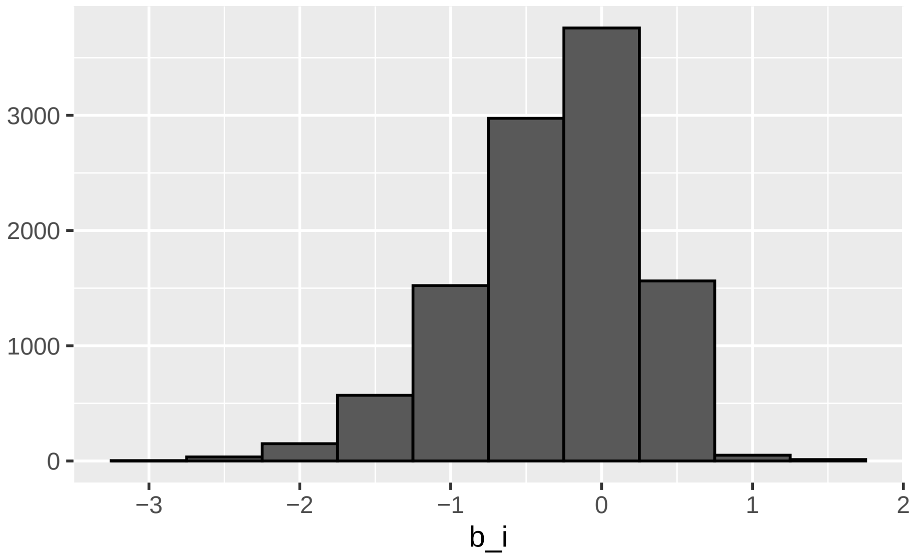
```
## [1] 3.512482
```

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

rmse_results <- data_frame(Method = "Train/Test: Just the average", RMSE = naive_rmse)

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

Method	RMSE
Train/Test: Just the average	1.059904

However, we know there is a movie effect as we saw this during data exploration. The movie effect b_i, can be defined as the average of residuals, grouped by movie, obtained when we use "Just the Average" model to predict ratings. You can see b_i and calculation code below. Note that additional effects will be calculated similarly from the residuals of the previous iteration of the model.

```r
#calculate and show distribution for movie effect b_i
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

b_i %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



We'll add the movie effect to the model and observe the improved RMSE.

```r
predicted_ratings <- mu + test_set %>%
  left_join(b_i, by='movieId') %>%
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Train/Test: Movie Effect Model",
                                     RMSE = model_1_rmse ))

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

Method	RMSE
Train/Test: Just the average	1.0599043
Train/Test: Movie Effect Model	0.9437429

During data exploration, we also noted that some users love all movies, some hate all moves, but most rate on average between 3-4 stars. Here is the summary of average rating by user as well as the RMSE obtained when we add the user effect to our model.

```r
#User distribution summary.  Note 1st-3rd quartile is 3.35-3.91
user_summary <- train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating))
summary(user_summary)
```

```
##      userId           b_u
##  Min.   :    1   Min.   :0.500
##  1st Qu.:17943   1st Qu.:3.353
##  Median :35799   Median :3.634
##  Mean   :35782   Mean   :3.614
##  3rd Qu.:53620   3rd Qu.:3.905
##  Max.   :71567   Max.   :5.000
```

```r
#add_user effect to model
b_u <- train_set %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Train/Test: Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Train/Test: Just the average	1.0599043
Train/Test: Movie Effect Model	0.9437429
Train/Test: Movie + User Effects Model	0.8659320

You can see that we have again improved the RMSE. Let's now add the genres effect which we also noted contributes to the ratings variation.

```r
#add genre effect to model
b_g <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
```

```r
predicted_ratings <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Train/Test: Movie + User+ Genre Effects Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Train/Test: Just the average	1.0599043
Train/Test: Movie Effect Model	0.9437429
Train/Test: Movie + User Effects Model	0.8659320
Train/Test: Movie + User+ Genre Effects Model	0.8655941

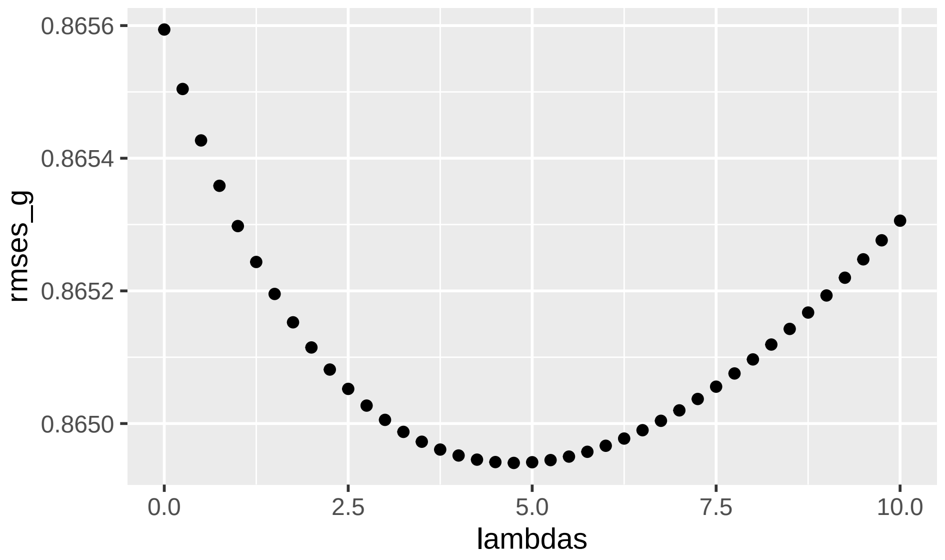
The model is slightly improved, but we should recall some additional observations we made during data exploration. We noted that some movies have not been rated very many times, some users have not given many ratings and some genres have fairly low counts. Because we have less confidence in the estimates that come from small sample sizes we need to use regularization to penalize large estimates of b_i, b_u and b_g coming from small sample sizes.

We can use cross validation to find the value of lambda that minimizes the RMSE using our new model equation. The value of the effects, say b_i, that minimize the equation are now:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

where n_i is the number of ratings b for movie i. Large sizes of n_i will result in the demoninator close to n_i thus the lamba will have less impact on the estimate.

```r
#Regularization with movie, user, genre

lambdas <- seq(0, 10, 0.25)
rmses_g <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses_g)
```



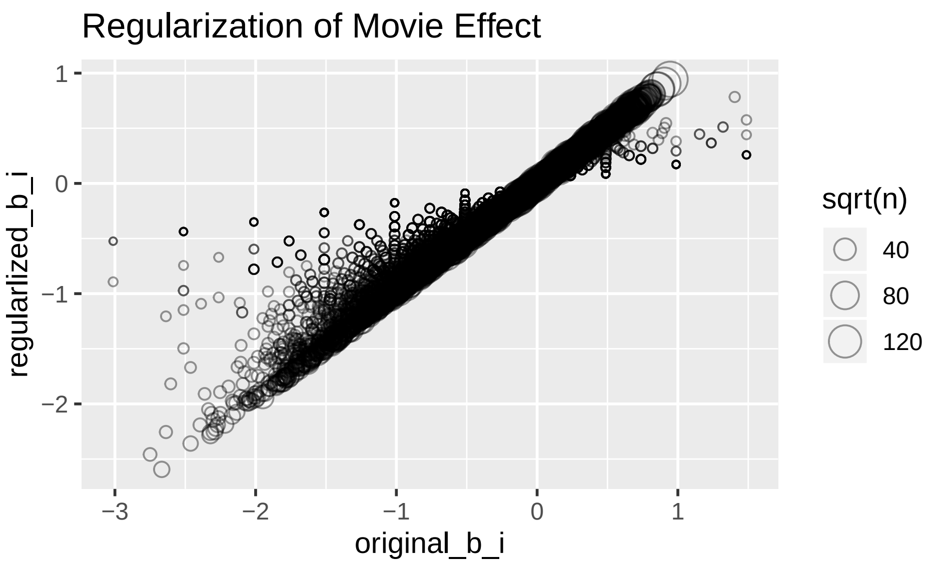
```
lambda_i_u_g <- lambdas[which.min(rmses_g)]

#the value of lambda that minimizes the RMSE
print(lambda_i_u_g)
```

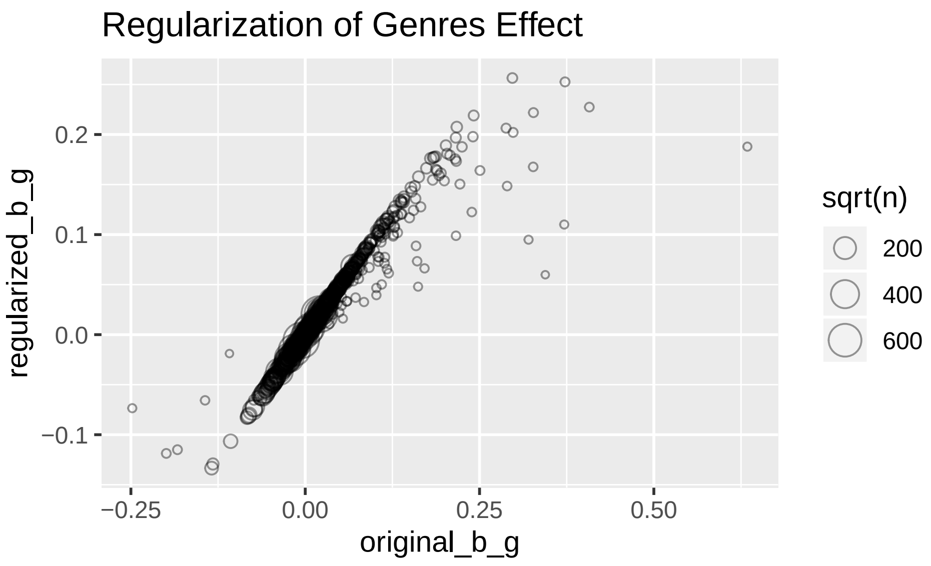
```
## [1] 4.75
```

And we observe improvement in the RMSE using the regularized model.

Method	RMSE
Train/Test: Just the average	1.0599043
Train/Test: Movie Effect Model	0.9437429
Train/Test: Movie + User Effects Model	0.8659320
Train/Test: Movie + User+ Genre Effects Model	0.8655941
Train/Test: Regularized Movie + User + Genre Effect Model	0.8649406

We can visualize the impact that regularization had on the effects. Here we see that large estimates with small sample sizes shirnk towards zero when regularized for both the movie and genres effect:

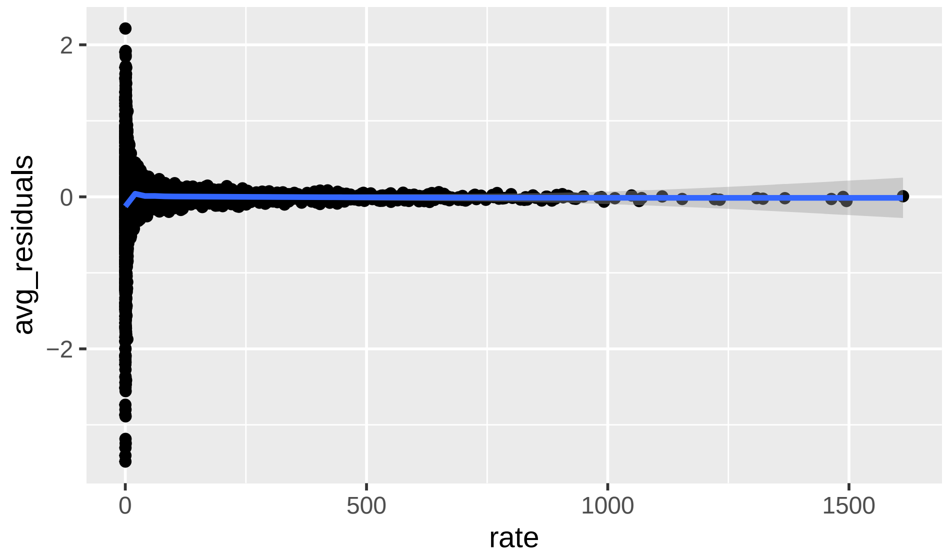




Now let's re-evaluate rate as a factor. Remember that rate is the average number of ratings a movie gets per year. When we apply a smooth fit of rate vs the residuals (using our regularized movie+user+genre effect model), we no longer see a relationship worth incorporating into our model.

```
#check for effect of rate once other effects accounted for
#first build joined_train table and compute residuals

movie_reg <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating-mu)/(n() + 1))

user_reg <-  train_set%>%
  left_join(movie_reg) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating-mu-b_i)/(n() + 1))
```

```
genre_reg <-  train_set%>%
  left_join(movie_reg) %>%
  left_join(user_reg) %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating-mu-b_i-b_u)/(n() + l))


#define rate
train_set_rate <-train_set %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2018 - first(as.numeric(year)),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  select(movieId, rate)

#join effects to test set and calculate residuals
joined_test <- test_set %>%
  left_join(movie_reg, by='movieId') %>%
  left_join(user_reg, by='userId') %>%
  left_join(genre_reg, by='genres') %>%
  left_join(train_set_rate, by="movieId") %>%
  replace_na(list(b_i=0, b_u=0, b_g=0)) %>%
  mutate(residuals = rating - mu - b_i - b_u - b_g)
```



For our final model we choose the Regularized Movie + User + Genres Effect Model and evaluate the RMSE of the predictions on the validation set.

```
#final evaluation: use edx to predict validation set ratings

mu <- mean(edx$rating)
lambda <- 4.75
```

```r
movie_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating-mu)/(n() + lambda))

user_reg <-  edx %>%
  left_join(movie_reg) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating-mu-b_i)/(n() + lambda))

genre_reg <-  edx %>%
  left_join(movie_reg) %>%
  left_join(user_reg) %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating-mu-b_i-b_u)/(n() + lambda))

joined_table <- validation %>%
  left_join(movie_reg, by='movieId') %>%
  left_join(user_reg, by='userId') %>%
  left_join(genre_reg, by='genres') %>%
  replace_na(list(b_i=0, b_u=0, b_g=0)) %>%
  mutate(residuals = rating - mu - b_i - b_u - b_g, predicted = mu + b_i + b_u + b_g)


validation_rmse <- RMSE(joined_table$predicted, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Validation Final: Regularized Movie + User + Genre Effect M
                                     RMSE = validation_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Train/Test: Just the average	1.0599043
Train/Test: Movie Effect Model	0.9437429
Train/Test: Movie + User Effects Model	0.8659320
Train/Test: Movie + User+ Genre Effects Model	0.8655941
Train/Test: Regularized Movie + User + Genre Effect Model	0.8649406
Validation Final: Regularized Movie + User + Genre Effect Model	0.8644514

The validation RMSE is 0.8644514.

We also wanted to determine how accurate we are at predicting whether a user will like a movie better than average. We find that we can accurately predict this 71.6% of the time with the sensitivity slightly higher and specificity slightly lower (with “thumbs up” as positive class).

```r
predicted_recommend <- ifelse(joined_table$predicted >= mean(edx$rating), "thumbs up", "thumbs down")
actual <- ifelse(validation$rating >= mean(validation$rating), "thumbs up", "thumbs down")

confusionMatrix(as.factor(predicted_recommend), as.factor(actual), positive = "thumbs up")
```

```
## Confusion Matrix and Statistics
##
```

```
##              Reference
## Prediction    thumbs down thumbs up
##   thumbs down      340646    124998
##   thumbs up        158949    375406
##
##                Accuracy : 0.7161
##                  95% CI : (0.7152, 0.7169)
##     No Information Rate : 0.5004
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4321
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7502
##             Specificity : 0.6818
##          Pos Pred Value : 0.7025
##          Neg Pred Value : 0.7316
##              Prevalence : 0.5004
##          Detection Rate : 0.3754
##    Detection Prevalence : 0.5344
##       Balanced Accuracy : 0.7160
##
##        'Positive' Class : thumbs up
##
```

4.Conclusion

I am satisfied with the validation RMSE of 0.8644514 and accurately predicting a "thumbs up" vs "thumbs down" almost 72% of the time.

I noted previously that there was another approach to deal with the genres factor. One could have filtered the train set to include only movies with a single genre and averaged each genre to obtain a b__g. Then a single b__g for the movies with multiple genres listed could have been calculated by summing the individual genre b__g's. Intuitively I wouldn't expect that an estimate obtained with this method would be better than leaving the genres grouped. We saw that after movie and user were accounted for, the estimates for b__g were quite small.

I believe that matrix factorization would have likely improved the model, possibly significantly, as we expect some interactions between movies, users and genres to be present; however, I don't feel totally conifident in my understanding of this approach so I chose to limit the model to concepts I can explain clearly.

I tried to utilize some additional machine learning algorithms, such as kNN, but I found pretty quickly that RAM was going to be a limitation with this type of approach.

Overall I enjoyed the challenge and the chance to bring together several of the tools and concepts I learned in the HarvardX Data Science program.