MovieLens Ratings Prediction Project Report

Becky Johnson

3/10/2021

Contents

---

1 Overview

Recommendation systems are frequently encountered in the world of online shopping and online viewing. Amazon uses predictive models to suggest products we may want to purchase based on our past buying behavior or purchases other customers have made who have similar buying patterns to ours. Netflix uses predictive recommendation systems to suggest movies or series we might enjoy. Goodreads suggest books a reader might like based on past reading behavior and the reader's "To Read" list. Examples of recommendation systems are endless.

In Data Science courses, learning to build a recommendation system is foundational. This project is the first of two required projects to complete the HarvardX Data Science Capstone course on the edX.org platform. Here, we create a recommendation system to predict how a user will rate a specific movie using data available on the grouplens.org site.

1.1 Project Overview

Once the problem has been articulated, there are 4 main steps in the process of building the model (these correspond to sections in the associated R file):

1. Identify the data and wrangle it into a workable dataset
2. Explore and visualize the data
3. Fit a model using several techniques to find a model that minimizes the error between actual and predicted values
4. Assess the model using a "hold back" dataset

After the model has been built, we can draw conclusions and suggest further analysis.

1.2 Problem Statement

Creating a recommendation system can be thought of as trying to fill in missing values in a matrix of users and items (in this case movies). In general, users rate only a few movies relative to the total population of movies. Our database has data on approximately 10,000 movies. I've likely seen only around 500 and of those I don't remember enough about many of them to provide a rating. The table below highlights this issue. Our recommendation system is built to predict the "NA" values. Then the movies with the highest predicted ratings could be offered as suggestions for movies a user might like.

user	Movie 1	Movie 2	Movie 3
User 1	4	NA	2
User 2	NA	5	NA
User 3	NA	NA	3

1.3 Project Data (Step 1 in R file)

We start with the 10M MovieLens dataset available on the grouplens.org site maintained by the University of Minnesota. This database includes 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. It was released 1/2009.

The README.txt for this dataset is found here: http://files.grouplens.org/datasets/movielens/ml-10m-README.html

The data can be found at: https://grouplens.org/datasets/movielens/10m/

To create the data needed for our analysis, ml-10m.zip is downloaded and then formatted into a movielens dataframe with columns "userId", "movieId", "rating", "timestamp", "title", "genres". This data frame is then randomly split into a 90% training set (edx) and a 10% validation set (validation). The edx data set will be further split into a train_set and test_set in the model fitting phase.

The first time this code is run it will create two .Rds files to store the edx and validation datasets. That step takes a long time. If this code is re-run, it will look for the .Rds and simply read the datasets into memory.

```
# Create edx set, validation set (final hold-out test set)

# If previously run, we'll load the data from files
if (file.exists("movielens_training_data.rds") == TRUE){
    # Read in the previously processed and stored datasets
    edx <- readRDS("movielens_training_data.rds")
    validation <- readRDS("movielens_validation_data.rds")
# Otherwise the download process will run taking several minutes
} else {
    # Process the data
    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")

    # if using R 4.0 or later:
    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")

    ## Create edx (to work with and create our models) and validation (10% of data)
    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,]

    # Use semi_join() to remove users from temp (validation set) that are not in edx
    validation <- temp %>%
        semi_join(edx, by = "movieId") %>%
        semi_join(edx, by = "userId")
```

```
    anti_join(temp, validation, by = "movieId") # shows the rows that have been removed

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

    # Save edx and validation objects to a file to reference later
    saveRDS(edx, 'movielens_training_data.rds')
    saveRDS(validation, 'movielens_validation_data.rds')

    # removes elements used to create needed data
    rm(dl, ratings, movies, test_index, temp, movielens, removed)
}
```

---

2 Methods, Data Exploration & Model Fitting

2.1 Explore/visualize the data (Step 2 in R file)

In this section we'll look at some high level metrics associated with the data and then dive into an exploration of userId, timestamp, movieId, and genres.

There are 9000055 rows and 6 columns in the edx dataframe and 999999 rows and 6 columns in validation.

The source data is in tidy format with each row representing a rating assigned by a single user to a single movie.

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-F
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

There are six variables in the edx dataset:

- **edx$userId** - an integer vector with 69878 distinct values. There are 0 NA values in the vector
- **edx$movieId** - a numeric vector with 10677 distinct values. There are 0 NA values in the vector
- **edx$rating** - a numeric vector with 10 distinct values. There are 0 NA values in the vector. The values range from 0 to 5, but higher ratings are more frequently used.

- **edx$timestamp** - an integer vector showing the time the rating was assigned. The values are seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970
- **edx$title** - a character vector showing the movie titles. These are not necessarily unique so we should rely on movieId which is unique
- **edx$genres** - a character vector showing the various movie genres. There are 797 distinct values in the vector.

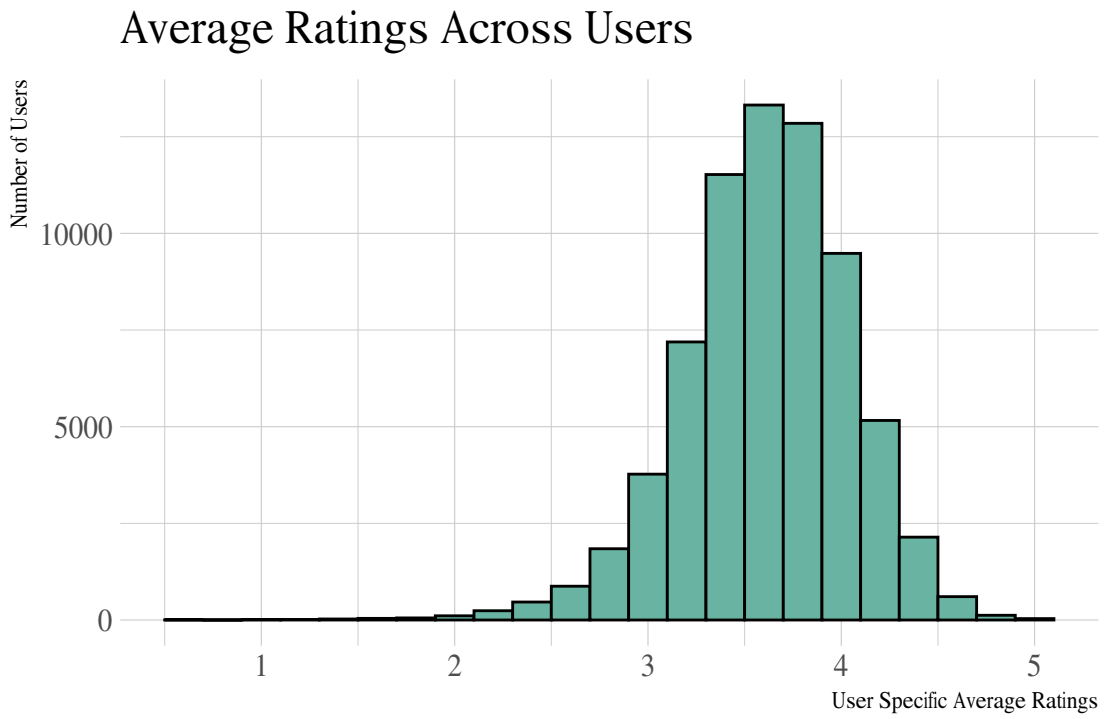
2.1.1 Users (Step 2a in R file)

Looking at the ratings from the perspective of the ~ 70,000 users, variations in the number of movies rated and the user specific average are clearly present. The overall average rating is 3.51, but the average of the user averages is 3.61. This difference is the result of the tendency for users who rate lots of movies to, on average, rate them lower.

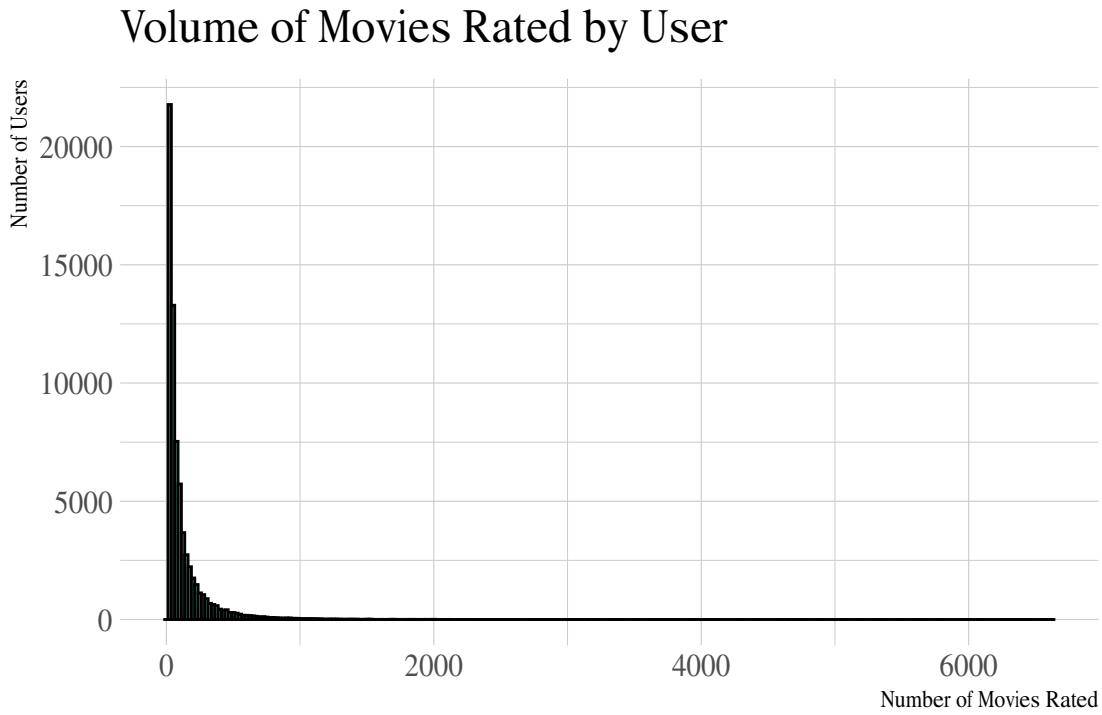
```
## [1] 3.613602
```

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

The user's average ratings appear normally distributed.

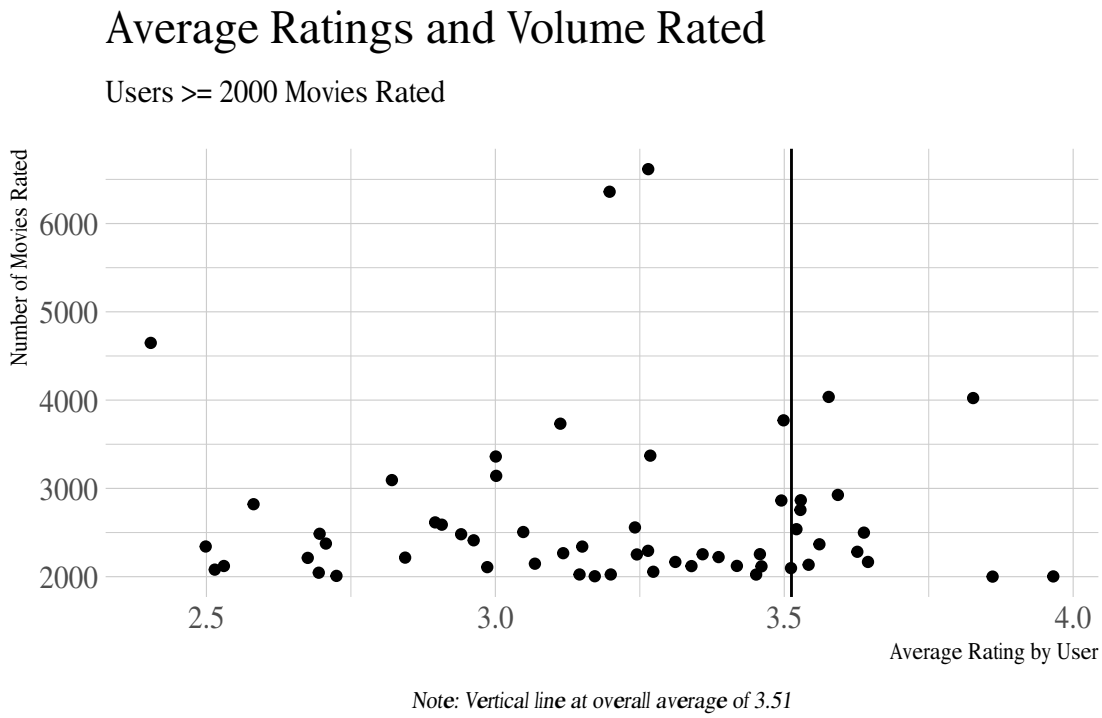


Most users rate relatively few movies, but there is a long tail of users who rate thousands of films.



We find that over 66% of users rate 100 or fewer movies, but the ratings associated with these users make up only 23% of the total ratings in the dataset.

Explore this further by looking at the averages for just those users with over 2000 ratings. They clearly are below the overall overage.



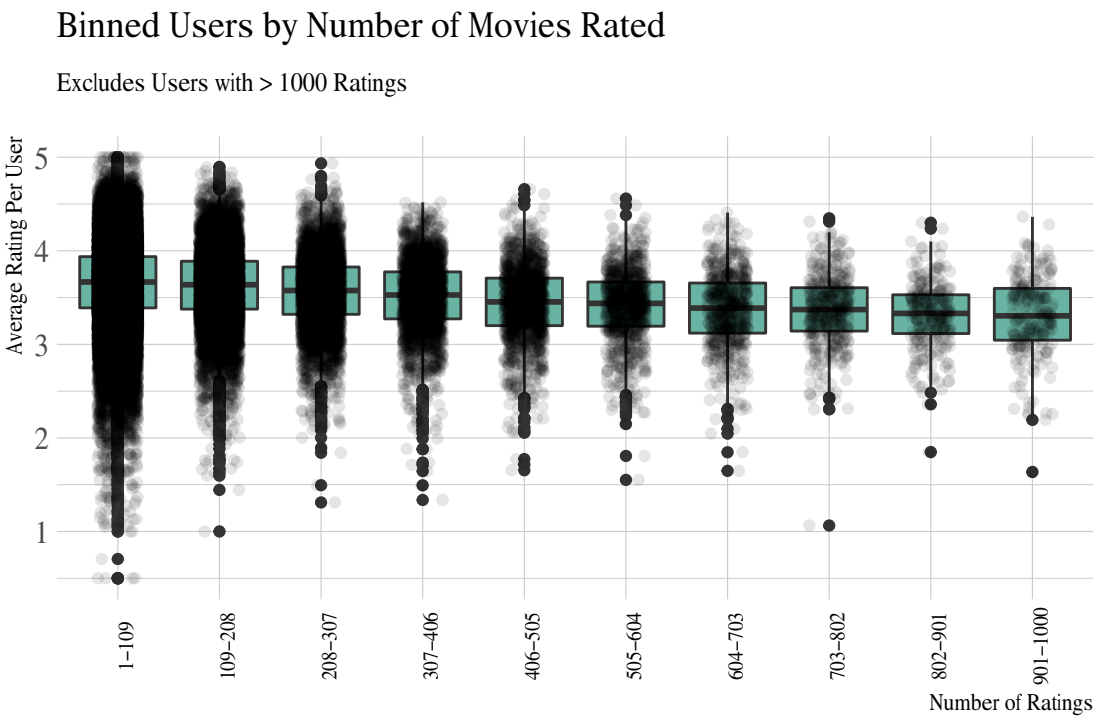
Note: Vertical line at overall average of 3.51

The averages for all users and those that rate 2000 or more movies are significantly different.

```
## [1] 3.194066
```

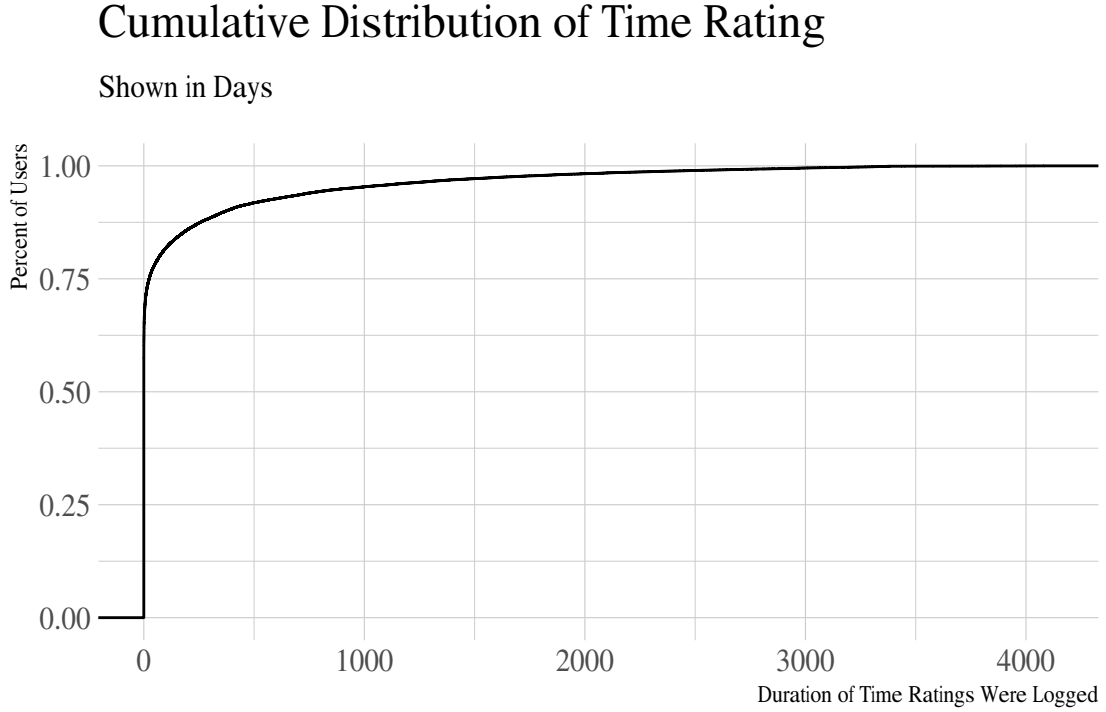
```
## [1] 3.613602
```

To visualize the differences between users with fewer ratings and those who've rated thousands we'll create bins of the data by number of ratings and create a boxplot for each bin. The plot shows that the largest population of users are those who only rate a few movies and these users tend to have higher ratings.



2.1.2 Timing (Step 2b in R file)

Interestingly, most users rate very few movies (as discussed in the prior section), and they rate them over a short period of time. The timestamp field allows us to define when each user rated their first movie and when they rated their last movie. From this, we can compute a duration or time rating and look at a cumulative distribution of that.



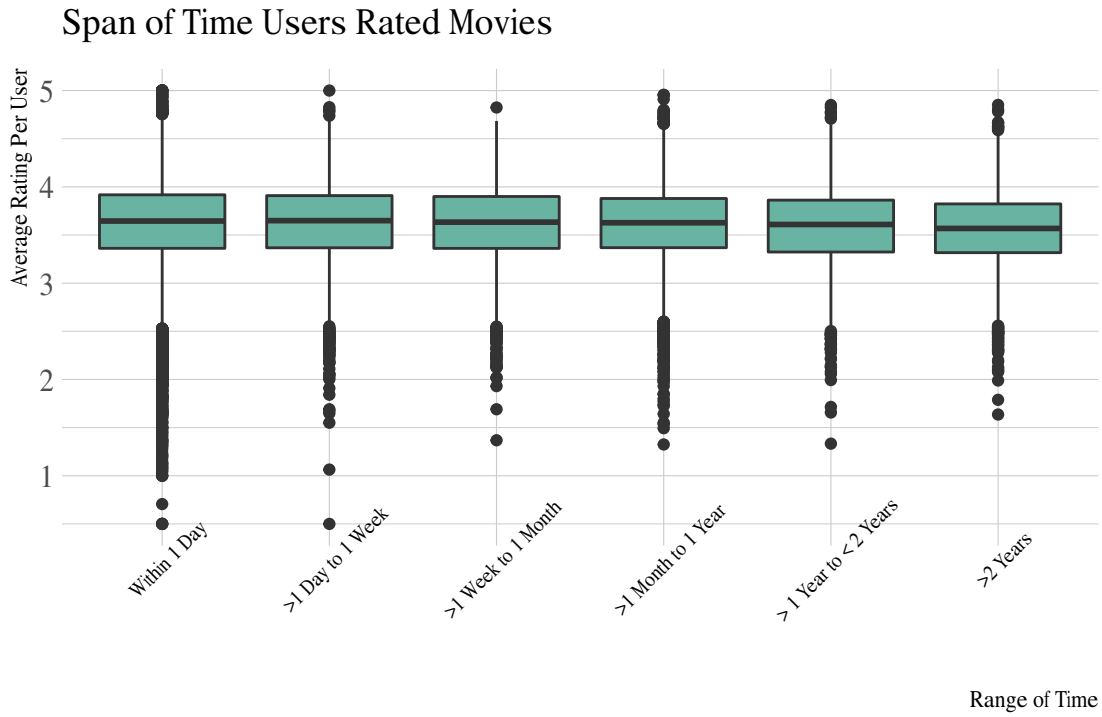
The chart above shows us that most users are rating over a very short period of time. In fact, over 63% are rating their movies over just 1 day ...

```
## [1] 0.6378116
```

... and 85% are rating in 180 days or less.

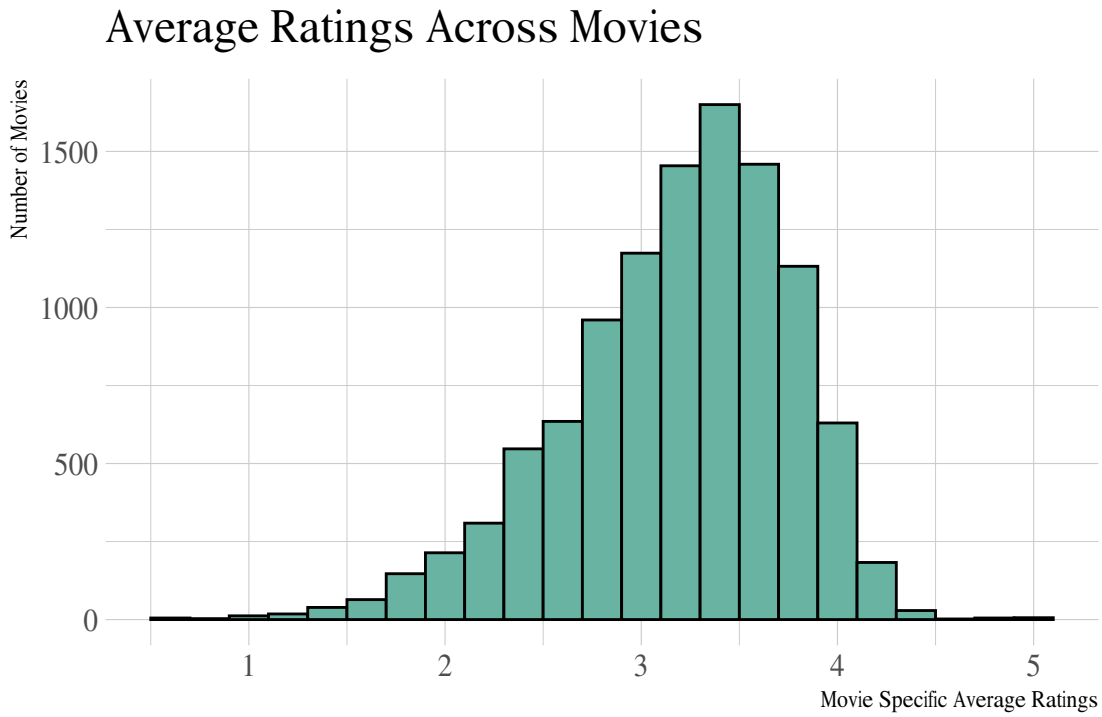
```
## [1] 0.8526575
```

However, even though most users are rating over short periods of time, that doesn't seem to impact their rating behavior. The chart below which shows the users in bins by the length of time over which they rated movies. The averages across the bins is consistent.



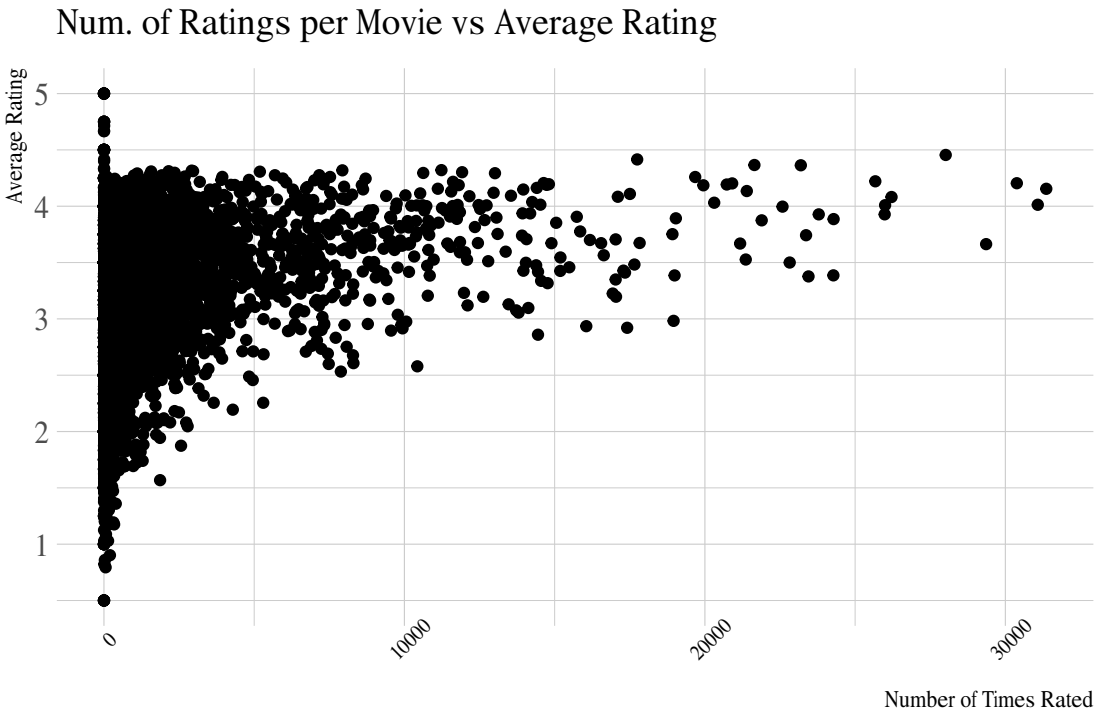
2.1.3 Movies (Step 2c in R file)

Across movies, the distribution of average ratings is left skewed.



The scatterplot of average ratings and volume of ratings (n) shows a positive relationship between

number of ratings and average ratings. It also shows a large number of movies with very few ratings



2.1.4 Genres (Step 2d in R file)

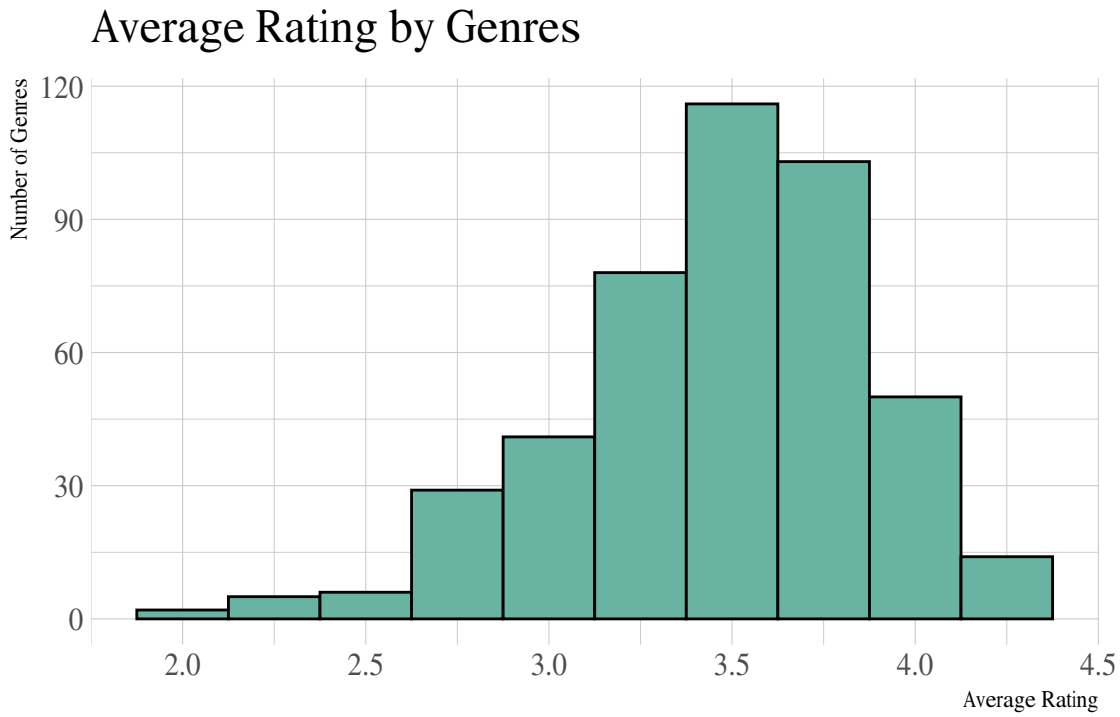
First, let's look at the most popular genres.

genres	n
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355
Action|Adventure|Sci-Fi	219938
Action|Adventure|Thriller	149091
Drama|Thriller	145373
Crime|Drama	137387

The genres assigned to each movie are combinations of specific categories separated by "|". We can create a list of just those.

	x
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

Explore the average rating per genre.



2.2 Fitting the model (Step 3 in R file)

Now that we've explored the data, we're ready to fit a model to try to successfully predict how users will rate movies. We want to avoid using the validation set until the very end so we'll further split edx into a test_set and ta train_set.

```
set.seed(755)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Remove these entries that appear in both, using the 'semi_join' function:
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

To measure the effectiveness of our model, we'll use a loss function that measures the differences between predicted values and actual values in the test_set. This measure is called the residual (or root) mean squared error or RMSE. An RMSE of 1 implies that our predictions are a full point away, on average, from the actual values. We'll define success as an RMSE below 0.86490.

We define $y_{u,i}$ as the rating for movie i by user u. The prediction is $\hat{y}_{u,i}$. The RMSE is then defined as shown in the formula below with N being the number of user/movie combinations and the sum occurring over all these combinations:

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

For the analysis, we'll need to define a function to compute the RMSE and a table to store the results. Add the target RMSE as the first entry in the table.

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

# Create a table to store the results and add the Target we're aiming to be below
target <-  0.86490
rmse_results <-
  tibble("Method" = "Target", RMSE = target, "Diff. from Target" = RMSE - target)
```

2.2.1 Explore simple approaches (Step 3a in R file)

In this section we'll look at several simple approaches to predicting how users will rate a movie. The first is to use just the simple average then will incorporate user and movie bias. The results for these simple approaches are surprisingly good when compared to our target RMSE.

Define a simple model just using the overall average rating and name it "Just the average". In this model every movie is rated at the overall average by every user.

```
# define the overall average and store in a variable, mu
mu <- mean(train_set$rating)

# compute the RMSE (loss function) for the ratings in our test_set
just_the_average <- RMSE(test_set$rating, mu)
just_the_average
```

```
## [1] 1.06053
```

```
# Add the results to the table
rmse_results <- bind_rows(rmse_results,
   tibble("Method" = "Just the Average", RMSE = just_the_average, "Diff. from Target" = RMSE - target
kbl(rmse_results, booktabs = T) %>% kable_styling(latex_options = "striped")
```

Method	RMSE	Diff. from Target
Target	0.86490	0.0000000
Just the Average	1.06053	0.1956303

Next, we'll add the movie effects recognizing that some movies are rated higher on average than others which we saw in the data exploration work shown previously.

```
# Store the difference between each movie's average rating and the overall average
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Compute the RMSE for adding the effect of the average rating for movieId
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.9440004
```

```
# Add these results to the table
movie_effects <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
      tibble("Method" = "Movie Effect Model",
             RMSE = movie_effects,
             "Diff. from Target" = RMSE - target))
kbl(rmse_results, booktabs = T) %>% kable_styling(latex_options = "striped")
```

Method	RMSE	Diff. from Target
Target	0.8649000	0.0000000
Just the Average	1.0605303	0.1956303
Movie Effect Model	0.9440004	0.0791004

We should also take into account individual user bias because we know they rate movies differently.

```
# Store the difference in the overall avg rating for a user minus the overall average
# average rating across all users minus the variability associated with a particular movie
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>% # adds b_i to the train_set
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Create predicted ratings using the user effect, movie effect and overall average
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# Reset ratings which are >5 to 5 as that is the highest possible rating
predicted_ratings <- ifelse(predicted_ratings > 5, 5, predicted_ratings)

# Compute the RMSE of this new model with both user and movie effects and store
user_bias <- RMSE(predicted_ratings, test_set$rating)

# Add the results to our table
rmse_results <- bind_rows(rmse_results,
   tibble("Method" = "Movie and User Effects",
          RMSE = user_bias,
          "Diff. from Target" = RMSE - target))
kbl(rmse_results, booktabs = T) %>% kable_styling(latex_options = "striped")
```

Method	RMSE	Diff. from Target
Target	0.8649000	0.0000000
Just the Average	1.0605303	0.1956303
Movie Effect Model	0.9440004	0.0791004
Movie and User Effects	0.8665457	0.0016457

2.2.2 Using regularization (Step 3b in R file)

One of the challenges with our data is that some movies have only a few ratings. These are likely skewing our predictions and over fitting the model. Regularization is a way to penalize estimates that are based on small sample sizes. The general idea behind regularization is to constrain the total variability of the effect sizes. Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$$\sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

Source: Introduction to Data Science, Irizarry (page 645)

When the sample size n_i is very large then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0. The larger λ, the more the estimate shrinks toward the mean.

When we look at the best and worst movies and how frequently they are rated, something looks amiss. These are all obscure movies with only a few ratings.

```
# The highest and lowest rated movies from the last model are a bunch of random movies
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()

# best movies
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  select(title, n) %>%
  kbl(., booktabs = T) %>% kable_styling(latex_options = "striped")
```

title	n
Hellhounds on My Trail (1999)	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	2
Shanghai Express (1932)	1
Satan's Tango (Sátántangó) (1994)	2
Appaloosa, The (1966)	1
Shadows of Forgotten Ancestors (1964)	1
Fighting Elegy (Kenka erejii) (1966)	1
Sun Alley (Sonnenallee) (1999)	1
Class, The (Entre les Murs) (2008)	2
Blue Light, The (Das Blaue Licht) (1932)	1

```
# worst movies
train_set %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  select(title, n) %>%
  kbl(., booktabs = T) %>% kable_styling(latex_options = "striped")
```

```
## Joining, by = "movieId"
```

title	n
30 Years to Life (2001)	1
Besotted (2001)	1
Blindness (2003)	1
Accused (Anklaget) (2005)	1
Confessions of a Superhero (2007)	1
War of the Worlds 2: The Next Wave (2008)	2
SuperBabies: Baby Geniuses 2 (2004)	47
Disaster Movie (2008)	22
From Justin to Kelly (2003)	144
Hip Hop Witch, Da (2000)	10

To address the problem of users and movies with low frequency in the data, we will use regularization. The best lambda, which creates the lowest RMSE, was generated using a cross-validation approach which is shown here, but only for illustrative purposes as it takes a few minutes to run.

```
# Define a set of lambdas to test
lambdas <- seq(0, 10, 0.25)

# store the results in rmses
rmses <- sapply(lambdas, function(l){

  # the overall average
  mu <- mean(train_set$rating)

  # movie effect scaled by lambda
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # user effect scaled by lambda
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  # compute the predicted ratings on test_set
  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))
})

# define the best lambda that minimizes RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

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

We'll generate predicted ratings using the best lambda defined above, compute the RMSE, and add that to our results table.

```
# Compute the predicted ratings using the final optimized lambda
mu <- mean(train_set$rating)

# movie effect scaled by lamdba
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# user effect scaled by lambda
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# compute the predicted ratings on test_set
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)

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

Let's re-look at the best and worst movies. These now make more sense. The best movies are familiar and seem appropriate to be at the top of the list.

title	n
Shawshank Redemption, The (1994)	22425
Godfather, The (1972)	14209
Usual Suspects, The (1995)	17302
Schindler's List (1993)	18630
Third Man, The (1949)	2380
Casablanca (1942)	8952
Rear Window (1954)	6398
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	2329
Seven Samurai (Shichinin no samurai) (1954)	4205
Godfather: Part II, The (1974)	9599

title	n
From Justin to Kelly (2003)	144
SuperBabies: Baby Geniuses 2 (2004)	47
Pokémon Heroes (2003)	111
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	159
Glitter (2001)	274
Barney's Great Adventure (1998)	171
Gigli (2003)	266
Carnosaur 3: Primal Species (1996)	56
Disaster Movie (2008)	22
House of the Dead, The (2003)	172

Add the results to our table.

Method	RMSE	Diff. from Target
Target	0.8649000	0.0000000
Just the Average	1.0605303	0.1956303
Movie Effect Model	0.9440004	0.0791004
Movie and User Effects	0.8665457	0.0016457
Regularized Movie + User Effect Model	0.8660397	0.0011397

2.2.3 Matrix factorization using recosystem (Step 3c in R file)

Matrix factorization (or collaborative filtering) is a powerful tool in developing recommendation systems. The recosystem library is purpose built to generate predicted values using this technique. To learn more about this tool see https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html written by one of the authors of the library.

The recosystem package requires the data to be in a specific matrix format. Each dataset will have three columns - user, item (movie), and rating. We'll use our training and test data to create the data structures needed for recosystem.

```
set.seed(123) # This is a randomized algorithm

# create the datasets in the right format for recosystem
# set index1 = TRUE as our values start with 1 (not 0)
train_set_new =
  data_memory(train_set$userId, train_set$movieId, train_set$rating, index1 = TRUE)
test_set_new =
  data_memory(test_set$userId, test_set$movieId, test_set$rating, index1 = TRUE)

# create the model object
r = Reco()
```

Next we need to find the best tuning parameters. This step can take ~ 30 minutes or so depending upon the power of your machine.

```
opts = r$tune(train_set_new, opts = list(
  dim = c(10, 20, 30), # number of latent factors
  lrate = c(0.1, 0.2), # learning rate, or the step size in gradient descent
  costp_l1 = 0, # L1 regularization cost for user factors
  costq_l1 = 0, # L1 regularization cost for item factors
  nthread = 1, # number of threads for parallel computing
  niter = 10 # number of iterations
  ))
opts
```

With the optimized tuning parameters, we need to train the model and then use it to predict our ratings.

```
# train the model using the best tuning parameters
r$train(train_set_new, opts = c(opts$min, nthread = 1, niter = 20))

# calculate the predicted values
predicted_ratings = r$predict(test_set_new, out_memory())

# Reset ratings which are >5 to 5 as that is the highest possible rating
predicted_ratings <- ifelse(predicted_ratings > 5, 5, predicted_ratings)
```

Finally, compute the RMSE and add the results to our table.

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

rmse_results <- bind_rows(rmse_results,
    tibble("Method" = "Matrix Factorization (recosystem)",
      RMSE = model_matrix_factorization,
      "Diff. from Target" = RMSE - target))
kbl(rmse_results, booktabs = T) %>% kable_styling(latex_options = "striped")
```

Method	RMSE	Diff. from Target
Target	0.8649000	0.0000000
Just the Average	1.0605303	0.1956303
Movie Effect Model	0.9440004	0.0791004
Movie and User Effects	0.8665457	0.0016457
Regularized Movie + User Effect Model	0.8660397	0.0011397
Matrix Factorization (recosystem)	0.7916594	-0.0732406

---

3 Results & Final Model Assessment (Step 4 in R file)

Now that we have a model we think will beat our target RMSE, let's test it using the original holdout set, validation. First, we'll need to create the same three column dataset as we did with our train_set and test_set above.

```
validation_set_new = data_memory(validation$userId, validation$movieId, index1 = TRUE)
```

Once we have the data in the right format we can use the model (r) to predict the values.

```
# compute the predicted ratings
predicted_ratings = r$predict(validation_set_new, out_memory())

# remove any ratings > 5
predicted_ratings <- ifelse(predicted_ratings > 5, 5, predicted_ratings)
```

Finally, we'll add it to the table.

```
model_validation <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
  tibble("Method"="Validation Matrix Factorization (recosystem)",
  RMSE = model_validation, "Diff. from Target" = RMSE - target))
kbl(rmse_results, booktabs = T) %>% kable_styling(latex_options = "striped")
```

Method	RMSE	Diff. from Target
Target	0.8649000	0.0000000
Just the Average	1.0605303	0.1956303
Movie Effect Model	0.9440004	0.0791004
Movie and User Effects	0.8665457	0.0016457
Regularized Movie + User Effect Model	0.8660397	0.0011397
Matrix Factorization (recosystem)	0.7916594	-0.0732406
Validation Matrix Factorization (recosystem)	0.7906998	-0.0742002

---

4 Conclusion

In this analysis, we've created several models to predict movie ratings. In total we created 5 models - three simple approaches just using the overall average, the movie specific bias, and user and movie specific bias. We then applied regularization to incorporate the information we know about some users rating only a few movies and some movies having only a few ratings. In regularization we create a scaling factor to lessen the impact on our fitted model of these low-frequency observations. Finally, we built a model based on matrix factorization using the recosystem package. This model far exceeded the performance of our other efforts and the target RMSE of .0.86490.

Could we do better? We're on average .79 points from the correct rating. This means that a movie I'd rate as a 4 might be considered a 3.21 by the model. That feels like a pretty big difference. Given the data available to Netflix and Amazon, surely they could do better. There may be more we could do with the existing dataset such as incorporating genres. It would seem, however, that any large streaming service such as Netflix, HBO, or Amazon has user information available to them because users login to use the service and what those users do once they've logged in is a treasure-trove of information. Do they start some movies and stop? Do they watch some movies more than once? Do they gravitate towards one section over another? To build a more robust, and ultimately accurate model, we'd want to incorporate some of this additional data.

Resources

In the course of putting together this report and building the underlying R script I consulted many online resources. The most important are listed here.

1. Irizarry, Rafael, A., "Introduction to Data Science", Published: October 24, 2019, Accessed: February 2021, https://rafalab.github.io/dsbook/ (A comprehensive guide to Data Science and R.)

2. Holtz, Yan, "The R Graph Gallery, ggplot2 boxplot from continuous variable", Published: 2018, Accessed: February 2021, https://www.r-graph-gallery.com/268-ggplot2-boxplot-from-continuous-variable.html (A complete ggplot resource.)

3. Wickham, Hadley, "Advanced R, Subsetting", Published: 2019, Accessed: February 2021, http://adv-r.had.co.nz/Subsetting.html (A review of subsetting basics.)

4. Qiu, Yixuan et al, "Package 'recosystem' ", Published: January 10, 2021, Accessed: February 2021, https://cran.r-project.org/web/packages/recosystem/recosystem.pdf (Documentation for the recosystem package.)

5. Qiu, Yixuan, "recosystem: recommender system using parallel matrix factorization", Published: July 15, 2016, Accessed: February 2021, https://statr.me/2016/07/recommender-system-using-parallel-matrix-factorization/ (A blog by the recosystem library author with an explanation of matrix factorization.)

6. Qiu, Yixuan, "recosystem: Recommender System Using Parallel Matrix Factorization", Published: January 9, 2021, Accessed: February 2021, https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html (A later blog by the recosystem library author with an explanation of matrix factorization.)

7. Ivamoto, Victor, "Movie Recommendation System in R", Published: September 12, 2019, Accessed: February 2021, https://rpubs.com/vsi/movielens (A prior submission that also leveraged the recosystem library.)

8. Qiu, Yizuan, "In recosystem: Recommender System using Matrix Factorization, An explanation of tuning parameters, tune: Tuning Model Parameters", Published: January 10, 2021, Accessed: February 2021, https://rdrr.io/cran/recosystem/man/tune.html (A detailed explanation of the tuning parameters.)

9. Asim Ansari, "Matrix Factorization for Recommender Systems", Accessed: February 2021, https://courses.edx.org/asset-v1:ColumbiaX+BAMM.104x+1T2020+type@asset+block/recommender1.nb.html# (An implementation example of recosystem from another edX course.)

10. Pandey, Parul, "Recommendation Systems in the Real world, An overview of the process of designing and building a recommendation system pipeline.", Published: March 17, 2019, Accessed: February 2021, https://towardsdatascience.com/recommendation-systems-in-the-real-world-51e3948772f3 (An overview of recommendation systems.)

11. Shetty, Badreesh, "AN IN-DEPTH GUIDE TO HOW RECOMMENDER SYSTEMS WORK, You might also like ... to know how programs know what you want before you do", Published: July 24, 2019, Accessed, February 2021, https://builtin.com/data-science/recommender-systems (An overview of recommendation systems.)

12. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872 (The Movielens Dataset)