PH125.9x Data Science: Capstone - MovieLens

Nathan Fischer

9/8/2020

Introduction

A recommendation system is a software algorithm that uses historical user ratings for items to make predictions for future ratings. Using these predicted ratings, companies can provide their users with recommended items that have a high predicted rating for the user. For companies such as Amazon, Netflix, or eBay, which have a large catalog of items, a large number of user ratings is also needed to develop a recommendation system with meaningful accuracy. Due to the large datasets involved, the standard statistical methods for regression (i.e. linear, non-linear) are not feasible on standard computing hardware. This makes these types of systems a good learning opportunity for the methods used in data science.

The MovieLens dataset contains user ratings for movies, similar to the Netflix rating system of 1 to 5 stars. It is provided by GroupLens research center at the University of Minnesota. In this study, the dataset with 10 million ratings will be used to develop a recommendation system.

Using the groundwork developed in the PH125.8x Data Science – Machine Learning course as a starting point, this study will use exploratory data analysis techniques to gain insight into the structure and nature of this data and then develop a model to accurately predict user ratings. In order to assure general applicability of our model and prevent overfitting, the MovieLens dataset is split into two datasets: edx and validation. The edx data contains 90% of the original dataset, while validation contains 10%. The validation dataset is used exclusively for evaluating the final model developed on the edx dataset, and not for calculating any model fitting parameters.

Analysis

Data Exploration

Overview - Movies and Users Before a model can developed for the recommendation system, a better understanding of the structure of the data provided is needed.

```
dim(edx)
```

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

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```
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
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

From this, it is observed that the dataset contains 6 variables: userId, movieId, rating, timestamp, title, and genres. The data is in a tidy format, that is, each column is a variable and each row is an observation, so no additional data transformations are necessary. The ratings are on a scale of 1 to 5. It is also observed that timestamp is in units of seconds since January 1, 1970, title includes the release year, and genres can include multiple genres separated by "|".

Using this information, it was decided to create a more meaningful date variable, separate the title and release year, and create new variables representing day of the week and the week of the rating.

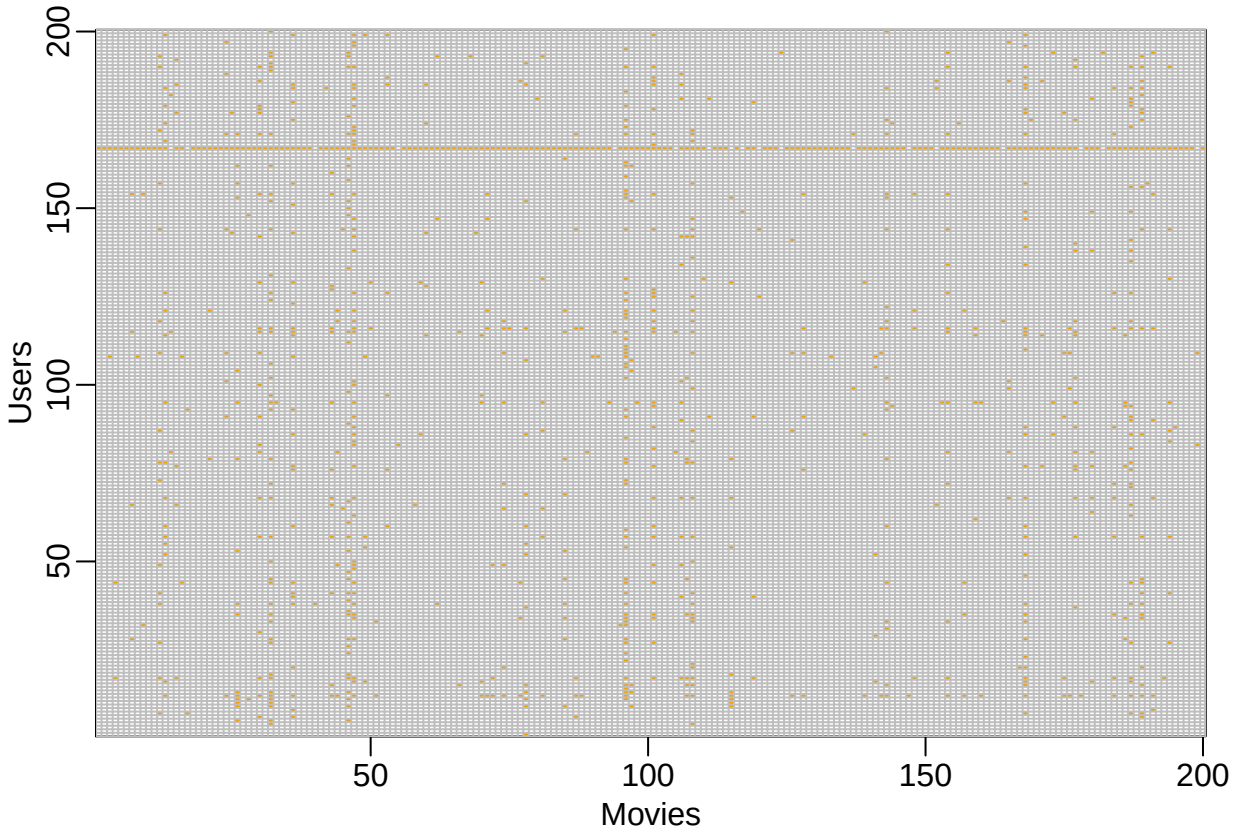
```
edx <- edx %>%
  extract(col=title,into=c("title","year"),regex="^(.*)\\((\\d{4})\\)$") %>%
  mutate(year=as.integer(year),
         date = as_datetime(timestamp),
         weekday = wday(date,label=TRUE),
         week = round_date(date, unit = "week"))
```

Further investigation determined the number of distinct users and movies.

```
##   n_users n_movies
## 1   69878    10677
```

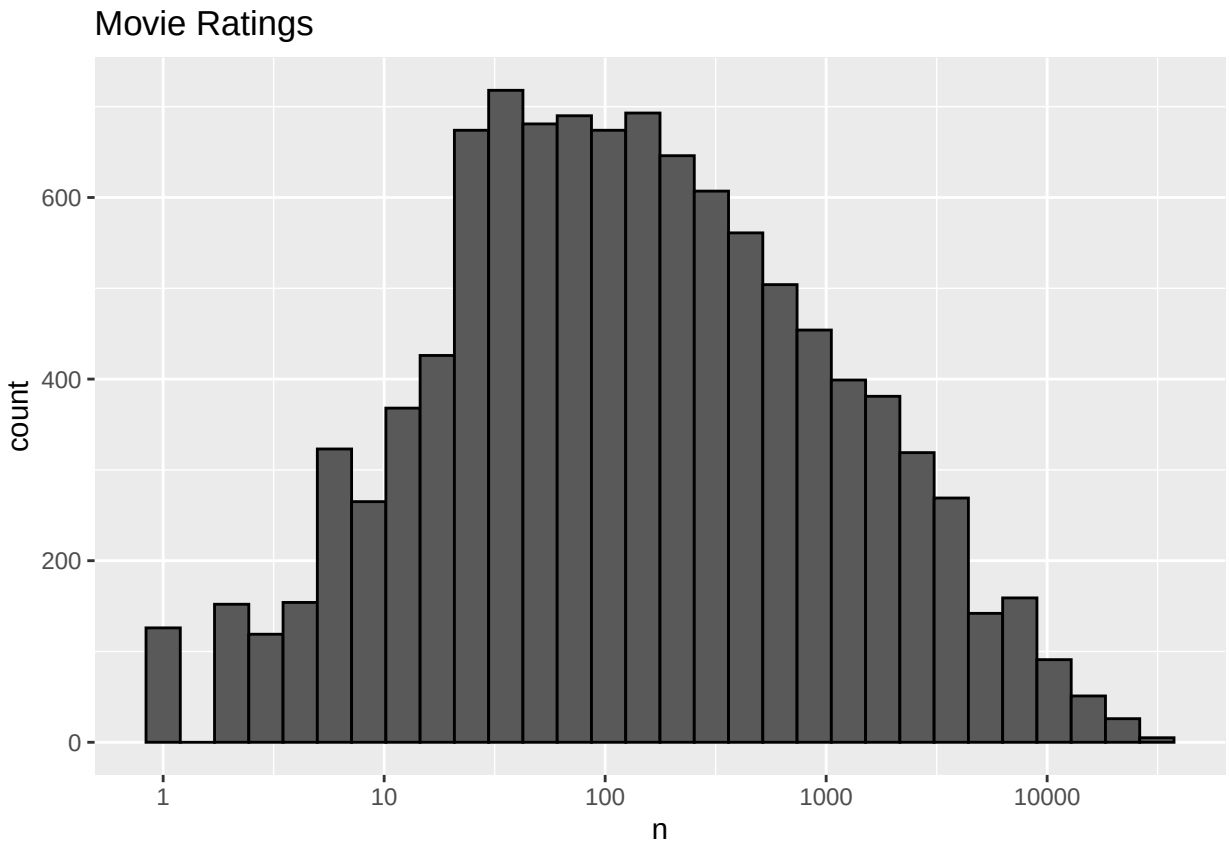
Not every movie received a rating from every user. To visualize how complete the dataset is, a random sample of 200 users was taken with the first 200 movies in the dataset. If a user provided a rating for that movie, a

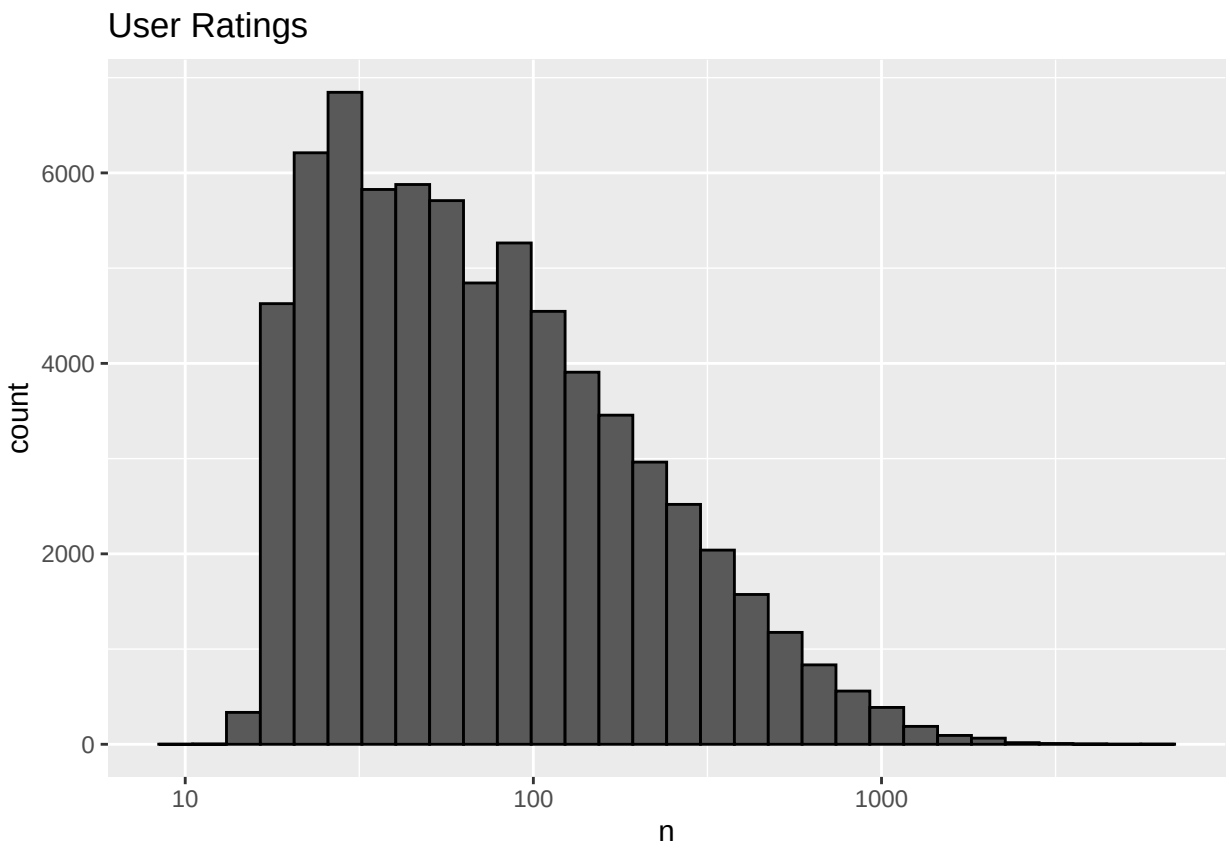
yellow square is filled in.



This shows that many user - movie combinations do not contain a rating and a user by movie matrix would contain several NAs.

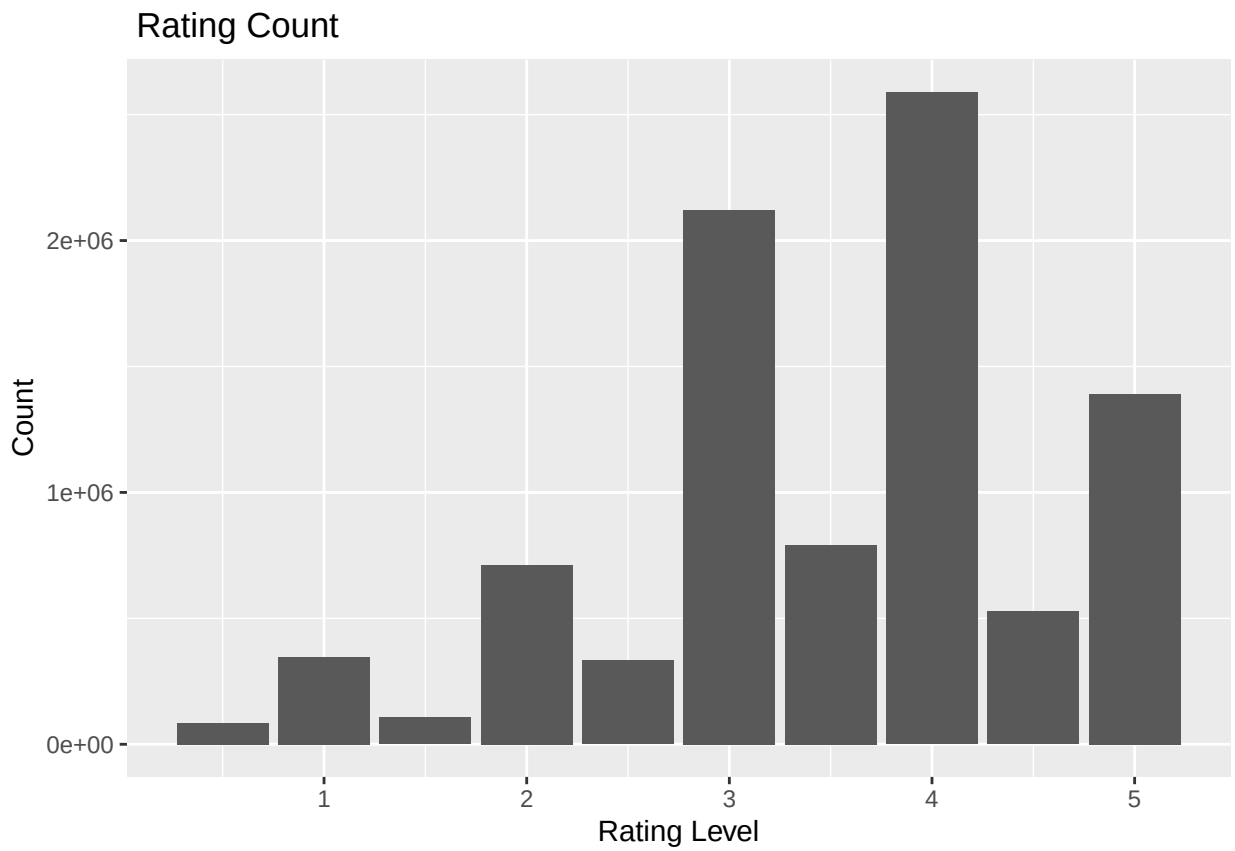
To investigate the frequency of movie and user ratings, histograms were created.


Movie Ratings
count
600
400
200
0
1
10
100
1000
10000
n



These plots show that the distribution of ratings per movie is approximately log normal, while the distribution of ratings per user is a positively skewed log normal distribution.

To explore the ratings in the dataset, a histogram of the ratings was created.



This histrogram indicates that the most common rating is 4, followed by 3 and 5. Whole number ratings are also more common than half ratings.

To determine the most rated movies, the following table was created.

```
## # A tibble: 10,407 x 2
##    title                                                count
##    <chr>                                                <int>
##  1 "Pulp Fiction "                                      31362
##  2 "Forrest Gump "                                      31079
##  3 "Silence of the Lambs, The "                         30382
##  4 "Jurassic Park "                                     29360
##  5 "Shawshank Redemption, The "                         28015
##  6 "Braveheart "                                        26212
##  7 "Fugitive, The "                                     26020
##  8 "Terminator 2: Judgment Day "                        25984
##  9 "Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) " 25672
## 10 "Batman "                                            24585
## # ... with 10,397 more rows
```

There a no suprises in this list, as they are all popular movies. Any movie fan will have likely seen all of these movies.

The following is a table with the least rated movies.

```
## # A tibble: 10,407 x 2
##    title                                                count
##    <chr>                                                <int>
##  1 "1, 2, 3, Sun (Un, deuz, trois, soleil) "                1
```

```
##  2 "100 Feet "                                                                  1
##  3 "4 "                                                                         1
##  4 "Accused (Anklaget) "                                                        1
##  5 "Ace of Hearts "                                                             1
##  6 "Ace of Hearts, The "                                                        1
##  7 "Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...~      1
##  8 "Africa addio "                                                              1
##  9 "Aleksandra "                                                                1
## 10 "Bad Blood (Mauvais sang) "                                                  1
## # ... with 10,397 more rows
```

These are more obscure movies that are likely independent films.

The movies with the highest average ratings are shown in the following table. Only movies with at least 1,000 ratings are included to ensure high ratings from few votes are not included.

```
## # A tibble: 1,893 x 3
##    title                                     n average
##    <chr>                                 <int>   <dbl>
##  1 "Shawshank Redemption, The "          28015    4.46
##  2 "Godfather, The "                     17747    4.42
##  3 "Usual Suspects, The "                21648    4.37
##  4 "Schindler's List "                   23193    4.36
##  5 "Casablanca "                         11232    4.32
##  6 "Rear Window "                         7935    4.32
##  7 "Sunset Blvd. (a.k.a. Sunset Boulevard) "  2922    4.32
##  8 "Third Man, The "                      2967    4.31
##  9 "Double Indemnity "                    2154    4.31
## 10 "Paths of Glory "                      1571    4.31
## # ... with 1,883 more rows
```

Conversely, movies with the lowest average ratings are shown in the following table.

```
## # A tibble: 1,893 x 3
##    title                                       n average
##    <chr>                                   <int>   <dbl>
##  1 "Battlefield Earth "                     1869    1.57
##  2 "Police Academy 6: City Under Siege "    1092    1.72
##  3 "Spice World "                           1288    1.74
##  4 "Police Academy 5: Assignment: Miami Beach "  1111    1.78
##  5 "Jaws 3-D "                              1052    1.79
##  6 "Home Alone 3 "                          1221    1.82
##  7 "Speed 2: Cruise Control "               2566    1.87
##  8 "Mighty Morphin Power Rangers: The Movie "  1316    1.88
##  9 "Look Who's Talking Now "                1059    1.88
## 10 "Grease 2 "                              1866    1.94
## # ... with 1,883 more rows
```

These lists could be helpful when trying to decide on a Friday night movie.

Genre As discovered during the earlier investigation, the genre variable can contain multiple genres separated by a "|" character. The data was first explored leaving the genres grouped together, and then with the genres separated out.

Grouped Genre The first question to answer is the number of unique genre groupings in the dataset.

```
##   n_genres
## 1      797
```

This shows that there are many different combinations of genres in our grouped genre varaible.

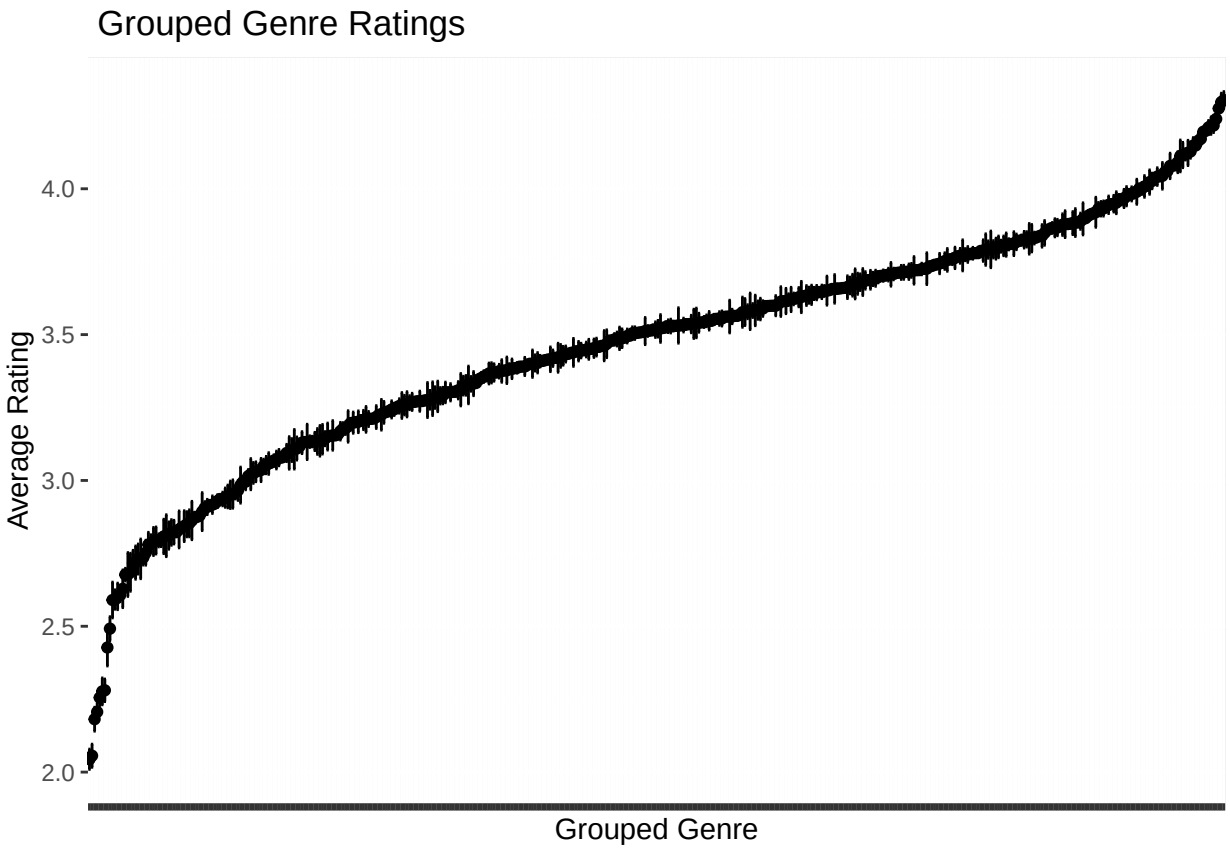
To determine which genres received the most and least ratings, the following tables were created.

```
## # A tibble: 25 x 2
##    genres                         n
##    <chr>                      <int>
##  1 Drama                     733296
##  2 Comedy                    700889
##  3 Comedy|Romance            365468
##  4 Comedy|Drama              323637
##  5 Comedy|Drama|Romance      261425
##  6 Drama|Romance             259355
##  7 Action|Adventure|Sci-Fi   219938
##  8 Action|Adventure|Thriller 149091
##  9 Drama|Thriller            145373
## 10 Crime|Drama               137387
## # ... with 15 more rows
```

```
## # A tibble: 25 x 2
##    genres                           n
##    <chr>                        <int>
##  1 Action|Crime|Drama|Thriller 65183
##  2 Action|Adventure             68688
##  3 Horror                       68738
##  4 Documentary                  70041
##  5 Action|Adventure|Fantasy     71176
##  6 Comedy|Crime                 73286
##  7 Horror|Thriller              75000
##  8 Thriller                     94662
##  9 Action|Sci-Fi|Thriller       95280
## 10 Action|Thriller              96535
## # ... with 15 more rows
```

One suprising result from this investigation, is that Action|Adventure received the second fewest ratings. Most summer blockbuster movies are of the action/adventure type, so one would assume these would have many ratings.

To visualize how the grouped genres affects ratings, a plot of ascending average rating with a confidence interval was created.



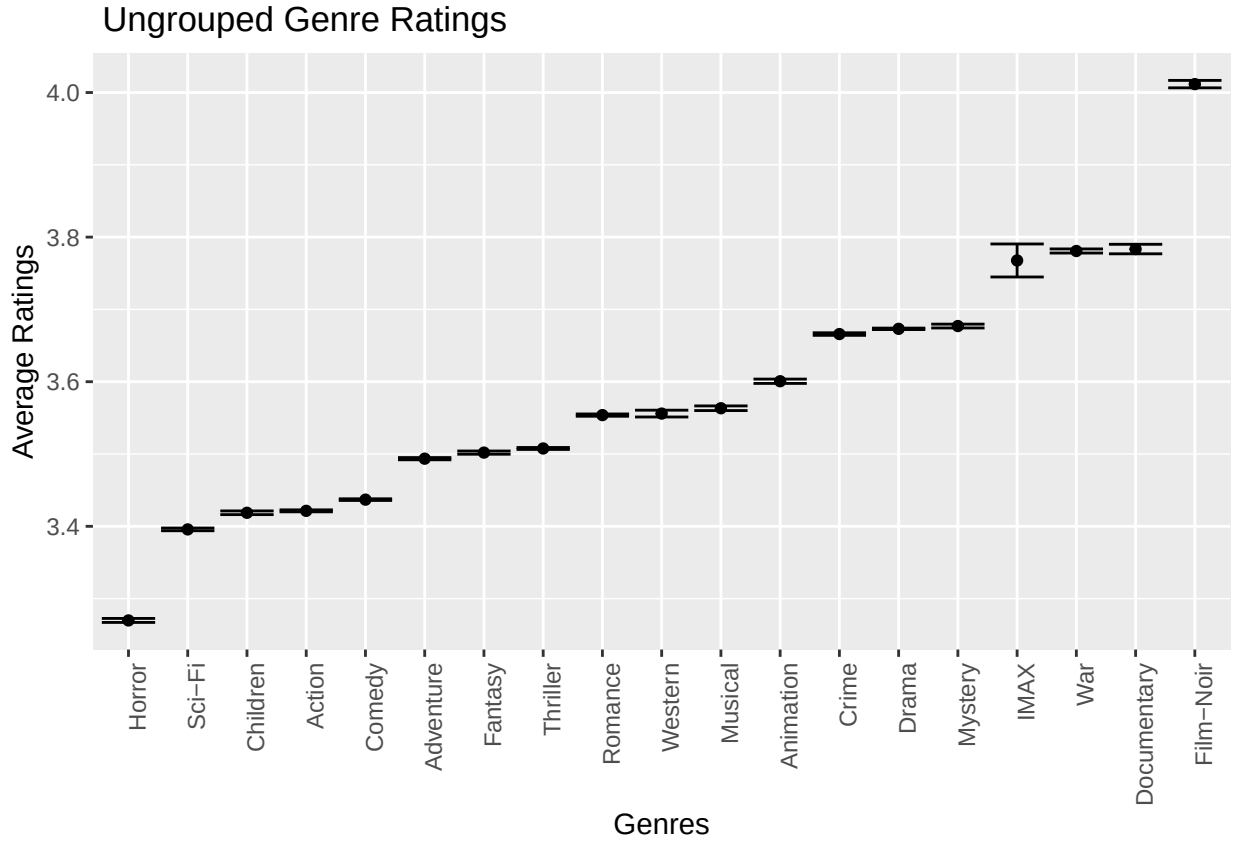
While there are too many grouped genres to be listed on the abscissa, the ordinate shows that grouped genres clearly have an effect on ratings and that the ratings within a grouped genre have a small confidence interval.

Ungrouped Genre As the grouped genre variable contains each genre separated by the "|" character, the command `separate_rows(genres, sep="\\|")` can be used to create duplicate rows, each with one of the ungrouped genres. This allows the determination the number of genres and how frequently they occur in the dataset.

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
```

```
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

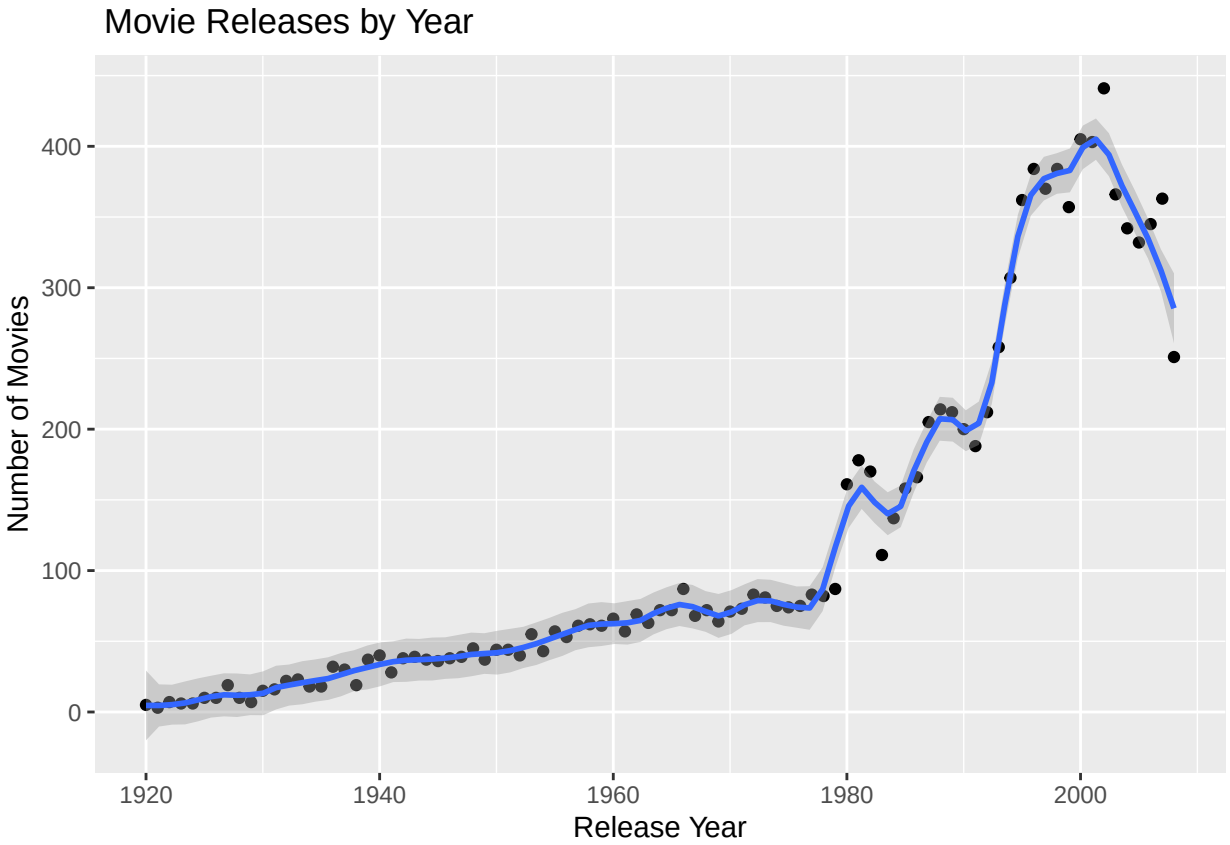
Similar to the previous investigation into grouped genres, to visualize how the ungrouped genres affects ratings, a plot of ascending average rating with a confidence interval was created.



This plot shows that ungrouped genres also have an effect on ratings with small confidence invervals to the average. Another interesting observation is that some of the genres with the fewest ratings (Film-Noir, Documentary), also receive the highest average ratings.

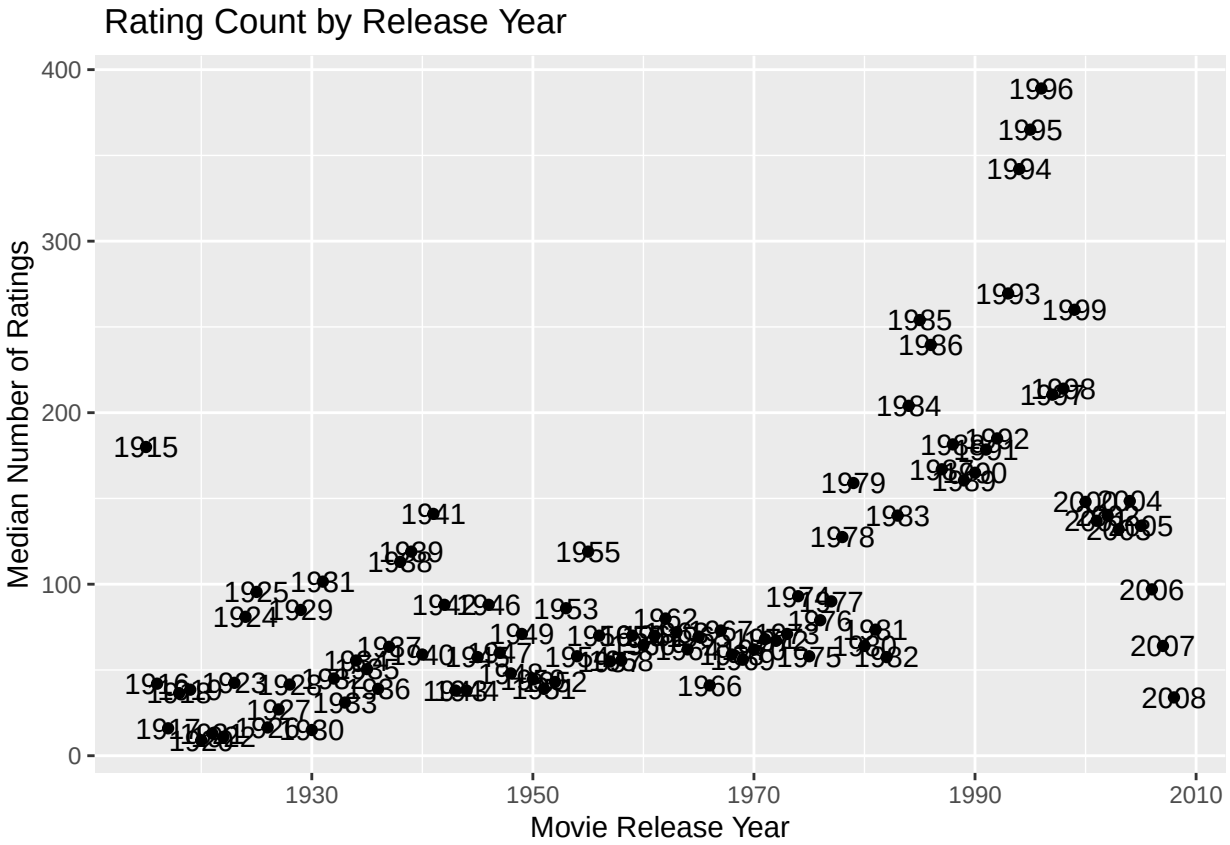
Release Year The original dataset included the release year in the title variable. Earlier in this investigation, we removed this from the title and created a new variable so that the effects of release year could be investigated.

The following plot shows the number of movies released each year since 1920.



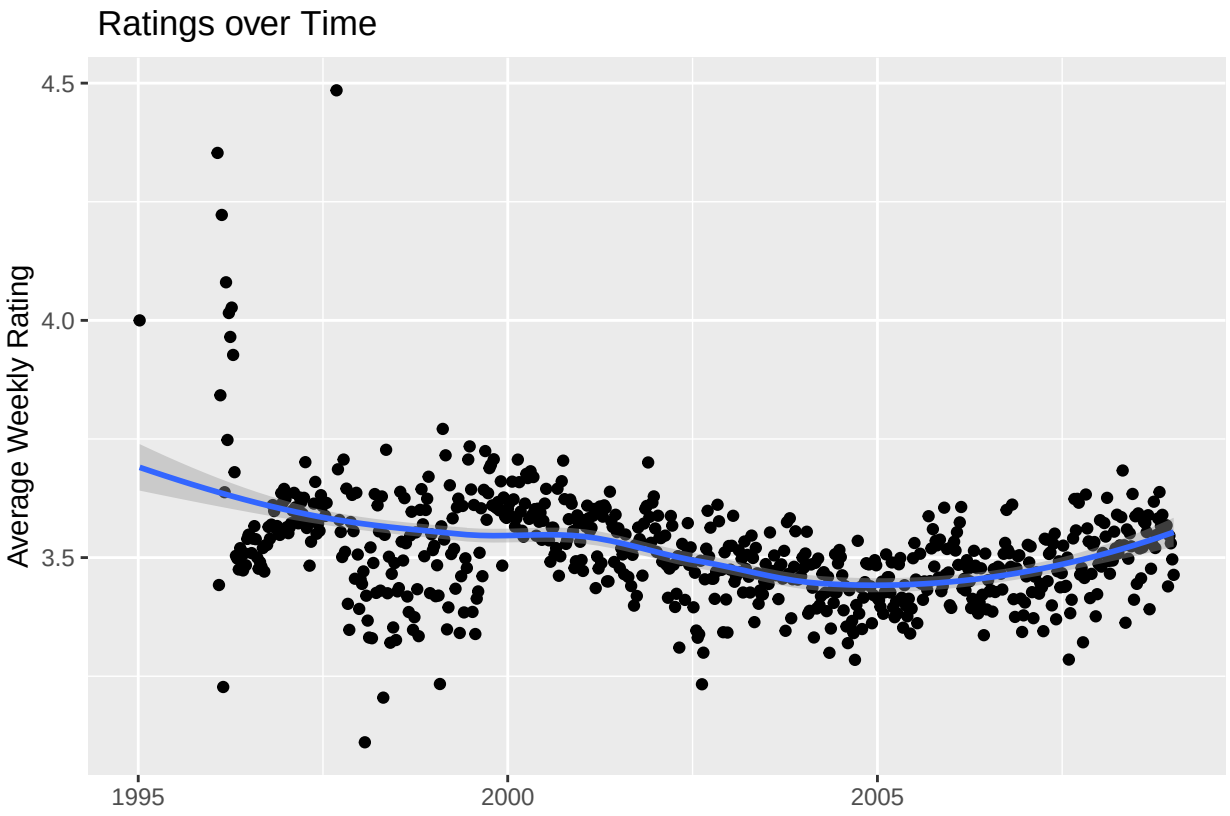
It can be seen that the number of movies steadily increases from 1920 until 1979. Starting in 1980 there is a sharp increase in the number of movies produced each year until approximately the year 2000, after which the number decreases.

To investigate how release year affects the median number of ratings, the following plot was created.



Similar to the previous plot, there is a period of steady increase in median number of ratings until approximately 1980, followed by a sharp increase, and then a decrease. The peaks in the two plots do not line up, however, with the median number of ratings peaking in 1996.

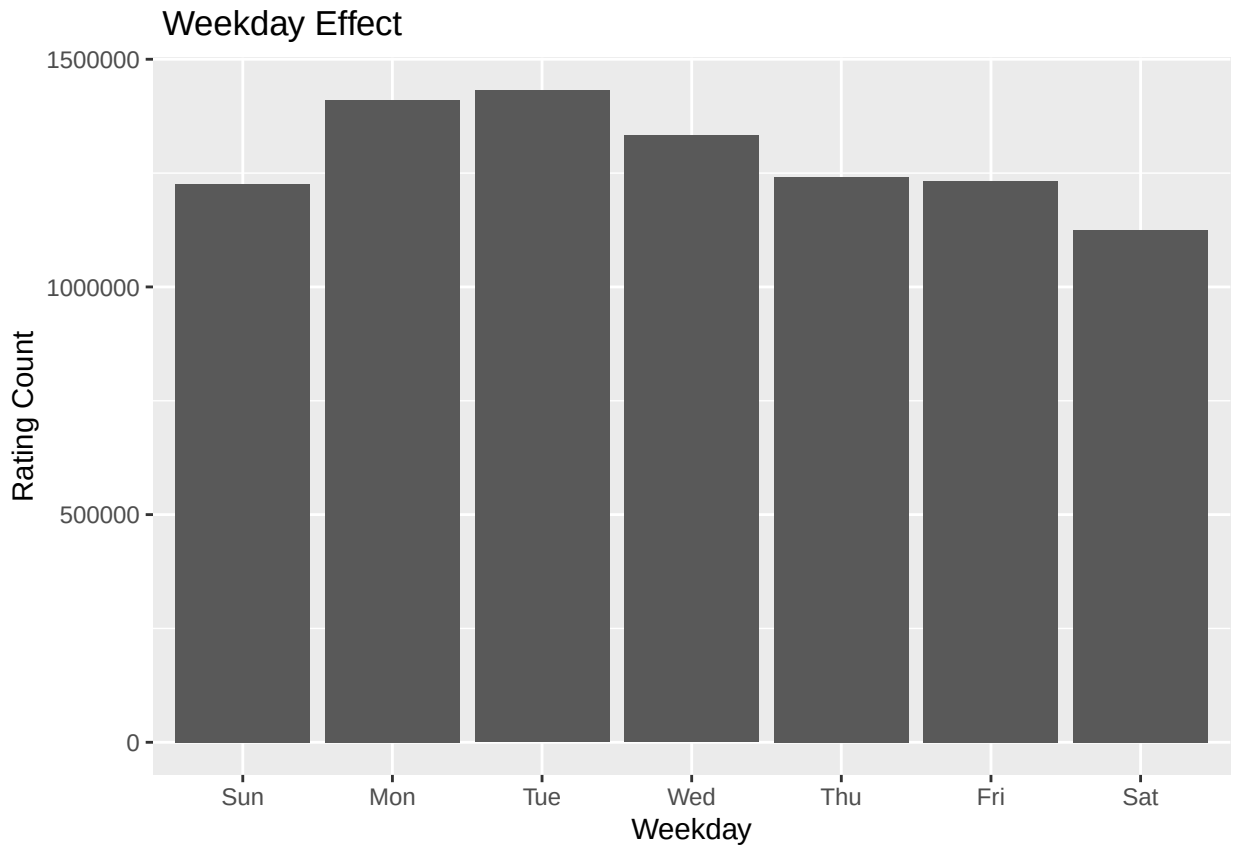
Rating Over Time Previously, the new variable week was created using the `round_date()` function from lubridate. This enables the exploration of any time effect on ratings.



As can be seen in the previous plot, there does appear to be a time effect on ratings. Since there does appear to be a time effect on the average rating per week, the following plot was created to determine if the time effect was consistent across a sample of movies. In this plot, the weekly average rating of the five movies with the highest overall average rating and more than 1,000 ratings is shown.

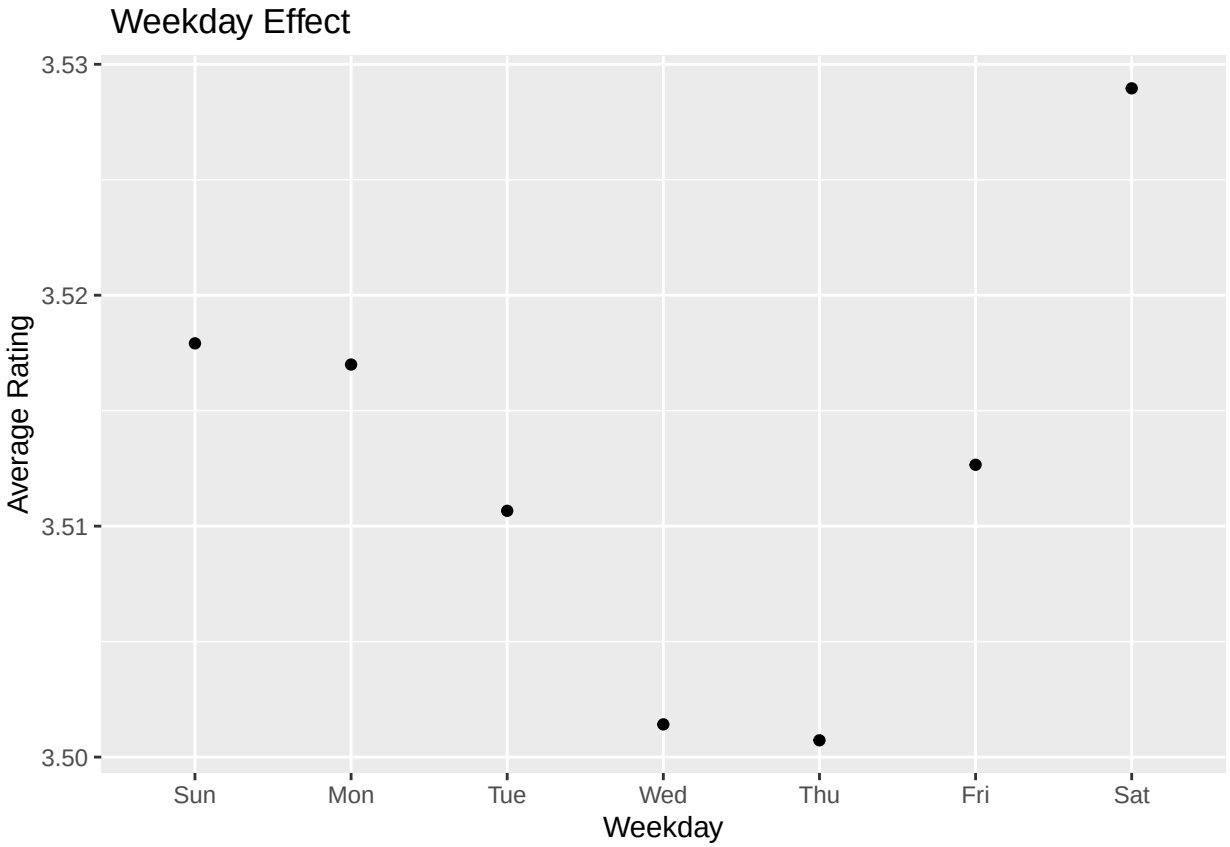
It appears that the average weekly ratings for these five movies have a similar shape, but the difference between them is marginal.

Day of the Week The last new variable that was created for the analysis is weekday, created using the `wday()` function. This was created to explore the possibility that users provide different ratings based on the day of the week they rate a movie. The first plot shows the number of ratings by day of the week.



Suprisingly, Tuesday is the most popular day for rating, while Saturday is the least popular.

The following plot displays the average rating by day of the week.

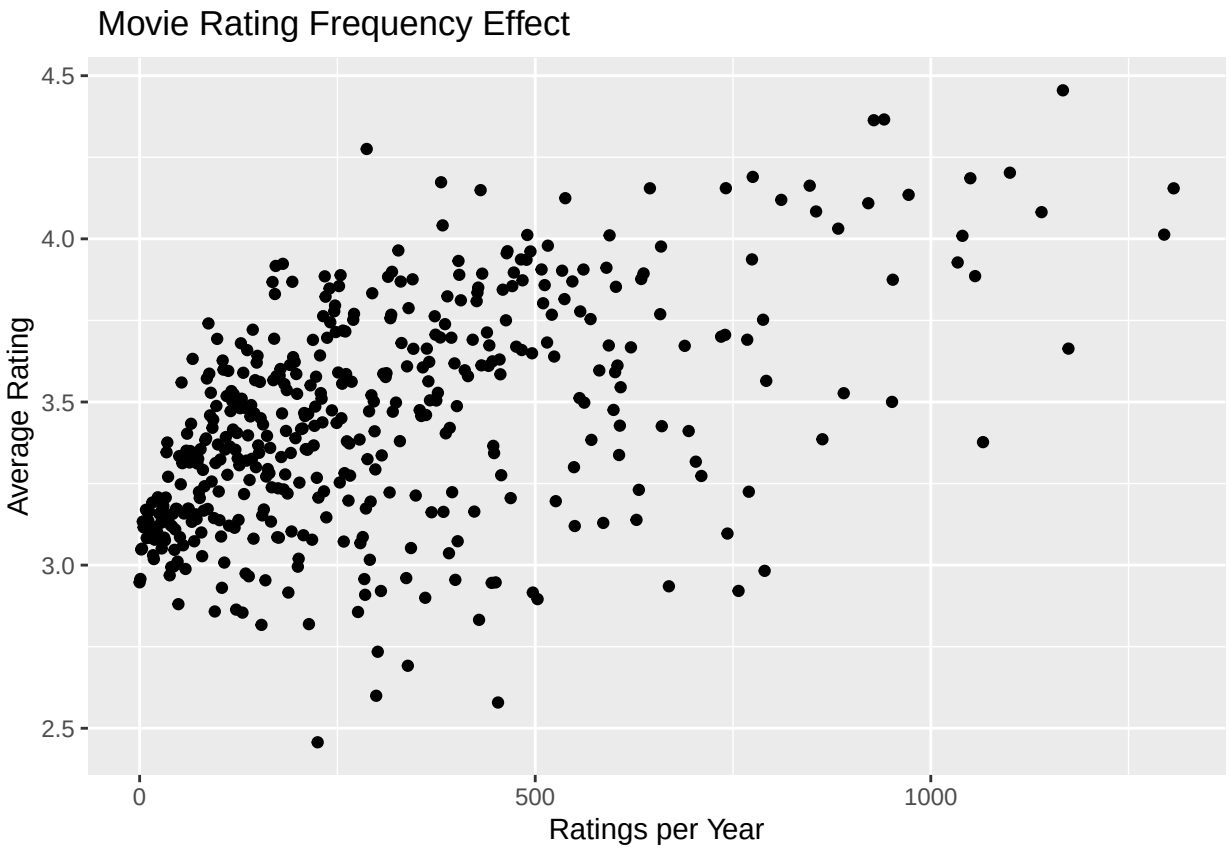


There does appear to be a small day of the week effect, with the highest average rating provided on Saturdays and the lowest ratings on Wednesday and Thursday.

Frequency of Movie Rating The next potential variable to explore is the effect of how frequent a movie is rated in period of time. The following table displays the ten movies released in 1980 or after, with the most ratings per year along with their average rating.

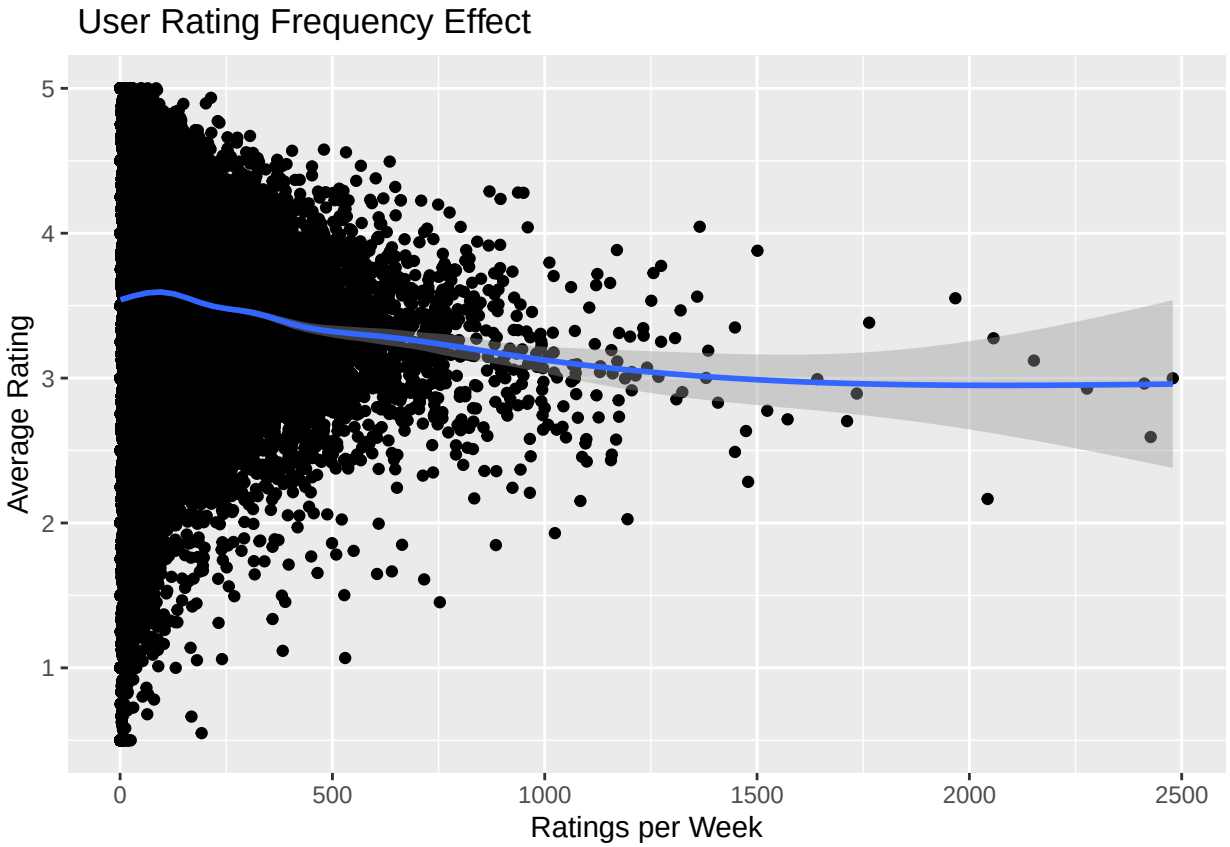
```
## # A tibble: 10 x 3
##    title                              rating_per_year avg_rating
##    <chr>                                        <dbl>      <dbl>
##  1 "Pulp Fiction "                              1307.       4.15
##  2 "Forrest Gump "                              1295.       4.01
##  3 "Jurassic Park "                             1174.       3.66
##  4 "Shawshank Redemption, The "                 1167.       4.46
##  5 "Braveheart "                                1140.       4.08
##  6 "Silence of the Lambs, The "                 1125.       4.20
##  7 "Matrix, The "                               1100.       4.20
##  8 "Independence Day (a.k.a. ID4) "             1066.       3.38
##  9 "Apollo 13 "                                 1056.       3.89
## 10 "American Beauty "                           1050        4.19
```

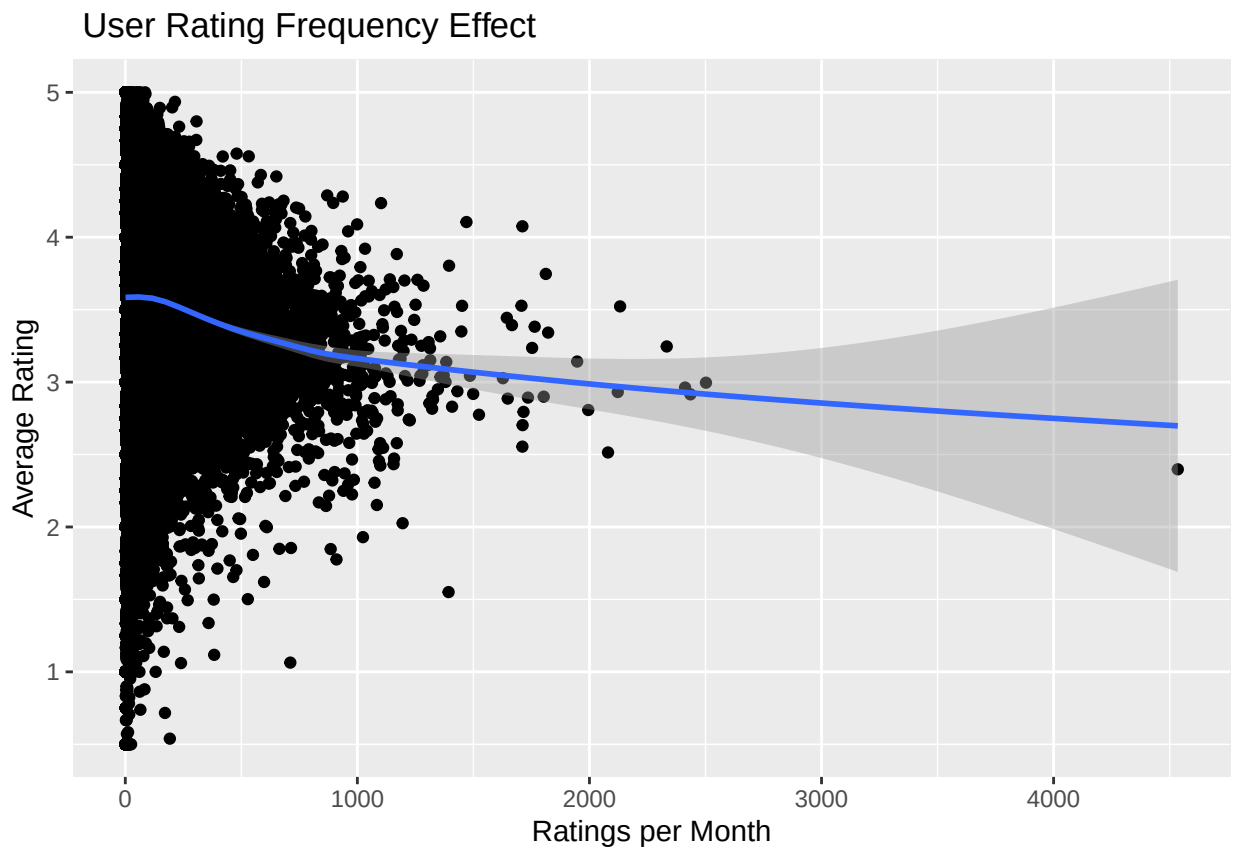
The following plot shows the average rating of a movie by the number or ratings per year.



Studying this plot, there appears to be a positive correlation between the number of ratings for a movie per year and the average rating, but there is also a large amount of scatter.

Frequency of User The corollary variable to the previous section is how frequent a user rates a movie in a period of time. It is hypothesized that may bias their ratings if they rate several movies at one time which they have viewed over their lifetime compared to rating a movie shortly after viewing it. The following two plots display the average rating versus the number of ratings provided in a week and month, respectively.


User Rating Frequency Effect
Average Rating
5
4
3
2
1
0
500
1000
1500
2000
2500
Ratings per Week



These plots show that there does appear to be a slight negative correlation between average rating and number of ratings a user provides in a week or month.

Modeling

Data Preparation In order to enable parameter tuning using cross-validation, the edx dataset is further split into a training set and a test set using the following code.

```
#partition of data for modeling
library(caret)
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
rm(test_index, temp, removed)
```

Loss Function To evaluate the error in this model, the loss function chosen is root mean square error (RMSE), which is given by the following formula.

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

where $\hat{y}_{u,i}$ is the predicted rating by user u for movie i, while $y_{u,i}$ is the actual rating, and N is the total number of ratings.

A function to calculate the RMSE is generated using the following code.

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

Modeling Approach The modeling approach for this study begins with the models developed during the PH125.8x Data Science: Machine Learning course. The initial model was

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

where μ is the average rating and $\varepsilon_{u,i}$ is the error term. This simple model can be created using the following code.

```
##Simple - average of all ratings
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

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

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results %>% kable(digits=5)
```

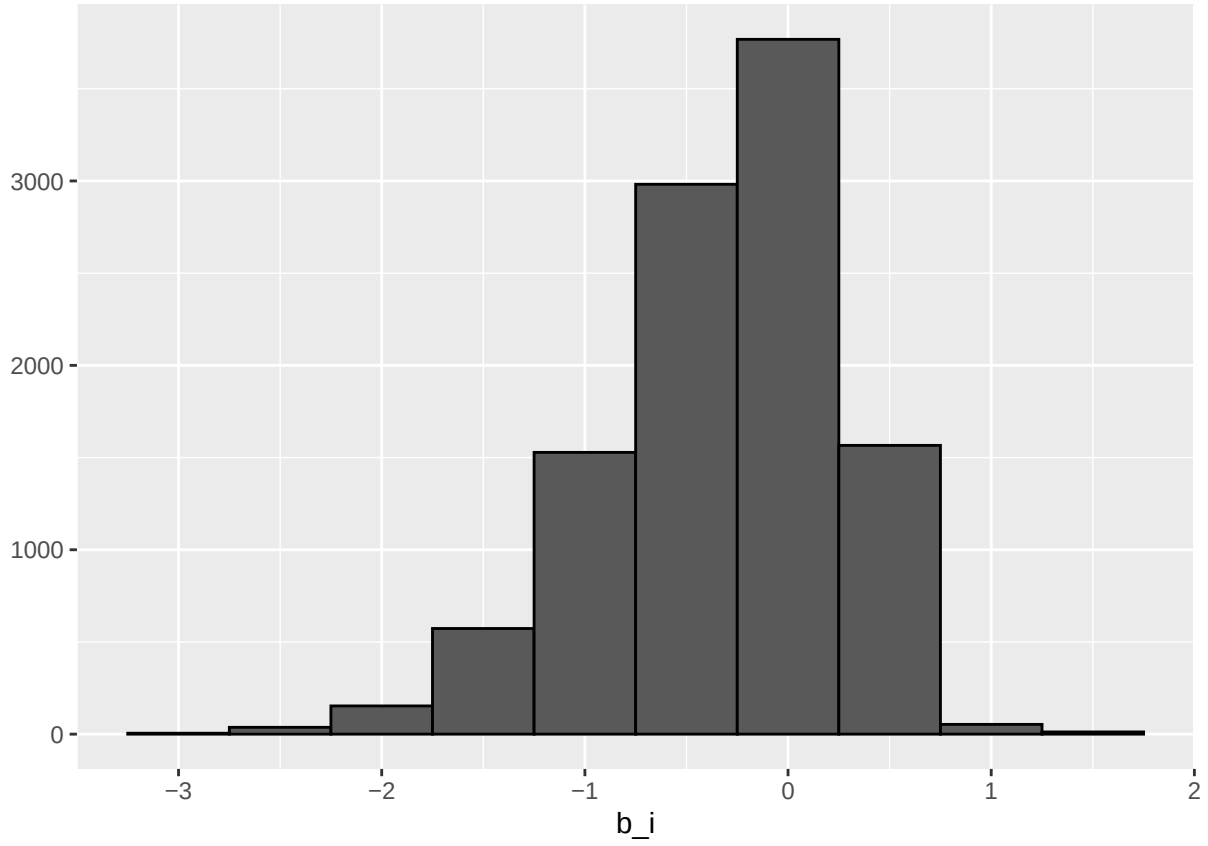
method	RMSE
Just the average	1.0599

The next model generated included a movie effect. This changed the model to

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

where b_i is the average rating given to a particular movie. This model is created using the following code.

```
##movie effects
mu <- mean(train_set$rating)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
qplot(b_i, data = b_i, bins = 10, color = I("black")) #histogram of b_i
```




```
predicted_ratings <- test_set %>%
  left_join(b_i, by="movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = model_1_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374

By including a movie effect term, the RMSE decreases by a significant amount.

The third model generated included a user effect. This changed the model to

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

where b_u is the average rating provided by a user after removing the movie effect. This effect is modeled using the following code.

```
##user effects
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) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                 RMSE = model_2_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593

By including a user effect term, the RMSE does decrease, but not to the same magnitude as the movie effect contributed.

While in the PH125.8x Data Science: Machine Learning course regularization was added at this point in the model development, in this study regularization is added after all of the effects are included in the model.

The first new variable added in this study is the genre effect. As discussed in the data exploration section, the genre variable was analyzed as a grouped listing of genres as well as ungrouped, as individual genres. Both methods are investigated here to determine which has better predictive capability. Starting with grouped genres, the model becomes

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

where b_g is the average grouped genre rating after accounting for the other effects. This effect is modeled using the following code

```
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))
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)
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre (grouped) Effect Model",
                                 RMSE = model_3_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559

This results in a small decrease in the RMSE.

For the ungrouped genre effect, the model would be

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + \varepsilon_{u,i}$$

where $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k. This effect is modeled using the following code.

```
b_g2 <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  separate_rows(genres, sep="\\|") %>%
  group_by(genres) %>%
  summarize(b_g2 = mean(rating - mu - b_i - b_u))
predicted_ratings <- test_set %>%
  separate_rows(genres, sep="\\|") %>%
  left_join(b_g2, by="genres") %>%
  group_by(movieId,userId) %>%
  summarize(b_g_sum = sum(b_g2)) %>%
  #opposite of separate_rows and sum b_gs
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  mutate(pred = mu + b_i + b_u + b_g_sum) %>%
  pull(pred)
model_4_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre (ungrouped) Effect Model",
                                 RMSE = model_4_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171

It appears that by ungrouping the genres, the RMSE is significatly increased. Because of this, the grouped genre methodology is chosen to account for the genre effect.

The next variable added to the model is the release year effect. The model then becomes

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

where b_y is the average rating for the release year, after accounting for the previous effects. This can be modeled using the following code.

```
#Model - release year effects
b_y <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g))
predicted_ratings <- test_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  pull(pred)

model_5_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre (grouped) + Year Effect Model",
                                 RMSE = model_5_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171
Movie + User + Genre (grouped) + Year Effect Model	0.86542

Including release year effects has a small contribution in decreaseing the RMSE.

While exploring the data, it was observed that day of the week effects appeared to have a small effect on ratings. Adding this effect to the model creates

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + b_{wd} + \varepsilon_{u,i}$$

where b_{wd} is the average rating for the weekday, after accounting for the previous effects. This effect is modeled using the following code.

```
#Model - day of week effects
b_wd <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  group_by(weekday) %>%
  summarize(b_wd = mean(rating - mu - b_i - b_u - b_g - b_y))
predicted_ratings <- test_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
```

```
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  left_join(b_wd, by="weekday") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y + b_wd) %>%
  pull(pred)
model_6_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre (grouped) + Year + Weekday Effect Model",
                                 RMSE = model_6_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171
Movie + User + Genre (grouped) + Year Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday Effect Model	0.86542

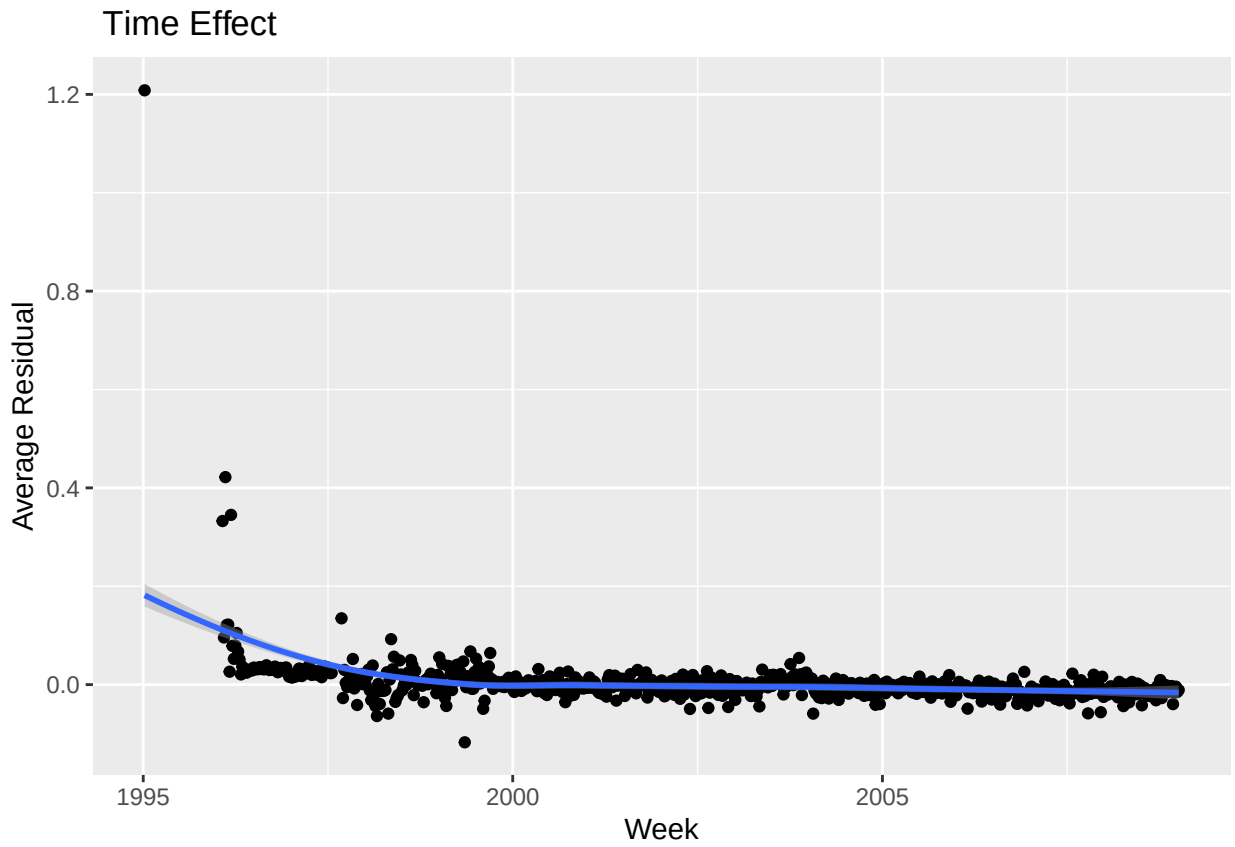
Including the weekday effect has a very small change to the RMSE, as there is no reduction in the RMSE out to five digits.

The final term added to the model in this study is the time effect. While it was observed previously that each movie, and likely user, has a different time effect, for computational simplicity, this study only accounts for the average rating as a function of time. This changes the model to

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + b_{wd} + f(w_{u,i}) + \varepsilon_{u,i}$$

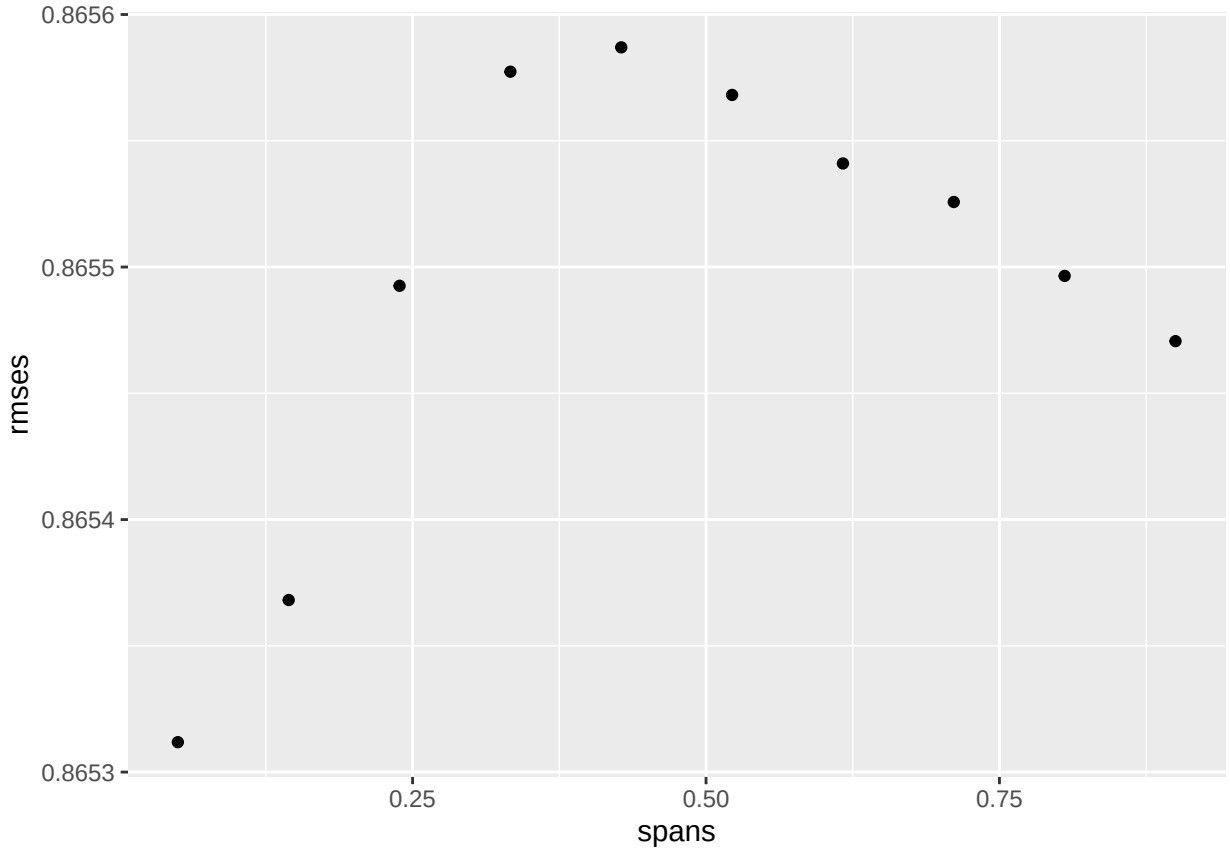
where $f(w_{u,i})$ is a smooth function of $w_{u,i}$, which is the week for user u's rating of movie i. To find this fuction, the residual left after accounting for all the previous effects is calculated. A loess function is then fit to the average residual as a function of week. To determine the optimal span, the RMSE is minimzied on the test data. The final model is then created using this optimal span parameter. This effect is modeled using the following code.

```
#Model - time effect (week) - loess of week #
#plot residuals as fn of week
resid <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  left_join(b_wd, by="weekday") %>%
  mutate(resid = rating - mu - b_i - b_u - b_g - b_y - b_wd) %>%
  select(week,resid) %>%
  group_by(week) %>%
  summarize(average = mean(resid))
resid %>%
  ggplot(aes(y=average, x=week)) +
  geom_point() +
  geom_smooth() +
  labs(title =" Time Effect", x = "Week", y = "Average Residual")
```




```
#train loess model on residual data - use test data to determine optimal span parameter
spans = seq(0.05, 0.9, len = 10)
rmses <- sapply(spans, function(s){
  train_loess_t <- resid %>%
    mutate(week = as.numeric(week)) %>%
    loess(average ~ week, data = ., span = s, degree = 2)
  predicted_ratings <- test_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_y, by="year") %>%
    left_join(b_wd, by="weekday") %>%
    mutate(week = as.numeric(week),
           b_t = predict(train_loess_t, week),
           pred = mu + b_i + b_u + b_g + + b_y + b_wd + b_t) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(spans, rmses)
```




```
span <- spans[which.min(rmses)]
span
```

```
## [1] 0.05
```

```
#train model using optimal span parameter
train_loess_t <- resid %>%
  mutate(week = as.numeric(week)) %>%
  loess(average ~ week, data = ., span = span, degree = 2)
predicted_ratings <- test_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  left_join(b_wd, by="weekday") %>%
  mutate(week = as.numeric(week),
         b_t = predict(train_loess_t, week),
         pred = mu + b_i + b_u + b_g + b_y + b_wd + b_t) %>%
  pull(pred)
model_7_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre (grouped) + Year + Weekday + Time Effect
                                 RMSE = model_7_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990

method	RMSE
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171
Movie + User + Genre (grouped) + Year Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday + Time Effect Model	0.86531

Regularization As described in the PH125.8x Data Science: Machine Learning course, regularization is a technique that penalizes large predictions created using small sample sizes. When sample sizes are small, there is larger uncertainty in the estimate, as shown by larger standard errors. To prevent these estimates from increasing the RMSE, a penalty term, λ_i is added to the model, so that instead of minimizing

$$\sum_{u,i} \left(y_{u,i} - \mu - b_i - b_u - b_g - b_y - b_{wd} - b_t\right)^2$$

the following equation is minimized.

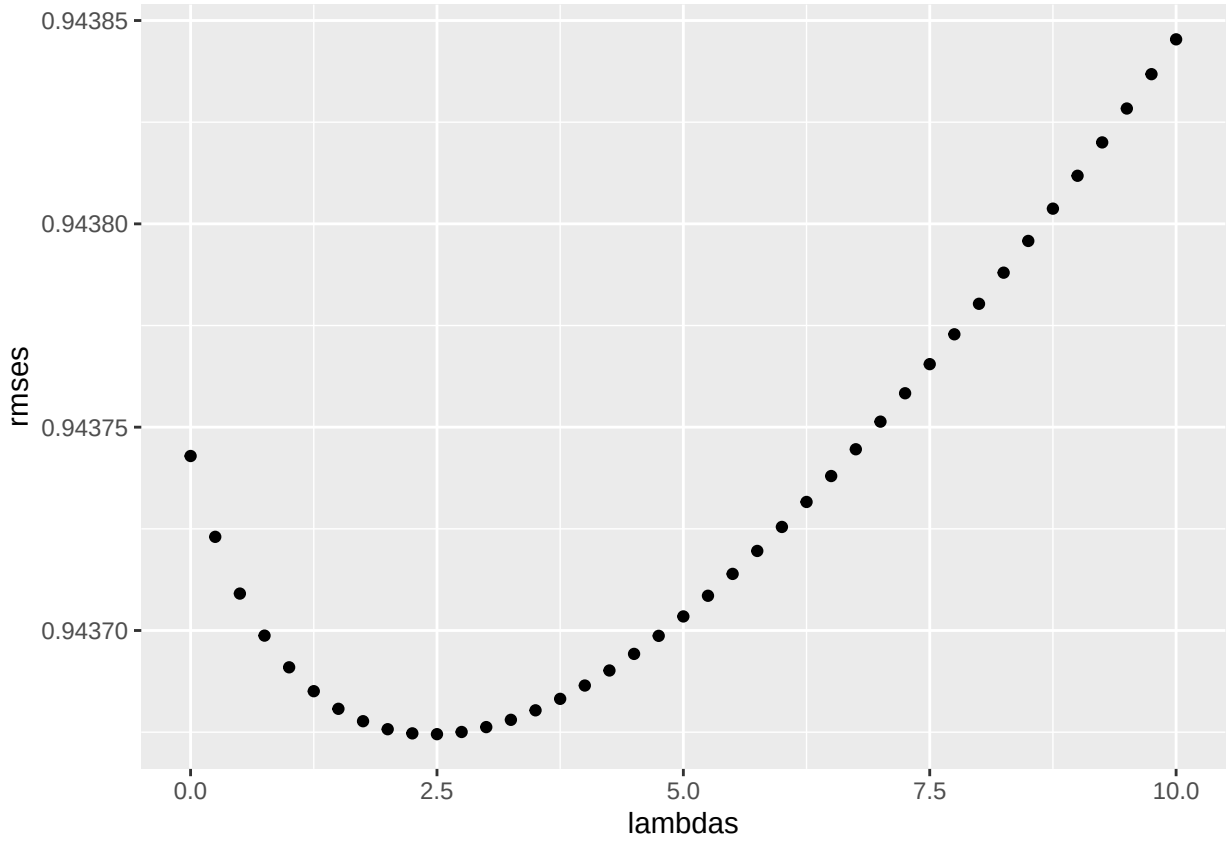
$$\sum_{u,i} \left(y_{u,i} - \mu - b_i - b_u - b_g - b_y - b_{wd} - b_t\right)^2 + \lambda_1 \left(\sum_i b_i^2\right) + \lambda_2 \left(\sum_u b_u^2\right) + \lambda_3 \left(\sum_g b_g^2\right) + \lambda_4 \left(\sum_y b_y^2\right) + \lambda_5 \left(\sum_{wd} b_{wd}^2\right)$$

The λ_i terms are fitting parameters, so the test dataset is used to select the optimal value.

```
lambda <- vector("double",4)
lambdas <- seq(0, 10, 0.25)

#determine lambda for movie effects
rmses <- sapply(lambdas, function(l){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```



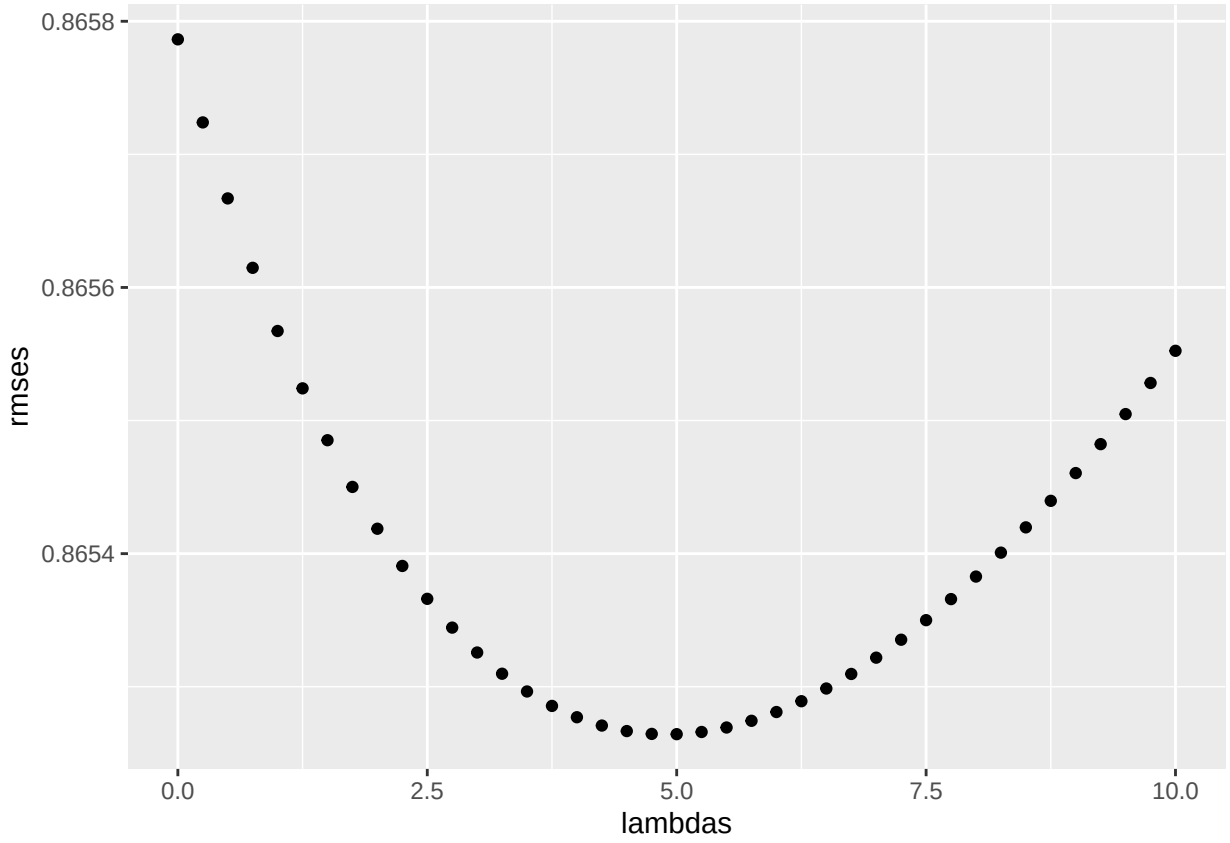

```
lambda[1] <- lambdas[which.min(rmses)]
lambda[1]
```

```
## [1] 2.5
```

```
#recalculate movie effect with optimal lambda
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda[1]))

#determine lambda for user effects
rmses <- sapply(lambdas, function(l){
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```



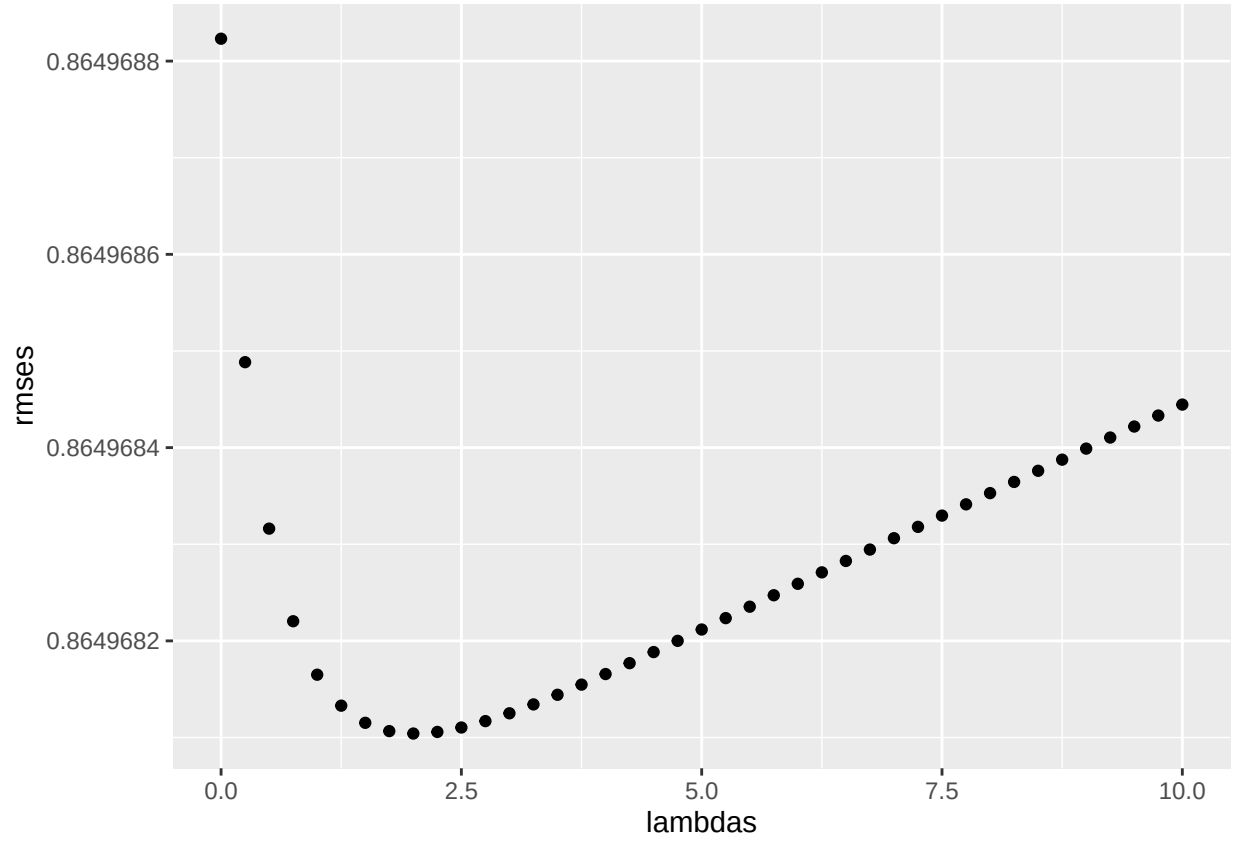

```
lambda[2] <- lambdas[which.min(rmses)]
lambda[2]
```

```
## [1] 5
```

```
#recalculate user effect with optimal lambda
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda[2]))

#determine lambda for genre effects
rmses <- sapply(lambdas, function(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)
```




```
lambda[3] <- lambdas[which.min(rmses)]
lambda[3]
```

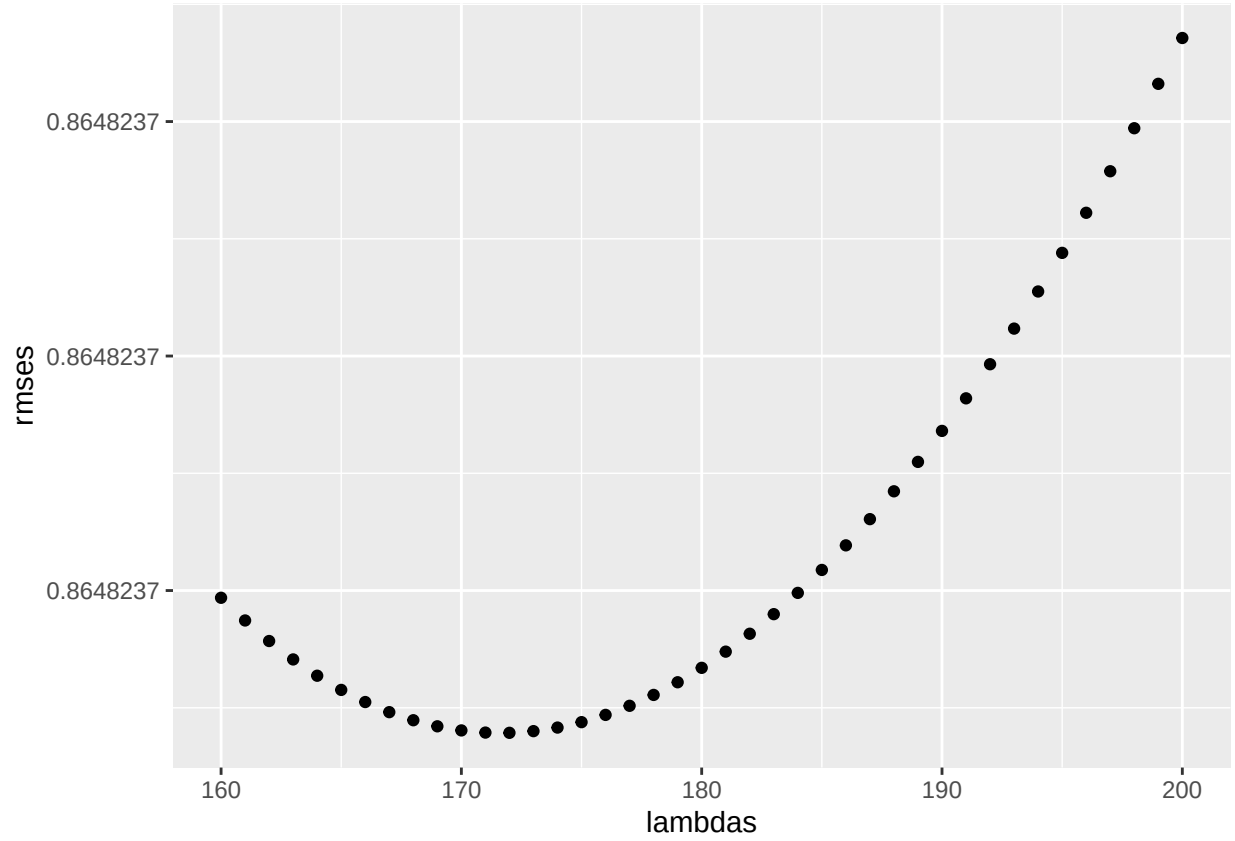
```
## [1] 2
```

```
#recalculate genre effect with optimal lambda
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()+lambda[3]))

#determine lambda for year effects
lambdas <- seq(160, 200, 1)
rmses <- sapply(lambdas, function(l){
  b_y <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u - b_g)/(n()+l))
  predicted_ratings <- test_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
```

```
    left_join(b_y, by="year") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```




```
lambda[4] <- lambdas[which.min(rmses)]
lambda[4]
```

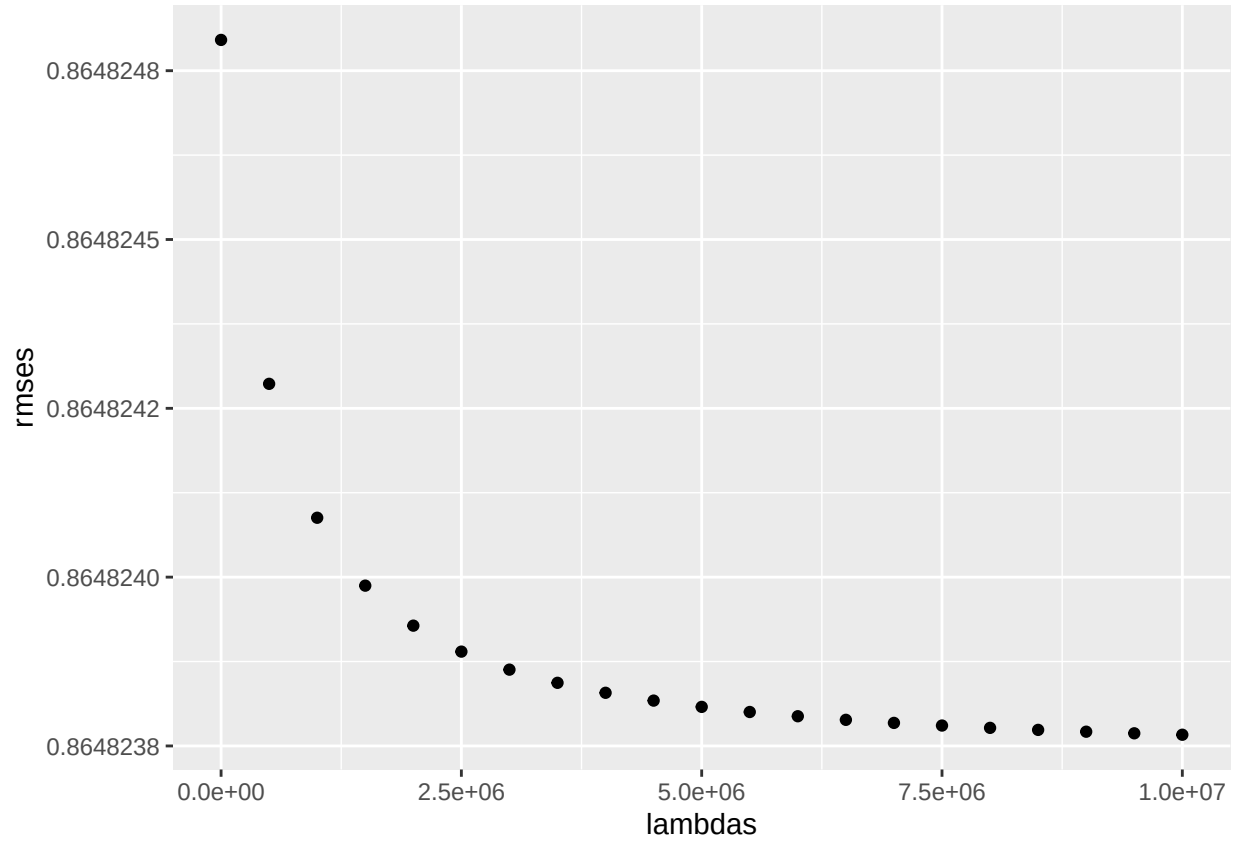
```
## [1] 172
```

```
#recalculate year effect with optimal lambda
b_y <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u - b_g)/(n()+lambda[4]))

#determine lambda for week day effects
lambdas <- seq(0, 10000000, 500000)
rmses <- sapply(lambdas, function(l){
  b_wd <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
```

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

qplot(lambdas, rmses)
```



Applying regularization to the weekday term produced some suprising results. The RMSE as a function of λ_i term does not produce a parabola. The RMSE decreases asymptotically as the λ_i term increases. This suggests that removing the weekday term would actually decrease the RMSE. After reviewing the previously calculated RMSE values with more significant digits, it was discovered that adding a weekday term does, in fact, increase the RMSE of this prediction. Because of this, the weekday term was removed from the model.

After regularizing the movie, user, genre, and release year terms, the loess fit of the time term was re-calculated.

```
#retrain loess model of time effect with updated movie, user, genre, and year effects
resid <- train_set %>%
```

```
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  mutate(resid = rating - mu - b_i - b_u - b_g - b_y) %>%
  select(week,resid) %>%
  group_by(week) %>%
  summarize(average = mean(resid))

train_loess_t <- resid %>%
    mutate(week = as.numeric(week)) %>%
    loess(average ~ week, data = ., span = span, degree = 2)
predicted_ratings <- test_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  mutate(week = as.numeric(week),
         b_t = predict(train_loess_t, week),
         pred = mu + b_i + b_u + b_g + b_y +  b_t) %>%
  pull(pred)
model_8_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User + Genre + Year + Time Effect Model",
                                 RMSE = model_8_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171
Movie + User + Genre (grouped) + Year Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday + Time Effect Model	0.86531
Regularized Movie + User + Genre + Year + Time Effect Model	0.86465

As a final step in model development, it was noticed that while the ratings are on a scale zero to five, the predicted values contain unrealistic values less than zero and greater than five.

```
summary(predicted_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.5254  3.1372  3.5657  3.5122  3.9432  6.0097
```

By truncating the predicted values to be between zero and five, a further slight decrease in RMSE is obtained.

```
#min/max adjustment - the prediction scale is 0 to 5 -
#any scores over 5 change to 5 and any below 0, change to 0
predicted_ratings[predicted_ratings > 5] <- 5
predicted_ratings[predicted_ratings < 0] <- 0
model_9_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
```

```
                          tibble(method="Regularized Movie + User + Genre + Year + Time Effect Model wi
                                 RMSE = model_9_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171
Movie + User + Genre (grouped) + Year Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday + Time Effect Model	0.86531
Regularized Movie + User + Genre + Year + Time Effect Model	0.86465
Regularized Movie + User + Genre + Year + Time Effect Model with Min/Max Adjustment	0.86454

Results

After removing the weekday term and including a minimum and maximum value for the prediction, the final model is

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + f(w_{u,i}) + \varepsilon_{u,i}$$
$$min(5, Y_{u,i})$$
$$max(0, Y_{u,i})$$

As a final evaluation of the model, the effect terms are caluculated on the entire edx dataset, instead of just the training set. The new variables year, date, and week are then created in the validation dataset, and predictions are created using the validation dataset, with a calculation of RMSE. All of the effects and tuning parameters were established on the edx dataset and the validation dataset is only used for the final prediction and calculation of RMSE.

```
#create the same variables for validation data as edx dataset - except weekday not needed
validation <- validation %>%
  extract(col=title,into=c("title","year"),regex="^(.*)\\((\\d{4})\\)$") %>%
  mutate(year=as.integer(year),
         date = as_datetime(timestamp),
         week = round_date(date, unit = "week"))

#####Train final model on whole edx dataset
mu <- mean(edx$rating)

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda[1]))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda[2]))

b_g <- edx %>%
```

```
  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()+lambda[3]))

b_y <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u - b_g)/(n()+lambda[4]))

resid <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  mutate(resid = rating - mu - b_i - b_u - b_g - b_y) %>%
  select(week,resid) %>%
  group_by(week) %>%
  summarize(average = mean(resid))

train_loess_t <- resid %>%
  mutate(week = as.numeric(week)) %>%
  loess(average ~ week, data = ., span = span, degree = 2)

predicted_ratings <- validation %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_y, by="year") %>%
  mutate(week = as.numeric(week),
         b_t = predict(train_loess_t, week),
         pred = mu + b_i + b_u + b_g + b_y +  b_t) %>%
  pull(pred)
predicted_ratings[predicted_ratings > 5] <- 5
predicted_ratings[predicted_ratings < 0] <- 0

model_10_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Validation Data",
                                 RMSE = model_10_rmse ))
rmse_results %>% kable(digits=5)
```

method	RMSE
Just the average	1.05990
Movie Effect Model	0.94374
Movie + User Effects Model	0.86593
Movie + User + Genre (grouped) Effect Model	0.86559
Movie + User + Genre (ungrouped) Effect Model	1.23171
Movie + User + Genre (grouped) + Year Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday Effect Model	0.86542
Movie + User + Genre (grouped) + Year + Weekday + Time Effect Model	0.86531

method	RMSE
Regularized Movie + User + Genre + Year + Time Effect Model	0.86465
Regularized Movie + User + Genre + Year + Time Effect Model with Min/Max Adjustment	0.86454
Validation Data	0.86406

This model was able to obatin an RMSE of 0.86406 on the validation data set, which met the goal of obtaining an RMSE of 0.86490.

Conclusion

Using the groundwork established during the PH125.8x Data Science – Machine Learning course, an improved recommendation system was developed that was able to reach an RMSE of 0.86406 against the goal of 0.86490. By including effects for genre, release year, and time, as well as by implementing regularization for all terms except time, this investigation was able to improve upon the performance of the original model. Further improvements to the model could be made by incorporating additional effects, such as frequency of movie ratings, frequency of user ratings, or the time effect on ratings for individual movies, users, or genres. Attempts to incorporate dimension reducing techniques such as singular value decomposition (SVD) or principal component analysis (PCA) were not fruitful, but further investigation and study may find value in these technqiues.