Book Prediction Capstone Project

Steffen Parratt

3/4/2019

Introduction

This report describes the approach and results of my Independent Capstone project for the HarvardX course PH125.9x. My project submission includes three files, all with the same file name, but in the following three formats: PDF, RMD and R. The computer code in the R file is identical to the RMD file.

There are four major sections to this report:

- Introduction
- Methods & Analysis
- Results
- Conclusions

Goal

The goal of this project is to create a book recommendation system using the goodbooks-10k data set. In addition to producing a thoughtful and readable report, a metric for success is producing a system with a reported root mean square error (RMSE) of less than or equal to 0.87750 for the validation data set, which was the target provided in our MovieLens project. This target is consistent with other Kaggle analyses of this data set that I reviewed.

Key Steps

This section provides a high-level overview of the key steps completed in this project. These steps are described in detail in the sections below. In addition, the R code provided with this report is also organized in this same format. The following are the key steps:

- R Packages Installation
- Data Gathering
- Data Exploration and Cleaning
- Data Preparation
- Model Selection
- Parameter Optimization
- Model Training and Evaluation
- Results
- Conclusions

R Package Installation checks to see if the user has all of the necessary packages, and, if not, they are downloaded from CRAN.

Data Gathering is downloading and unzipping four data files from Github.

Data Exploration & Cleaning describes through exploration and visualization the parameters that I considered adding to my model. It also describes all of the checks I made on the data to correct any errors and to remove any fields that I considered superfluous.

Data Preparation is the process of setting up the data to be analyzed. The first step was to separate and set aside 10% of the data to be the validation data set. The second step was to split the remaining 90% of the data into a training set (80%) and testing set (20%), similar to how we prepared the data in the MovieLens project.

Model Selection describes the activity of reviewing and selecting an approach to modeling the problem and deciding which parameters to include in the model. In this project we use the modeling approach described in section 35.4 of our course textbook.

Parameter Optimization is the activity of optimizing the parameter values that have been included in the model. In this project, the user and book effect parameters were optimized via **regularization**, which is described in section 35.5 of our textbook.

Model Training and Evaluation is the activity of training the chosen model and parameters on the training data set, and then applying the trained model to the validation data set and reporting the RMSE score.

Results simply summarizes the RMSE of the models built and the RMSE of the validation data set.

Conclusions describes some lessons learned in this project.

Methods & Analysis

Here we go into detail on each key step of the data analysis project. The code and output shown below is contained in the RMD file and executed with Knit.

R Package Installation

As always, we begin by checking if we have all of the necessary R packages installed for the project. If not, they are downloaded and installed.

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(utils)) install.packages("utils", repos = "http://cran.us.r-project.org")
```

Data Gathering

In this project the original data was provided through Kaggle, which I downloaded and analyzed. Then I read there was a corrected and expanded data set that was available directly through Github.

There are several kernels that are posted on Kaggle that address this data set. The one by Philipp Spachtholz is terrific, and I have borrowed ideas, observations and code snippets from his notebook. Philipp uses RecommenderLab for his modeling, while I have used the approach we covered in class. Philipp's RMSE results are slightly better than mine, although we are using different data sets. (Since Philipp's notebook was published, a corrected and greatly expanded data set was published on Github. Philipp's notebook does not analyze this updated data set.)

In this section of the code I download and unzipped the four relevant data sets.

```
dl <- tempfile()

download.file("https://github.com/zygmuntz/goodbooks-10k/releases/download/v1.0/books.zip", dl)
books <- read_csv(unzip(dl))

download.file("https://github.com/zygmuntz/goodbooks-10k/releases/download/v1.0/ratings.zip",dl)
ratings <- read_csv(unzip(dl))

download.file("https://github.com/zygmuntz/goodbooks-10k/releases/download/v1.0/book_tags.zip", dl)
book_tags <- read_csv(unzip(dl))

download.file("https://github.com/zygmuntz/goodbooks-10k/releases/download/v1.0/tags.zip", dl)
tags <- read_csv(unzip(dl))

file.remove(dl)
```

```
## [1] TRUE
```

Data Exploration and Cleaning

In the previous section we downloaded four files: books.csv, ratings.csv, book_tags.csv and tags.csv. In this section we explore each file, see what data it contains, decide what might be useful, and what we should discard. We also run checks on the data and do any necessary cleaning.

We begin with the ratings.csv file, which is the core data set for the project. This file constains user ratings for all 10,000 books listed. Each row of the file contains a book_id, user_id and rating between 1 and 5.

```
    head(ratings)
```

```
## # A tibble: 6 x 3
##   user_id book_id rating
##     <dbl>   <dbl>  <dbl>
## 1       1     258      5
## 2       2    4081      4
## 3       2     260      5
## 4       2    9296      5
## 5       2    2318      3
## 6       2      26      4
```

The file has almost 6 million ratings by over 53,000 users for 10,000 books.

```
    cat("Number of ratings =", length(ratings$rating))
```

```
## Number of ratings = 5976479
```

```
    cat("Number of users =", length(unique(ratings$user_id)))
```

```
## Number of users = 53424
```

```
    cat("Number of books =", length(unique(ratings$book_id)))
```

```
## Number of books = 10000
```

First we will filter out any NA fields in this data set.

```
    ratings <- ratings %>% filter(!is.na(book_id) |
                                  !is.na(user_id) |
                                  !is.na(rating))
```

Then we check to ensure there are no duplicate entries were removed. In this case, the same rating given to the same book by the same user. (This was a problem in the old data set.)

```
    duplicates <- ratings %>% group_by(user_id, book_id, rating) %>%
      mutate(n=n()) %>% filter(n > 1)
    duplicates
```

```
## # A tibble: 0 x 4
## # Groups:   user_id, book_id, rating [0]
## # ... with 4 variables: user_id <dbl>, book_id <dbl>, rating <dbl>,
## #   n <int>
```

Now let's look for a similar case: a user giving multiple different ratings for the same book. (This was also an issue with the previous data set.)

```
    duplicates <- ratings %>% group_by(user_id, book_id) %>%
      mutate(n=n()) %>% filter(n > 1)
    duplicates
```

```
## # A tibble: 0 x 4
## # Groups:   user_id, book_id [0]
## # ... with 4 variables: user_id <dbl>, book_id <dbl>, rating <dbl>,
## #   n <int>
```

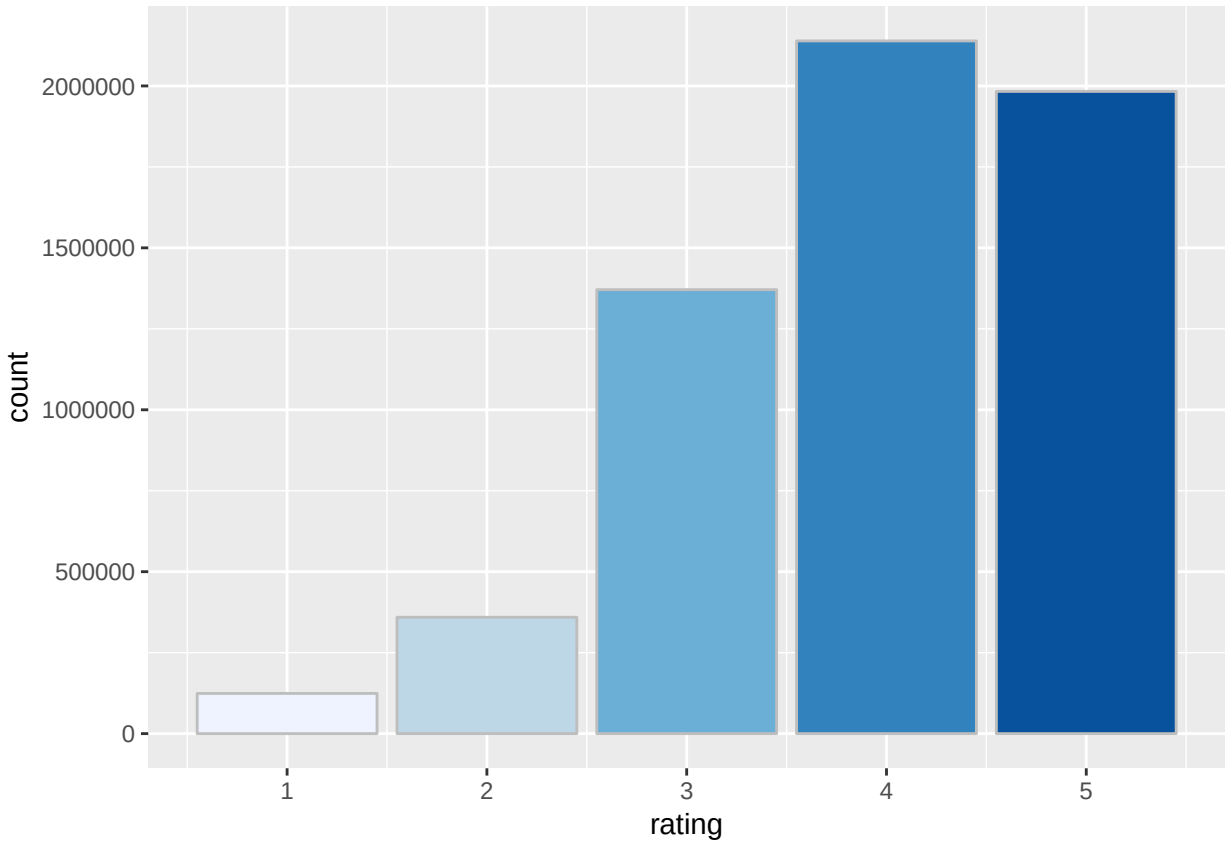
We also check that all ratings are between 1 and 5.

```
    rating_errors <- ratings %>% filter(rating < 1 | rating > 5)
    rating_errors
```

```
## # A tibble: 0 x 3
## # ... with 3 variables: user_id <dbl>, book_id <dbl>, rating <dbl>
```

Now let's take a look at the distribution of ratings...

```
    ratings %>%
      ggplot(aes(x = rating, fill = factor(rating))) +
      geom_bar(color = "grey") +
      scale_fill_brewer(palette = "Blues") +
      guides(fill = FALSE)
```



As you can see from the plot, the median rating is 4.

```
    median(ratings$rating)
```

```
## [1] 4
```

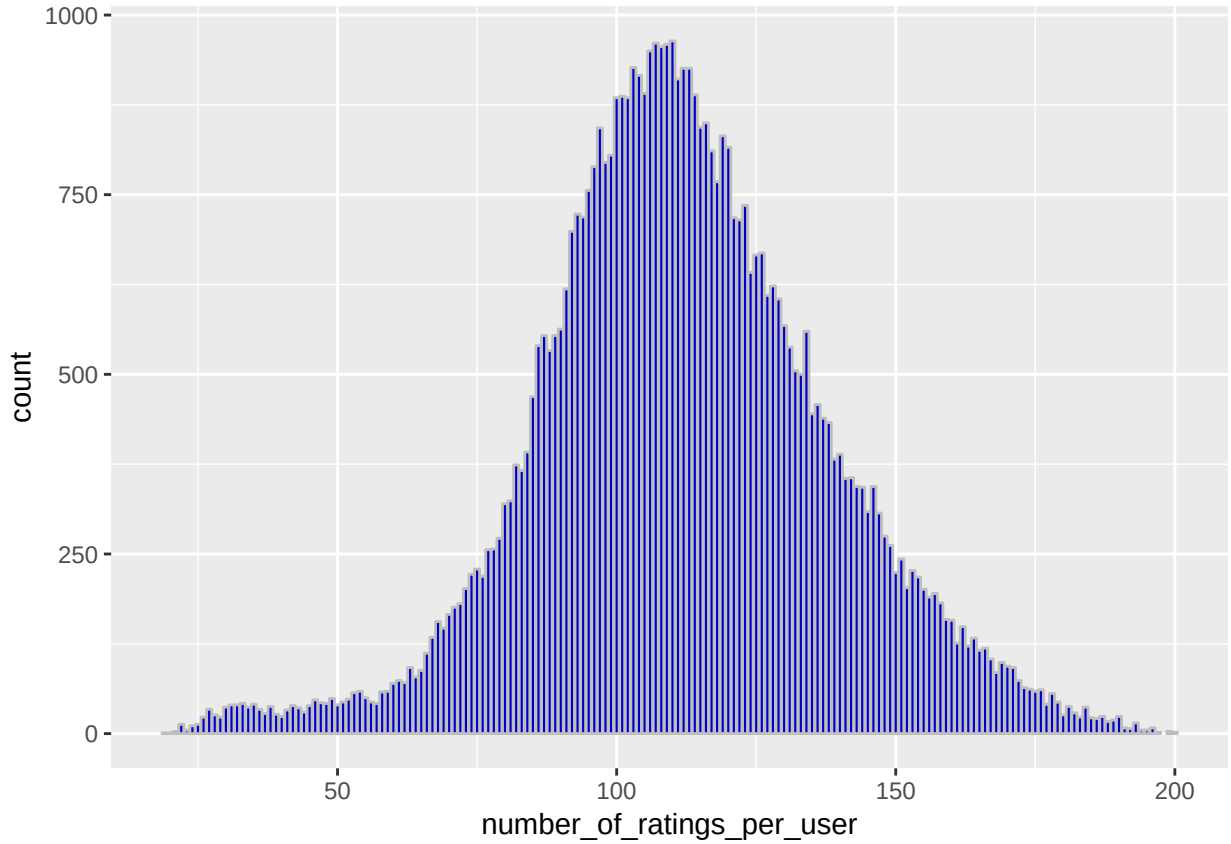
And the mean is 3.9

```
    mean(ratings$rating)
```

```
## [1] 3.919866
```

Looking at the number of ratings per user, we can see that it ranges from about 20 to 200.

```
    ratings %>%
      group_by(user_id) %>%
      summarize(number_of_ratings_per_user = n()) %>%
      ggplot(aes(number_of_ratings_per_user)) +
      geom_bar(fill = "mediumblue", color = "grey")
```



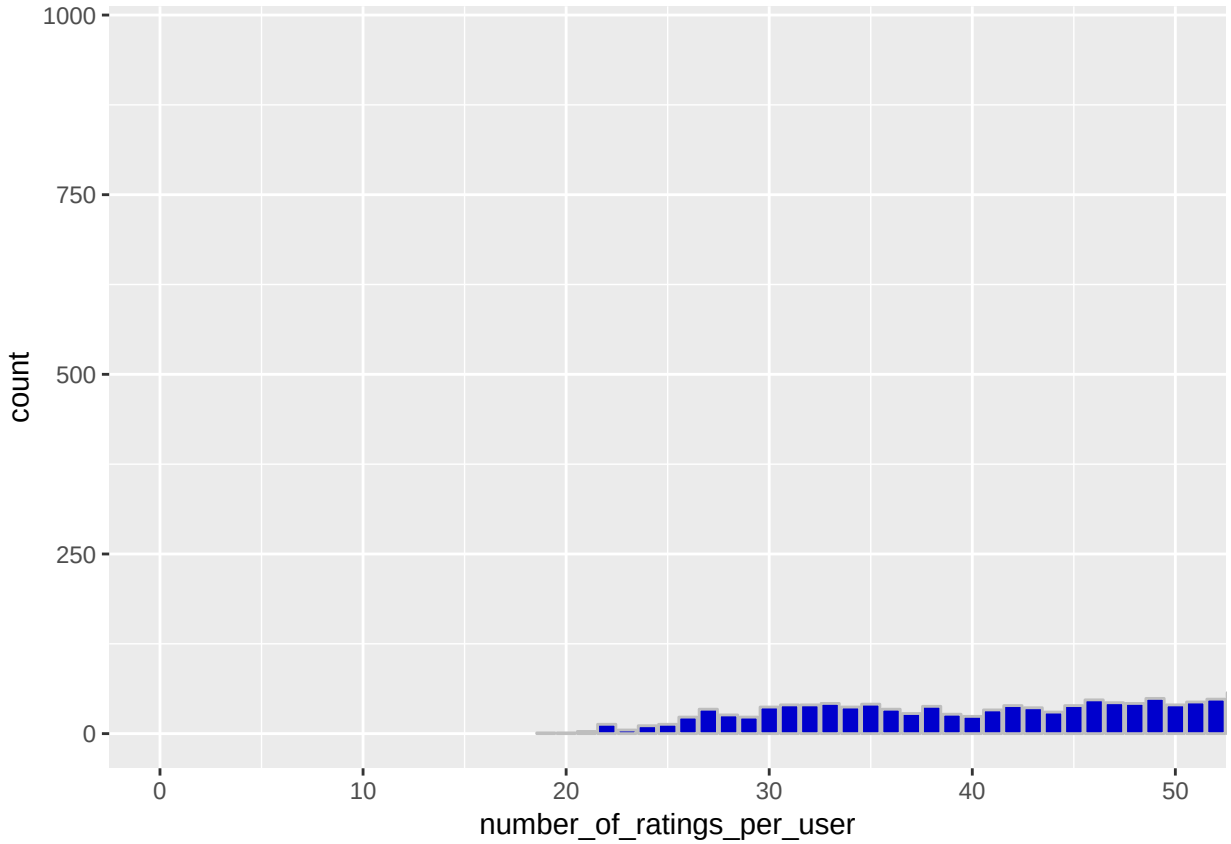
The median number is 111 reviews per user.

```
user_habits <- ratings %>% group_by(user_id) %>% summarize(number_of_ratings_per_user = n())
median(user_habits$number_of_ratings_per_user)
```

```
## [1] 111
```

It is doubtful how useful it is to include the preferences of a user who has rated few books. Let's zoom into the low end of the chart and determine where we might set a cutoff.

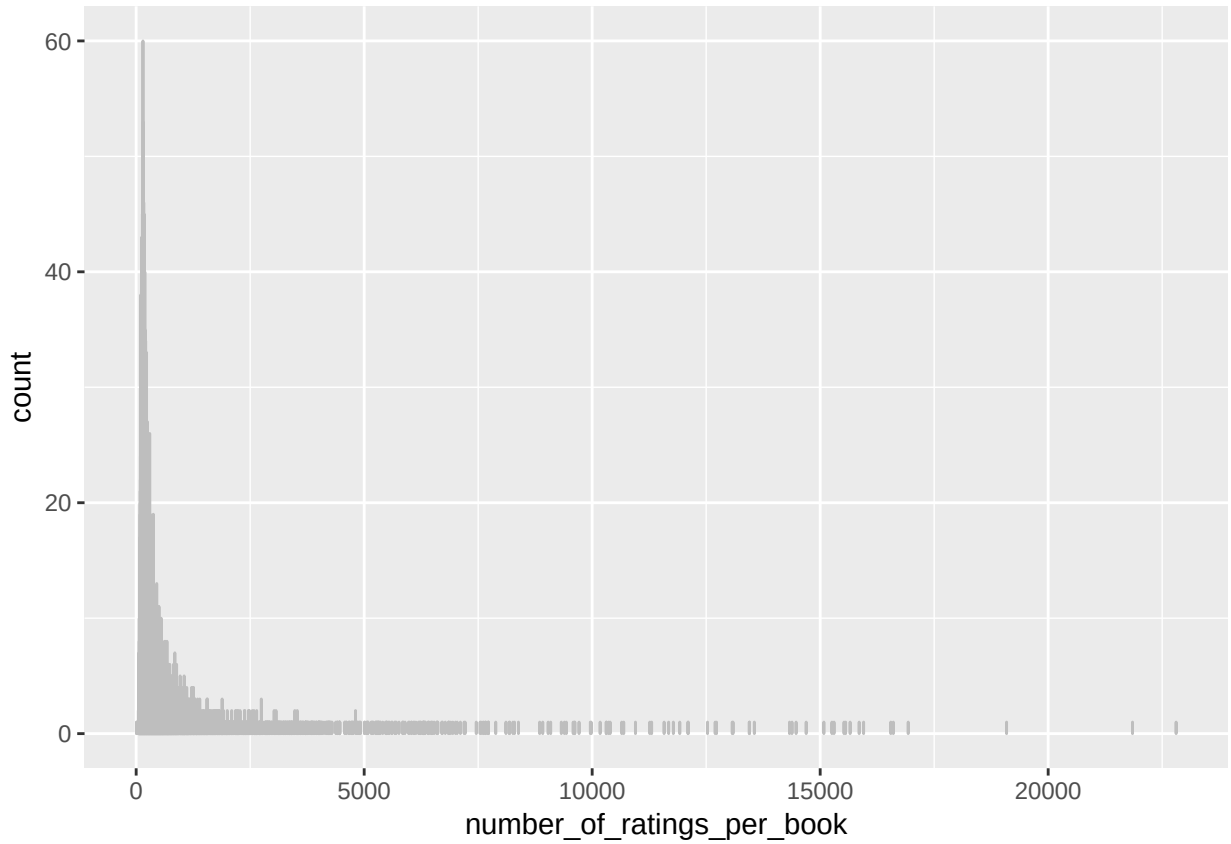
```
ratings %>%
  group_by(user_id) %>%
  summarize(number_of_ratings_per_user = n()) %>%
  ggplot(aes(number_of_ratings_per_user)) +
  geom_bar(fill = "mediumblue", color = "grey") + coord_cartesian(c(0, 50))
```



We can see that each user has rated at least 20 books, which seems like a sufficient number to be include in our analysis.

Similarly, let's look at the number of ratings per book.

```
ratings %>%
  group_by(book_id) %>%
  summarize(number_of_ratings_per_book = n()) %>%
  ggplot(aes(number_of_ratings_per_book)) +
  geom_bar(fill = "blue", color = "grey", width = 1)
```




```
    ratings_per_book <- ratings %>% group_by(book_id) %>%
      summarize(number_of_ratings_per_book = n())
```

The median ratings per book is 248, the mean is about 600.

```
    cat("Median rating =",median(ratings_per_book$number_of_ratings_per_book))
```

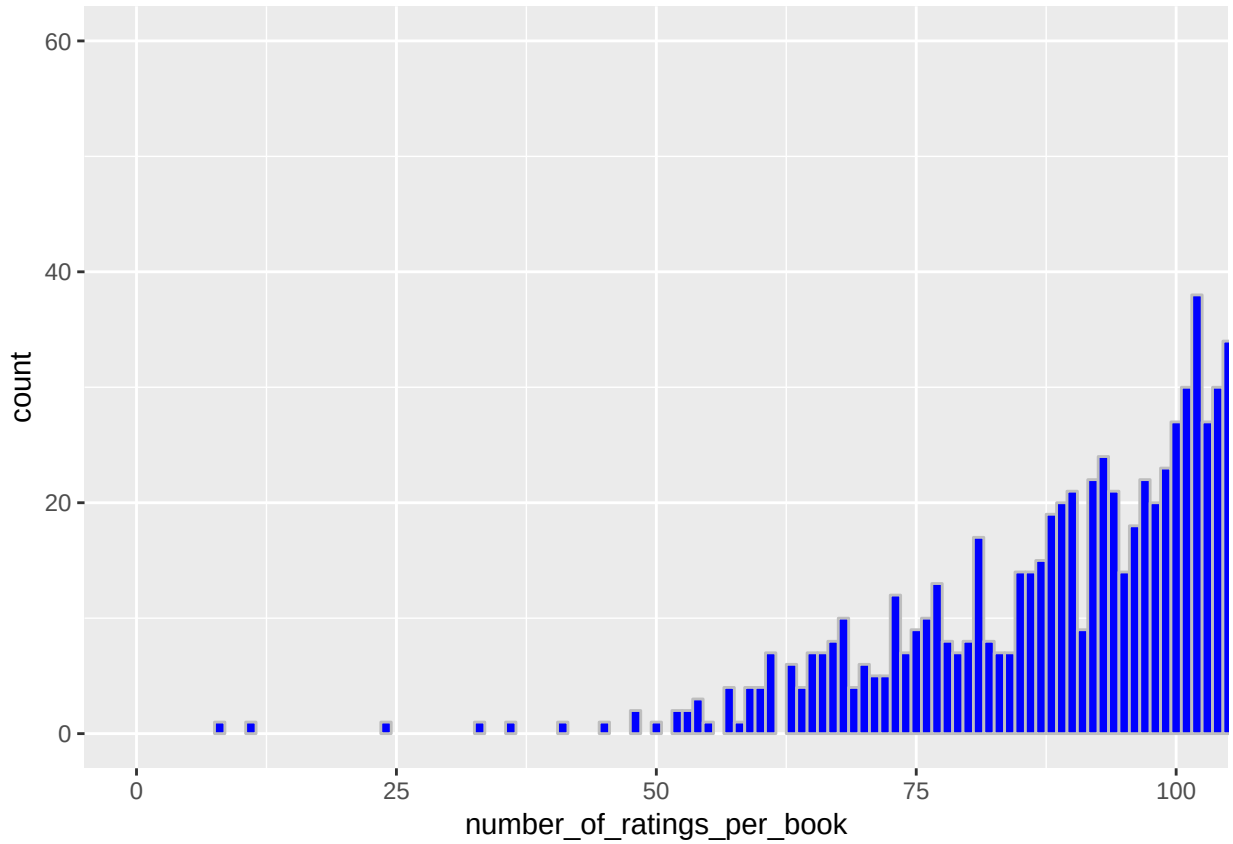
```
## Median rating = 248
```

```
    cat("Mean rating =", mean(ratings_per_book$number_of_ratings_per_book))
```

```
## Mean rating = 597.6479
```

Let's zoom in to the low end of the range to see if any books have too few ratings to be meaningful.

```
    ratings %>%
      group_by(book_id) %>%
      summarize(number_of_ratings_per_book = n()) %>%
      ggplot(aes(number_of_ratings_per_book)) +
      geom_bar(fill = "blue", color = "grey", width = 1) + coord_cartesian(c(0, 100))
```



And we see that there is only one book with fewer than 10 ratings.

```
few_ratings_books <- ratings_per_book %>% filter( number_of_ratings_per_book < 10 )
few_ratings_books
```

```
## # A tibble: 1 x 2
##   book_id number_of_ratings_per_book
##     <dbl>                      <int>
## 1    7803                          8
```

At this point we have a clean data set from the ratings.csv file, which we now understand pretty well. Let's now move on to the tags.csv file.

The tags.csv file contains tag_names and tag_ids.

```
head(tags)
```

```
## # A tibble: 6 x 2
##   tag_id tag_name
##    <dbl> <chr>
## 1      0 -
## 2      1 --1-
## 3      2 --10-
## 4      3 --12-
## 5      4 --122-
## 6      5 --166-
```

Tag_names are similar in spirit to "genres". Tag_ids are unique identifiers for each tag_name. Users assign self-chosen tag_names to books, and these tag names do not need to conform to any particular standard. Let's look at a few random entries...

```
    tags[100,2]
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 02-folklore
```

```
    tags[500,2]
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 1997
```

```
    tags[1000,2]
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 3-mistakes
```

```
    tags[5000,2]
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 book-lust-to-go
```

There are over 34,000 unique tag names, and I check to make sure that none of the tag_name fields are "NA".

```
    cat("Number of tags =", length(tags$tag_id))
```

```
## Number of tags = 34252
```

```
    cat("Number of unique tag_names =", length(unique(tags$tag_name)))
```

```
## Number of unique tag_names = 34252
```

```
    cat("Number of NAs in tag_names =", sum(is.na(tags$tag_name)))
```

```
## Number of NAs in tag_names = 0
```

The third file, book_tags.csv, maps each tag_id in tags.csv to a goodreads_book_id. This identification number maps to the book_id the books.csv file.

```
    head(book_tags)
```

```
## # A tibble: 6 x 3
##   goodreads_book_id tag_id  count
##               <dbl>  <dbl>  <dbl>
## 1                 1  30574 167697
## 2                 1  11305  37174
## 3                 1  11557  34173
## 4                 1   8717  12986
## 5                 1  33114  12716
## 6                 1  11743   9954
```

The file also contains a "count" field, which is a code for descriptors that I do not have access to. Therefore I do not use this field and drop it from the data set.

```
    book_tags <- subset(book_tags, select = -c(count))
```

Note that one book can have many different tags...

```
    book_tags
```

```
## # A tibble: 999,912 x 2
##    goodreads_book_id tag_id
##                <dbl>  <dbl>
##  1                 1  30574
##  2                 1  11305
##  3                 1  11557
##  4                 1   8717
##  5                 1  33114
##  6                 1  11743
##  7                 1  14017
##  8                 1   5207
##  9                 1  22743
## 10                 1  32989
## # ... with 999,902 more rows
```

The book_tags file has almost 1 million book_tags corresponding to the 34,000 tags we reviewed above, for 10,000 books, which means that on average a book has about 100 "genre" tags associated with it!

```
    cat("Total rows of book_tags =", nrow(book_tags))
```

```
## Total rows of book_tags = 999912
```

```
    cat("Number of unique tags =", length(unique(book_tags$tag_id)))
```

```
## Number of unique tags = 34252
```

```
    cat("Number of unique goodread_book_id =", length(unique(book_tags$goodreads_book_id)))
```

```
## Number of unique goodread_book_id = 10000
```

I double-checked that none of these tag_ids or goodreads_book_ids contained "NA" values.

```
    cat("Number of NAs in tag_names =", sum(is.na(book_tags$tag_id) |
                                            is.na(book_tags$goodreads_book_id)))
```

```
## Number of NAs in tag_names = 0
```

There are 34,252 unique tag_name fields in both tag files that represent user chosen genres. To make sense of these, we impose a genres classification on this user tag_names. I use a methodology presented by Philipp Spachtholz in his notebook.

Start with the goodbook's genres classification:

```
    genres <- str_to_lower(c("Art", "Biography", "Business", "Chick Lit", "Children's", "Christian",
                             "Classics", "Comics", "Contemporary", "Cookbooks", "Crime", "Ebooks",
                             "Fantasy", "Fiction", "Gay and Lesbian", "Graphic Novels",
                             "Historical Fiction", "History", "Horror", "Humor and Comedy", "Manga",
                             "Memoir", "Music", "Mystery", "Nonfiction", "Paranormal", "Philosophy",
                             "Poetry", "Psychology", "Religion", "Romance", "Science", "Science Fiction
                             "Self Help", "Suspense", "Spirituality", "Sports", "Thriller", "Travel",
                             "Young Adult"))
```

And exclude overly general classifications:

```
    exclude_genres <- c("fiction", "nonfiction", "ebooks", "contemporary")
    genres <- setdiff(genres, exclude_genres)
```

Select only those standard genres that appear in the tag_name fields, which corresponds to 27 available genres.

```
    available_genres <- genres[str_to_lower(genres) %in% tags$tag_name]
    length(available_genres)
```

```
## [1] 27
```

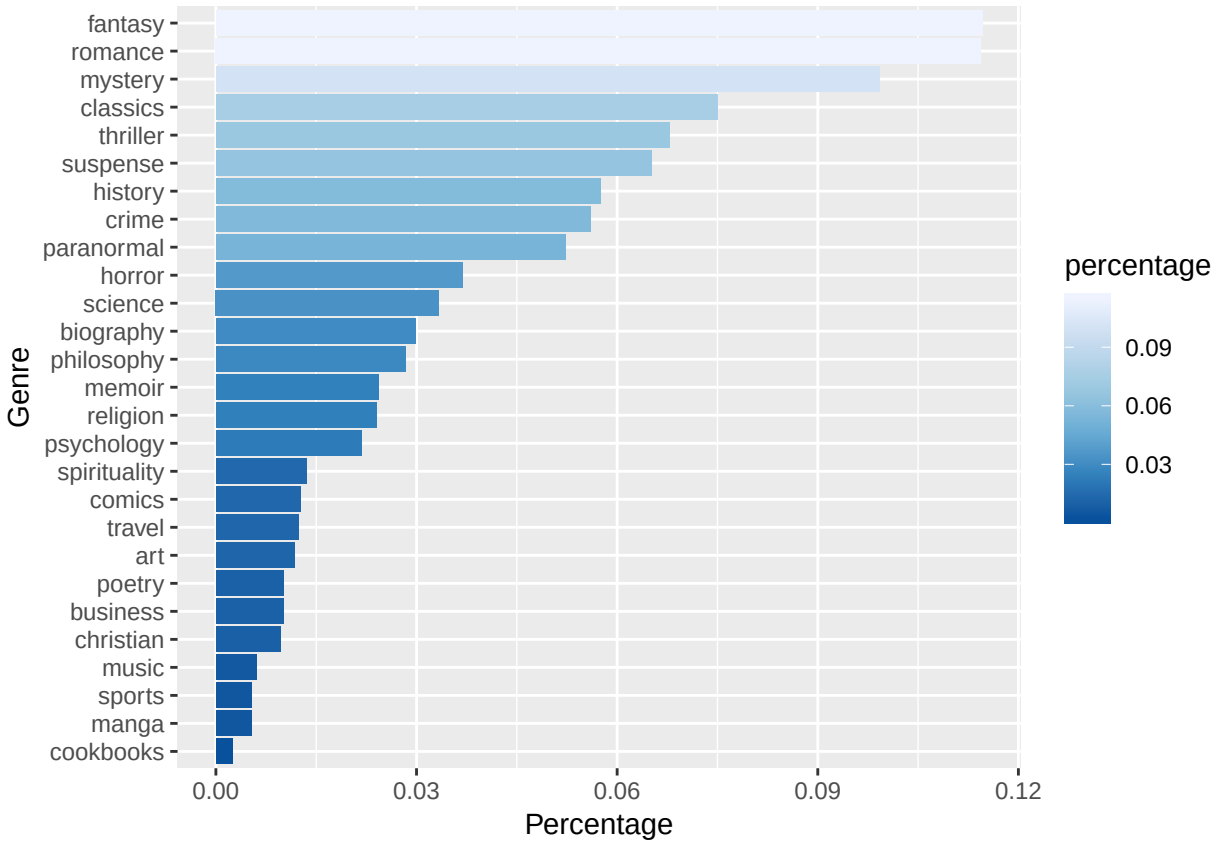
And then locate the corresponding tags, which should also be 27 in number.

```
    available_tags <- tags$tag_id[match(available_genres, tags$tag_name)]
    length(available_tags)
```

```
## [1] 27
```

Using the book_tags, we can determine the prevalence of each genre in this collection of ratings, and view through the following plot:

```
    tmp <- book_tags %>%
      filter(tag_id %in% available_tags) %>%
      group_by(tag_id) %>%
      summarize(n = n()) %>%
      ungroup() %>%
      mutate(sumN = sum(n), percentage = n / sumN) %>%
      arrange(-percentage) %>%
      left_join(tags, by = "tag_id")

    tmp %>%
      ggplot(aes(reorder(tag_name, percentage), percentage, fill = percentage)) +
      geom_bar(stat = "identity") + coord_flip() +
      scale_fill_distiller(palette = 'Blues') + labs(y = 'Percentage', x = 'Genre')
```



If we look at our book_tags after filtering for availability

```
    available_book_tags <- book_tags %>%
      filter(tag_id %in% available_tags)

    available_book_tags
```

```
## # A tibble: 37,153 x 2
##    goodreads_book_id tag_id
##                <dbl>  <dbl>
##  1                 1  11305
##  2                 1   7457
##  3                 1  22973
##  4                 1  20939
##  5                 1  26138
##  6                 2  11305
##  7                 2  22973
##  8                 2  20939
##  9                 2  26138
## 10                 3  11305
## # ... with 37,143 more rows
```

We see that the 37,000 tags are spread over about 10,000 books, or 3.7 tags per book, which which is summarized below.

```
    available_book_tags <- book_tags %>%
      filter(tag_id %in% available_tags) %>%
      group_by(goodreads_book_id) %>%
      summarize(n = n())
```

```
    available_book_tags
```

```
## # A tibble: 9,863 x 2
##    goodreads_book_id     n
##                <dbl> <int>
##  1                 1     5
##  2                 2     4
##  3                 3     4
##  4                 5     4
##  5                 6     4
##  6                 8     5
##  7                10     4
##  8                11     3
##  9                13     4
## 10                21     4
## # ... with 9,853 more rows
```

About 1.4% of our books are excluded because they have a genre not within our available set. If we look at the tags for any particular book, say goodreads_book_id number 1 above, we can see that a book with several tags covers a lot of general subjects.

```
    print(tags %>% filter(tag_id == 11305) %>% select(tag_name))
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 fantasy
```

```
    print(tags %>% filter(tag_id == 7457) %>% select(tag_name))
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 classics
```

```
    print(tags %>% filter(tag_id == 22973) %>% select(tag_name))
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 paranormal
```

```
    print(tags %>% filter(tag_id == 20939) %>% select(tag_name))
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 mystery
```

```
    print(tags %>% filter(tag_id == 26138) %>% select(tag_name))
```

```
## # A tibble: 1 x 1
##   tag_name
##   <chr>
## 1 romance
```

This particular book (# 1) has genres covering 5 of the 27 available genres. Some books have up to a dozen of the 27 genres. We need to do some more genre feature engineering to make this more useful.

The fourth and final file, books.csv, contains meta data for each book.

```
## # A tibble: 6 x 23
##   book_id goodreads_book_~ best_book_id work_id books_count isbn   isbn13
##     <dbl>            <dbl>        <dbl>   <dbl>       <dbl> <chr>   <dbl>
## 1       1          2767052      2767052  2.79e6         272 4390~ 9.78e12
## 2       2                3            3  4.64e6         491 4395~ 9.78e12
## 3       3            41865        41865  3.21e6         226 3160~ 9.78e12
## 4       4             2657         2657  3.28e6         487 6112~ 9.78e12
## 5       5             4671         4671  2.45e5        1356 7432~ 9.78e12
## 6       6         11870085     11870085  1.68e7         226 5254~ 9.78e12
## # ... with 16 more variables: authors <chr>,
## #   original_publication_year <dbl>, original_title <chr>, title <chr>,
## #   language_code <chr>, average_rating <dbl>, ratings_count <dbl>,
## #   work_ratings_count <dbl>, work_text_reviews_count <dbl>,
## #   ratings_1 <dbl>, ratings_2 <dbl>, ratings_3 <dbl>, ratings_4 <dbl>,
## #   ratings_5 <dbl>, image_url <chr>, small_image_url <chr>
```

There are 10,000 rows, one for each book in our data set.

```
## [1] 10000
```

As above, we will work through and explain each of the fields, removing those that provide no expected benefit. First, let's remove the ISBN and ISBN13 which add no value to our modeling.

Second, there are four book identification numbers listed:

(1) "book_id" with 10,000 entries, which match the id numbers in the ratings.csv file

(2) "goodreads_book_id" that match the identifier in the book_tags file

(3) "best_book_id", which corresponds to the most popular edition of a given book, generally this is the same as the as the goodreads_book_id, except in 241 cases.

```
print(nrow(books %>% filter(goodreads_book_id != best_book_id)))
```

```
## [1] 241
```

(4) "work_id" refers to a book in the abstract sense. That is, it represents a generic class of all editions of a book.

Which identification numbers should we use? The book_id ties this file to the ratings.csv file, and so this will be our main identification number for books. The goodreads_book_id ties this file to the book_tags.csv and so we will keep this identification number in the file for the time being. We will discard the best_book_id and the work_id.

```
books <- subset(books, select = -c(best_book_id, work_id))
```

Since we removed the work_id, we can also remove the related column work_ratings_count, work_text_reviews_count, as well as the irrelevant image files.

```
books <- subset(books, select = -c(work_ratings_count,
                                   work_text_reviews_count))
books <- subset(books, select = -c(image_url, small_image_url))
```

The file lists the books original_title and title, about half of which changed over time. I chose to use the current title.

```
print(nrow(books %>% filter(original_title != title)))
```

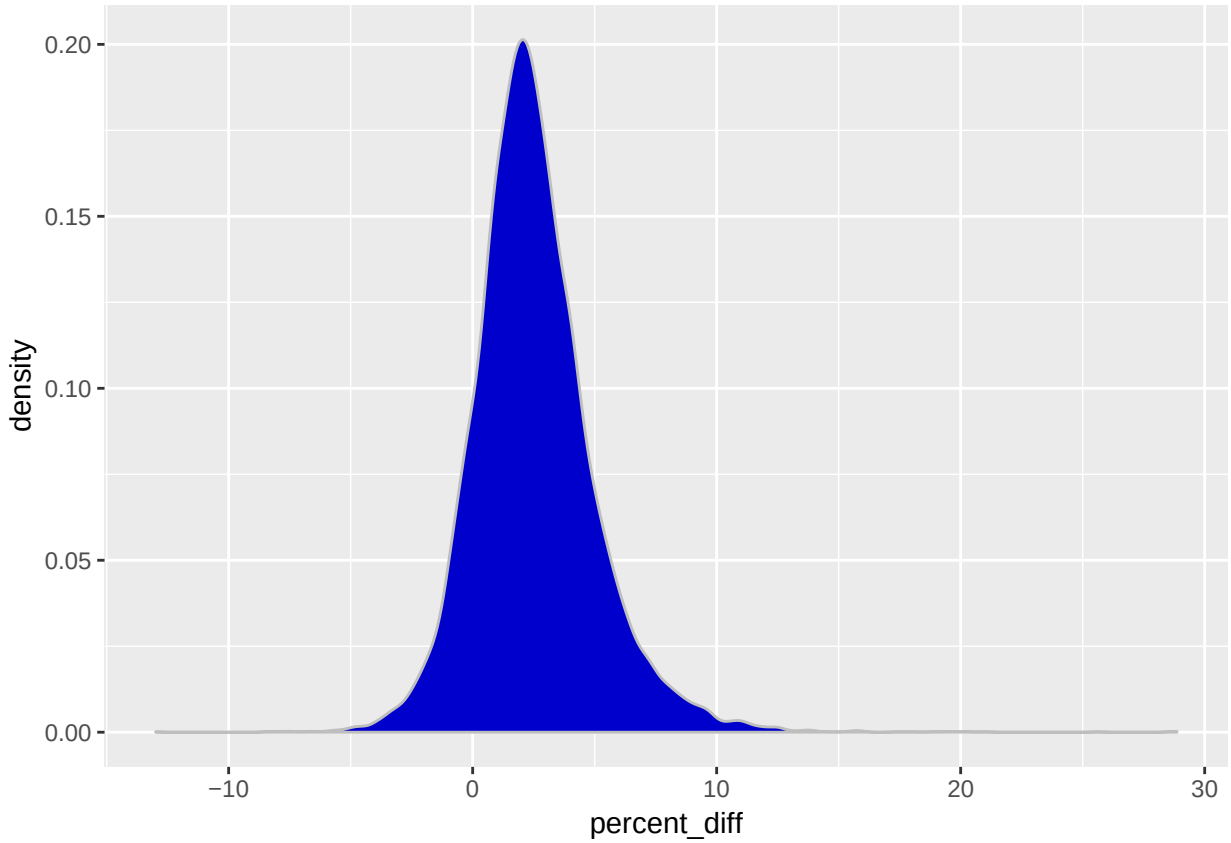
```
## [1] 5314
```

```
books <- subset(books, select = -c(original_title))
```

This file contains the ratings in each category (1-5), as well as the average. Question – do the ratings in this file match the ratings in our ratings.csv file?

To see any differences, we pull the averages out of the books.csv file, calculate the averages from the ratings.csv file, join the two, and plot the percentage difference.

```
# Pull the averages out of the books file
books_file_summary <- books %>% select(book_id, average_rating)

# Calculate the averages out of the ratings.csv file
ratings_file_summary <- ratings %>% group_by(book_id) %>%
  summarize(rating_file_average = mean(rating))

# Join the two together
ratings_comparison <- left_join(books_file_summary,
                                ratings_file_summary, by= "book_id")
# Calculate the percent difference
ratings_comparison <- ratings_comparison %>%
  mutate(percent_diff = 100*(1-rating_file_average/average_rating))

# Plot it
ratings_comparison %>%
  ggplot(aes(percent_diff)) +
  geom_density(fill = "mediumblue", color = "grey")
```



The mean difference is 2.5%, and the standard deviation is 2.5%.

```
    # The mean difference in estimates is 2.5%
    mean(ratings_comparison$percent_diff)
```

```
## [1] 2.509124
```

```
    # The standard deviation is 2.5%
    sd(ratings_comparison$percent_diff)
```

```
## [1] 2.489418
```

These two rating averages are close. We will use the averages reported in the books.csv file for our preliminary analysis, but use the ratings in the ratings.csv file for our modeling. We do not need to use the ratings_1 through _5, so we remove them.

```
    books <- subset(books, select = -c(ratings_1,
                                       ratings_2,
                                       ratings_3,
                                       ratings_4,
                                       ratings_5))
```

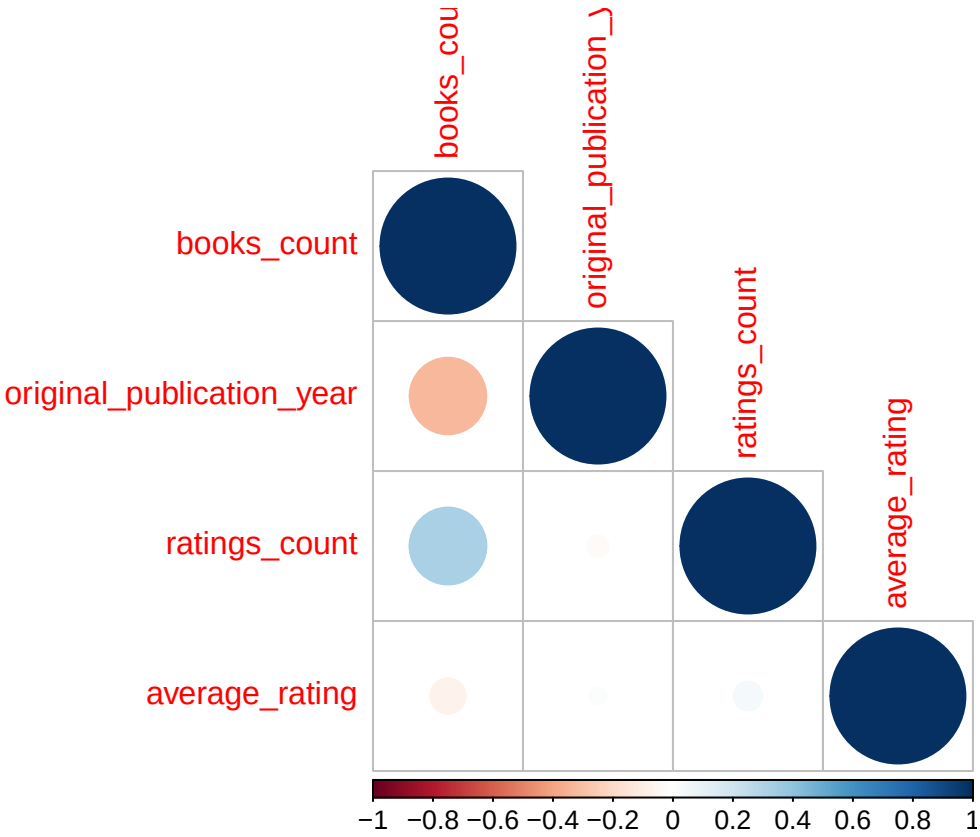
Explanation of remaining fields:

- books_count is how many editions of a book have been published
- authors and original_publication_date are self-explanatory
- ratings_count is the number of ratings

Which of these fields influence a book's average rating?

I borrowed this helpful analysis from Philipps notebook:

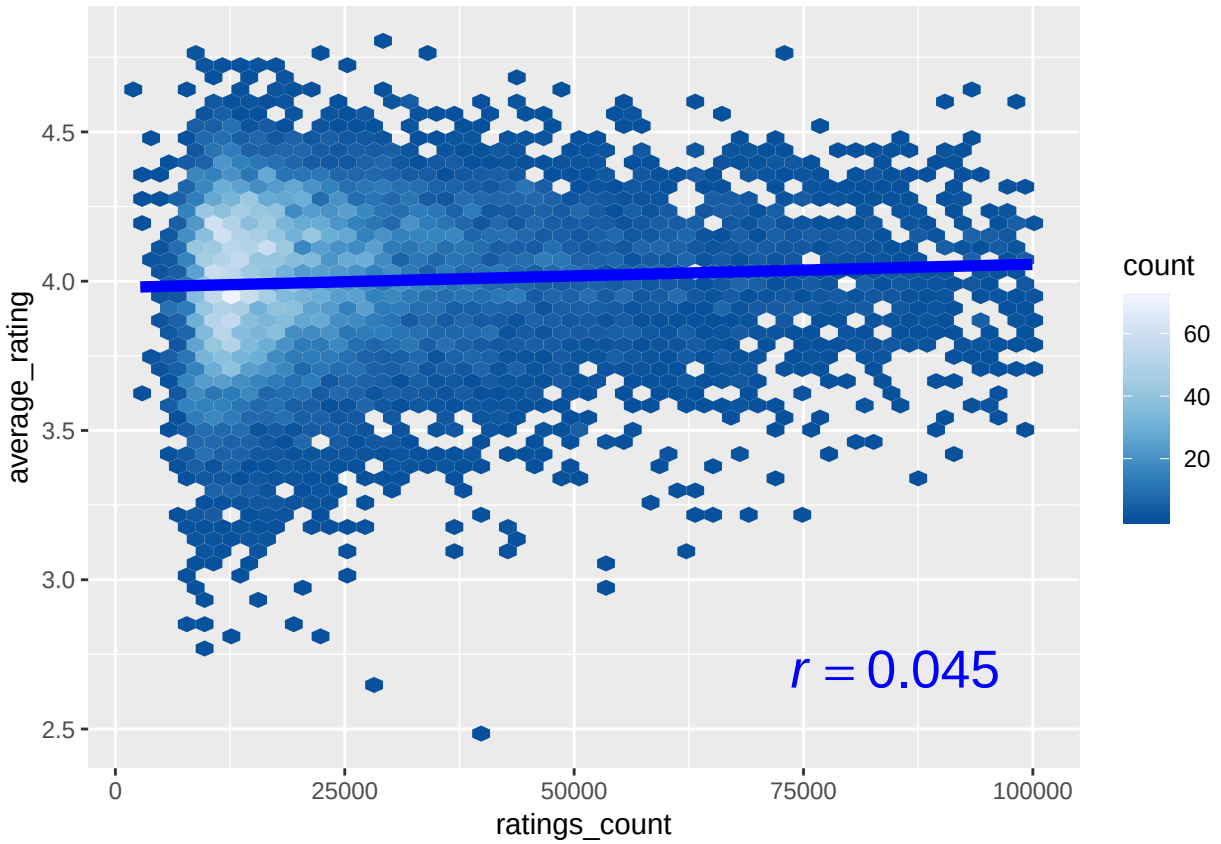
```
    tmp <- books %>%
      select(one_of(c("books_count","original_publication_year",
                      "ratings_count", "average_rating"))) %>% as.matrix()

    corrplot(cor(tmp, use = 'pairwise.complete.obs'), type = "lower")
```



From which we can see a very slight negative correlation with the number of editions and very slight positive correlation with the number of ratings.

We examine these these correlations further with the following plot from Philipp

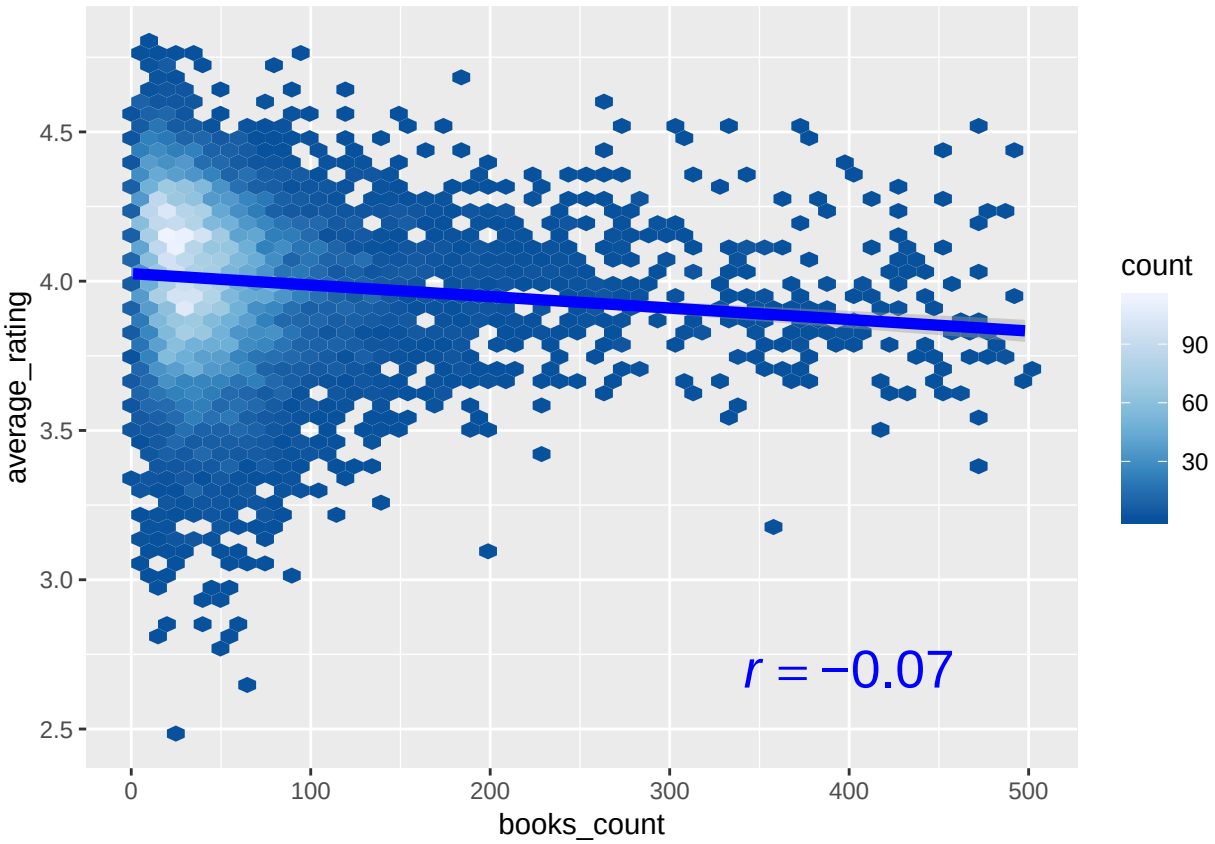
```
get_cor <- function(df){
  m <- cor(df$x,df$y, use="pairwise.complete.obs");
  eq <- substitute(italic(r) == cor, list(cor = format(m, digits = 2)))
  as.character(as.expression(eq));
}

books %>%
  filter(ratings_count < 1e+5) %>%
  ggplot(aes(ratings_count, average_rating)) + stat_bin_hex(bins = 50) +
  scale_fill_distiller(palette = "Blues") +
  stat_smooth(method = "lm", color = "blue", size = 2) +
  annotate("text", x = 85000, y = 2.7,
           label = get_cor(data.frame(x = books$ratings_count, y = books$average_rating)),
           parse = TRUE, color = "blue", size = 7)
```



And we see the correlation between ratings_count and average rating is only 0.045 and not worth including in our model.

Similarly, look at the correlation with the number of editions.

```
books %>% filter(books_count <= 500) %>% ggplot(aes(books_count, average_rating)) +
  stat_bin_hex(bins = 50) + scale_fill_distiller(palette = "Blues") +
  stat_smooth(method = "lm", color = "blue", size = 2) +
  annotate("text", x = 400, y = 2.7,
           label = get_cor(data.frame(x = books$books_count, y = books$average_rating)),
           parse = TRUE, color = "blue", size = 7)
```



The original_publication_year has no impact, and so we remove that from our file as well.

```
    books <- subset(books, select = -c(original_publication_year))
```

Philipp's analysis shows that there is a small correlation between the rating and number of authors and average rating (0.075), but not enough to include in our model.

Data Preparation

After analyzing the files, we prepare the data for modeling.

First we create the validation data set and set it aside. Then we create training and testing data sets form the remaining data.

```
  set.seed(1)
  val_index <- createDataPartition(y = ratings$rating, times = 1, p = 0.1, list = FALSE)
  data_set <- ratings[-val_index,]
  validation_set <- ratings[val_index,]

# To ensure we do not include users and books in the test set
# that do not appear in  the training set we remove those entries
# with the semi_join function
  validation_set <- validation_set %>%
    semi_join(data_set, by = "book_id") %>%
    semi_join(data_set, by = "user_id")

# We then partition the data_set into training and test sets
  set.seed(1)
```

```
  test_index <- createDataPartition(y = data_set$rating, times = 1, p = 0.2, list = FALSE)
  train_set <- data_set[-test_index,]
  test_set <- data_set[test_index,]

  # To ensure we do not include users and books in the test set
  # that do not appear in  the training set we remove those entries
  # with the semi_join function
  test_set <- test_set %>%
    semi_join(train_set, by = "book_id") %>%
    semi_join(train_set, by = "user_id")
```

Model Selection

The model and parameter selection process began by building the recommendation system described in the course textbook to see how it performed relative to the RMSE levels shown in the rubric.

We will test the accuracy of our model with the following RMSE function.

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

Similar to the textbook, our first model predicts the same rating for all books regardless of the user. We compute mu, the associated naive_rmse, and store it in a table.

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

```
## [1] 3.919892
```

```
# If we predict all unknown ratings with mu we obtain the following RMSE:
  naive_rmse <- RMSE(mu, test_set$rating)
  naive_rmse
```

```
## [1] 0.9911228
```

```
# Let's create a results table and add our first entry
  rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

We augment our model by adding term b_i to represent the average ranking for book i. b_i is the average least square estimate of the difference between Y - mu for each book. Let's see how much our prediction improves once we add b_i to our model

```
  book_avgs <- train_set %>%
    group_by(book_id) %>%
    summarize(b_i = mean(rating - mu))

  predicted_ratings <- mu + test_set %>%
    left_join(book_avgs, by='book_id') %>%
    pull(b_i)

  model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

  rmse_results <- bind_rows(rmse_results,
                            tibble(method="Book Effect Model", RMSE = model_1_rmse))
  cat("Model_1_rmse =", model_1_rmse)
```

```
## Model_1_rmse = 0.9538607
```

Now we augment our previous model by adding the term b_u to represent the average ranking for user u. We can now construct predictors and see how much the RMSE improves:

```
user_avgs <- train_set %>%
  left_join(book_avgs, by='book_id') %>%
  group_by(user_id) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test_set %>%
  left_join(book_avgs, by='book_id') %>%
  left_join(user_avgs, by='user_id') %>%
  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="Book + User Effects Model",
                                 RMSE = model_2_rmse))
cat("Model_2_rmse = ", model_2_rmse)
```

```
## Model_2_rmse =  0.8584616
```

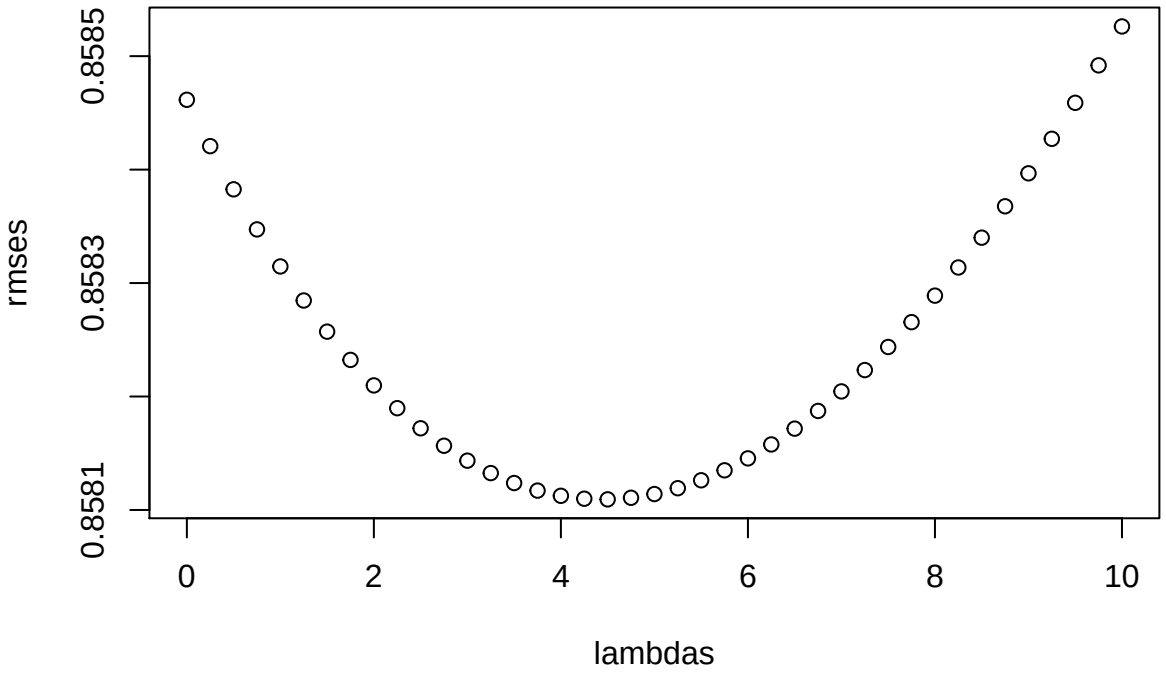
Similar to our textbook, I explored the possibility of optimizing the b_i and b_u parameters to potentially enhance the model.

Parameter Optimization

As we witnessed in our course materials and exercises, small numbers of observations for any parameter can lead to large variability, which degrades a model. The parameters b_i and b_u may be optimized for low numbers of ratings for any particular movie i or user u. In this case, we can regularize the parameters by the number of observed ratings and a constant lambda. The following code computes a lambda value for b_i and b_u that minimizes the RMSE.

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(book_id) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="book_id") %>%
    group_by(user_id) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "book_id") %>%
    left_join(b_u, by = "user_id") %>%
    mutate(pred = mu + b_i + b_u) %>%
```

```
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

plot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda # 4.5
```

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

As we can see from the plot a value of 4.5 minimizes the RMSE, but different values of lambda in the neighborhood of 4.5 also produce acceptable RMSE values.

We record the RMSE associated with this lambda in our results table.

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Book + User Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	0.9911228
Book Effect Model	0.9538607
Book + User Effects Model	0.8584616
Regularized Book + User Effect Model	0.8581094

As we can see from the table, the regularization process adds little additional predictive power to our model.

Model Training and Evaluation

At this point we are satisfied with our model and the parameter optimization. We now train the model with the full training data set:

```
mu <- mean(data_set$rating)
lambda <- 4.5

b_i <- data_set %>%
  group_by(book_id) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- data_set %>%
  left_join(b_i, by="book_id") %>%
  group_by(user_id) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

predicted_ratings <- validation_set %>%
  left_join(b_i, by = "book_id") %>%
  left_join(b_u, by = "user_id") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

We then evaluate our final trained model against the "validation" data set, and recorded the results in our data table.

```
final_results <- RMSE(predicted_ratings, validation_set$rating)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Validation dataset",
                                 RMSE = final_results))
```

Results

Below is the final results table from the previous Methods and Analysis section.

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

method	RMSE
Just the average	0.9911228
Book Effect Model	0.9538607
Book + User Effects Model	0.8584616
Regularized Book + User Effect Model	0.8581094
Validation dataset	0.8560672

Our model includes a regularized book effect parameter and regularized user effect parameter. The resulting evaluated RMSE was 0.8560672, which is well below the target of 0.87750.

Conclusions

In this project I learned how to use Kaggle.com, which was a goal of mine. I learned that there is a lot of work that needs to be done examining, cleaning and constructing a new data set before even thinking about building a model. I applied the methodologies we learned in class to a book recommendation system, similar to the MovieLens project, and built and trained a model that produced a result on the validation data set that was similar in improvement to my MovieLens project. I also experimented with the RecommenderLab package in both my MovieLens project and in this project, although I did not include this work in this project writeup because I had a bug that kept crashing my model. (RecommenderLab worked a little better on my MovieLens project.)