Amazon Review Research

Ryan Erwin

December 2, 2015

Contents

Overview

This project entails web scraping, text mining, and predictive models with the objective of predicting "Review" (y/n) and the rating (number of stars). This approach seeks to help sellers target the reviewers most likely to review their product with a high rating, which will also be seen as helpful to other shoppers. However, due to memory constraints and computing power, I've fallen short of this goal. I have established a relationship between the review text and the review ratings. Lower ratings are associated with shorter reviews, while higher ratings are associated with longer reviews, where length is defined by word count. In

addition, I've observed how the average word count per review has varied over time, with more recent reviews more likely to be slightly shorter than previous years.

Another theme of the project is the adoption of technology outside of **R** such as Amazon Web Services (AWS), PuTTY, and WinSCP to name a few. The large volume of reviews targeted for web scraping translates to a large number of requests to Amazon's web servers which are sophisticated and keep track of who is requesting data — the IP address of the machine requesting the data — and how often they are requesting that data. As a result, I needed multiple IP addresses to gather the data to avoid being temporarily banned from retrieving data from the Amazon's servers.

While I did explore techiques outside of **R**, the vast majority of the work takes place in **R** using RStudio as an IDE. I also used many packages the various tasks which will be outlined below. The packages used include `rvest`, `stringr`, `dplyr`, `qdap`, and `snow`. Each of the packages handles a specific need within the project and without them the tasks required would most likely be inaccessible to me. Thus they are a critical part of my research and I will dedicate significant time to outlining why I use them and when to use them.

Data

Overview

The information to be analyzed must be scraped from Amazon.com's list of Top Reviewers. For example, we'll need to identify the top reviewers then gather reviews. For each review, we'll want the review text, rating, percent and absolute value of helpful votes, product, product metadata, and any user (Reviewer) information available. Once the data is gathered, it must be stored in a format that makes data analysis possible and easy to generalize across multiple iterations of web scraping. For example, the same information should be gathered for each review. If this is not true, then the storage of the data becomes a bit trickier and the standard tabular format found in **R**'s `data.frame`'s will not be the best tool. So, to make life easier I have to map which review characteristics to retrieve beforehand. In addition, regarding the size of the data set

- Total Reviews — 1.7 M reviews
- Total Reviewers — 9,747 unique reviewers
- Total Products — 971k different products reviewed
- Total Size — 2.5 Gb

What Does the Data Originate?

As mentioned above, the data consists of live Amazon product reviews written by Amazon's Top Reviewers. The Top Reviewers list is a program that seeks to identify Amazon's most influential members of the review community. The list is updated every two days, so the list of reviewers I examine is snapshot of reviewers as of September 2015. The Top Reviewers ranks the top 10,000 reviewers at any given time. The rankings are based on an algorithm where recency of review and the number of helpful reviews, voted upon by fellow Amazon customers, are inputs. For illustration, I've inlcuded a screen shot of the Top Reviewers web page below.

Amazon's Top Customer Reviewers

Our top reviewers have helped millions of their fellow customers make informed purchase decisions on Amazon.com with their consistently helpful, high-quality reviews contributors at the moment, while the Hall of Fame honors those who have been highly ranked in previous years. Take a minute to explore the reviews written by these

Top Reviewer Rankings | Hall of Fame Reviewers

10,000 customer reviewers — « Previous 1 2 ... 1000 Next »

Rank	Customer Reviewer	Total Reviews	Helpful Votes
# 1	Ali Julia See all 3,884 reviews	3,884	37,023
# 2	Jackie Cooper See all 913 reviews	913	14,912
# 3	J. Chambers See all 4,092 reviews	4,092	57,880

So, the first task is to get list of reviewers and the link to their reviews. This information is stored in HTML delivered from Amazon with each request sent to their servers. The HTML is then rendered into a

final output locally using a web browser, in my case Google Chrome. The first challenge is to determine a systematic way to get the list delivered in HTML, select the elements I want, then transform those elements into a format that is consistent for each reviewer. First, I need to load the `rvest` package, which downloads the HTML document delivered from a HTTP GET request. Furthermore, `rvest` comes with a set of functions that make working the HTML document intuitive, such as `html_nodes`, `html_attr`, and `html_text`. I utilized the *xpath* argument of `html_nodes` to determine where each element I wanted would be located across all pages of the Top Reviewers list. Yet, before I begin coding this part of the project an inpsection of the web page's HTML structure must take place. This process of trial and error is the most time consuming component of the web scraping: how to determine a pattern that will generalize across all web pages? For example, see the screen shot of the HTML source code below.

```
<table class="a-bordered a-horizontal-stripes  a-align-center a-spacing-top-none a-size-small crDataGrid neg-margin">
  <tr>
      <th class="a-color-secondary crNum">Rank</th>
      <th></th>
      <th class="a-color-secondary crNum">Customer Reviewer</th>
      <th class="a-color-secondary crNum">Total Reviews</th>
      <th class="a-color-secondary crNum">Helpful Votes</th>
    <th class="a-color-secondary crNumPercentHelpful">Percent Helpful</th>
  </tr>
  <tr id="reviewer1">
     <td class="crNum"># 1 </td>
     <td class="img" align="center">  <a href="/gp/pdp/profile/A2D1LPEUCTNT8X/ref=cm_cr_tr_tbl_1_pic"><img src="https://images-na.ssl-images-
amazon.com/images/I/51qmN9z9PwL._SS72_.jpg" info="cmsImage" border="0"/></a>  <a name="A2D1LPEUCTNT8X"></td>
     <td> <a href="/gp/pdp/profile/A2D1LPEUCTNT8X/ref=cm_cr_tr_tbl_1_name"><b>Ali Julia</b></a><div class="tiny"><a href="/gp/cdp/member-
reviews/A2D1LPEUCTNT8X/ref=cm_cr_tr_tbl_1_sar?ie=UTF8&amp;sort_by=MostRecentReview">See all 3,887 reviews</a></div> </td>
     <td class="crNum"> 3,887 </td>
     <td class="crNum"> 37,046 </td>
  <td class="crNumPercentHelpful"> 96% </td>

  </tr>
  <tr id="reviewer2">
     <td class="crNum"># 2 </td>
     <td class="img" align="center">  <a href="/gp/pdp/profile/A14ZQ17DIPJ6UB/ref=cm_cr_tr_tbl_2_pic"><img src="https://images-na.ssl-images-
amazon.com/images/I/81gLXmeDY0L._SS72_.jpg" info="cmsImage" border="0"/></a>  <a name="A14ZQ17DIPJ6UB"></td>
     <td> <a href="/gp/pdp/profile/A14ZQ17DIPJ6UB/ref=cm_cr_tr_tbl_2_name"><b>Jackie  Cooper</b></a><div class="tiny"><a href="/gp/cdp/member-
reviews/A14ZQ17DIPJ6UB/ref=cm_cr_tr_tbl_2_sar?ie=UTF8&amp;sort_by=MostRecentReview">See all 913 reviews</a></div> </td>
     <td class="crNum"> 913 </td>
     <td class="crNum"> 14,960 </td>
  <td class="crNumPercentHelpful"> 96% </td>
```

As you can see, I've highlighted — in Yellow — two elements that I'd like to scrape from the HTML, the identifier Amazon uses to identify an reviewer and the reviewer metrics, in this case the number of total reviews. To see how I've approached this problem, please see the code below. The first step is to load the package and read the HTML document to memory using the `read_html` function. Then, use *xpath* arguments to specify each piece of reviewer information.

```
library(rvest)

# page is a url, such as http://www.google.com
html <- read_html(page)


# Amazon identifier
url_name <- html %>%
  html_nodes(xpath = "//tr[contains(@id, 'reviewer')]/td[2]/a[2]") %>%
  html_attr("name") %>%
  as.data.frame.character(stringsAsFactors = F)


# get number of reviews, votes, and % helpful
```

```
metrics <- html %>%
  html_nodes(xpath = "//tr[contains(@id, 'reviewer')]/td[contains(@class, 'crNum')]") %>%
  html_text() %>%
  matrix(nrow = 10, byrow = T) %>%
  as.data.frame.matrix(stringsAsFactors = F)


# rename the columns to meanful names
colnames(metrics) <- c("Rank", "Reviews", "Helpful_votes", "%_helpful")
```

This gets us the list of the reviewers, which contains a url to each reviewers list of reviews. Getting to the list of review urls is what we really want. Each review url will take you to a web page that contains up to ten reviews, with as many pages needed to capture all of that particular reviewers' reviews. For instance, let's refer to a hypothetical reviewer, AmazonJoe to illustrate this process. AmazonJoe's review list url is.

> 'http://www.amazon.com/gp/cdp/member-reviews/AmazonJoe?ie=UTF8&display=public&page=1&sort_by=MostRecentReview'

This url will lead to the last ten reviews written by AmazonJoe. If AmazonJoe has one hundred total reviews, then AmazonJoe will have ten pages (`100 reviews / 10 reviews per page`) of reviews to gather and scrape. Notice that in the url above, the first part of the specifies a directory. The second part, beginning with `?`, is the query with name-value pairs of parameters. Returning to AmazonJoe, we'd have to go through each of the ten url's by using the `page` parameter, iterating through each value:

> 'http://www.amazon.com/?.....page=1'
> 'http://www.amazon.com/?.....page=2'
> '...'
> 'http://www.amazon.com/?.....page=10'

The code that produces the links and iterates through them is located in the Appendix section. If you're interested, the name of the function that creates the list of urls is `.create_links` and is defined in the `data_last_step.R` script. So, once I've procured the list of url's for the actual reviews, I have to translate the HTML document into useable information. I use the `rvest` package again. Let's turn to another quick example of how I went about parsing the HTML data, using a screen shot and code snippet.

```
      <div style="margin-bottom:0.5em;">
        <span style='margin-left: -5px;'><img src="http://g-ecx.images-ama
5 stars" title="3.0 out of 5 stars" height="12" border="0" /> </span>
        <b>Large pearl earrings</b>, <nobr>November 29, 2015</nobr>
      </div>
      <div class="tiny" style="margin-bottom:0.5em;">
        <span class="crVerifiedStripe"><b class="h3color tiny" style="marg
href="http://www.amazon.com/gp/community-help/amazon-verified-purchase" ta
'AmazonHelp', 'width=400,height=500,resizable=1,scrollbars=1,toolbar=0,sta
      </div>
      <div class="tiny" style="margin-bottom:0.5em;">
        <b><span class="h3color tiny">This review is from: </span><a href=
Simulated/dp/B0170A6GKK/ref=cm_aya_orig_subj">Presentski Black Friday Big
(Jewelry)</a></b>
      </div>

<div class="reviewText">Some costume jewelry pieces are hard to tell from
pearl has imperfections so it is clear that it is a bead.  The fact that t
the quality of the simulated pearl that is in question.<br /><br />The siz
The dangling part is 1.25&amp;quot; and 0.75&amp;quot; wide.  Despite the size the
loop I was able to determine that it is decorate stamp rather than the met
I rate the earrings as okay.  True to their description but with some impe
distance.  I received these earrings for evaluation and review.  Getting a
so other people have additional information that can be only be gleaned in
```

In this case, I want the date and the review text. So, let's take a look at the `R` code that would parse the review text component out of the HTML.

```
# review_page is a url to a page of reviews
html <- read_html(review_page)


text <- html %>%
  html_nodes(xpath = "//div[@class = 'reviewText']") %>%
  html_text()
```

Up to this point, I've refered to *xpath*, but I have not explained it, though using the accompanying screen shots should allow you to get an intuitive understanding. First, the

"//"

tells the function to select all nodes that appear in the document satisfying the remainder of the expression. Next, the

"div"

component of the argument specifies to look for *div* nodes. Combining these two components, the *xpath* argument says grab all the *div* nodes in the HTML document, but we're not finished yet. Now, we use a special element of the argument called a *predicate*. In this case the predicate is

"[@class = 'reviewText']"

and it says, combined with all previous elements, find all *div* nodes where the *class* attribute is "reviewText."

Rate Restrictions and Multiple IP's

I'd like to briefly discuss the role of several technologies outside of **R** that made this project possible, namely Amazon Web Services (AWS). To retrieve millions of reviews from Amazon's servers required hundreds of thousands of GET requests. If all of those GET requests came from one PC with the same IP address, then Amazon might think I have malicious intentions and temporarily ban my IP from retrieving additional data. To overcome this hurdle, I employed AWS and ran my web scraping script across more than twenty machines. This approach delivers on two fronts: it parallelizes the data collection, thus reducing human waiting time, and it mitigates the problem of sending too many GET requests from one IP. I have included the code and a reference link in the accompanying PowerPoint presentation if you're interested.

The Final Data Set

I've now gone through each url and grabbed the information I want, but how do I store this information to be used later? Well, let's turn the another code snippet to get an understanding. Here I've already parsed the information and I'm storing it in a `data.frame`. This makes it very easy to save to disk, using `write.csv` and read back to memory later using `read.csv`.

```
df <- data.frame(text = text,
                 rating = rating,
                 review_date = date,
                 product_id = p_id,
                 review_id = id,
                 reviewer = user,
                 review_page = link,
                 trouble = "correct",
                 stringsAsFactors = F
                 )
```

Methodology

Overview

As a first approximation, we'll apply the random forest algorithm. I choose RF initially for out-of-box performance and relative ease of application. As the modeling progresses, the modelling approach will certainly evolve. The vast majority of programming will take place within the R language. The primary predictors of interest are the word frequencies for each review. This means some text mining preprocessing must take place before joining with the other predictors. The final table used for model training will be composed such that each row represents a customer and the columns are reviewer characteristics used as predictors.

The response I am looking to predict is whether or not a reviewer will review a certain product given past behavior. Furthermore, the response is binary and random forest is well-suited for this problem because of it's flexibility and strong performance on many applications and data sets. To describe the primary question of the project in more detail, we're interested in this question, and data set in general, for a few reasons. First, the ability to predict, given a product, whether an Amazon reviewer will write a review has potential value for the company that supplies this product. If the companies can identify the Amazon reviewers most likely to speak about their product, then they can reach out to these individuals and offer them the ability to review the item prior to release. To take this analysis a step further, we can also attempt to predict the rating a particular Amazon reviewer will give a product. Combining these two predictions, one could determine not only who will review, but also whether it will be favorible or not.

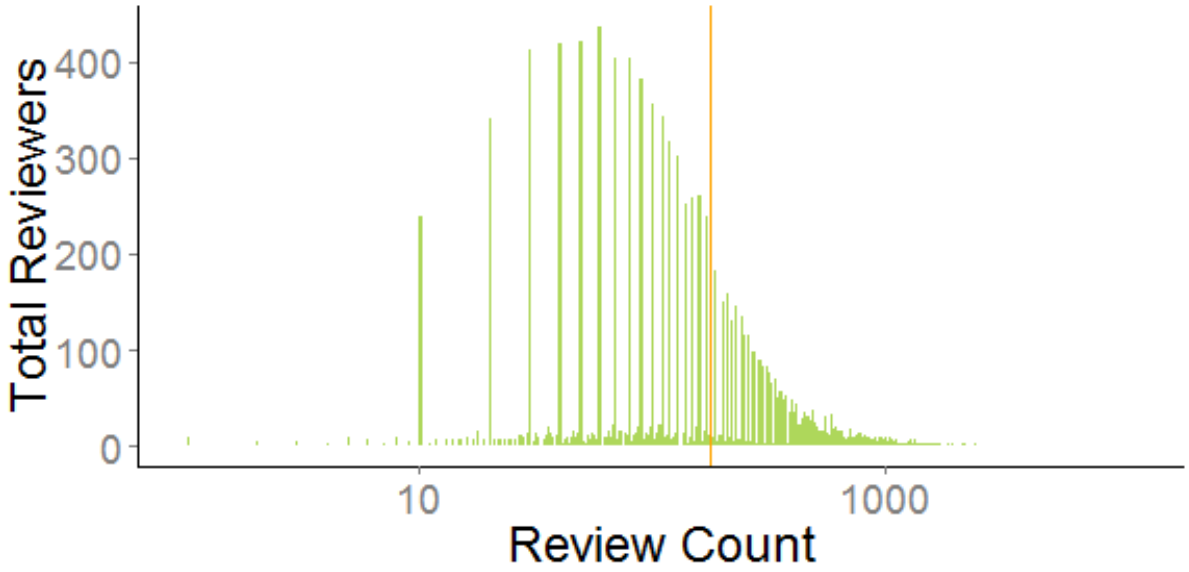
Companies work very hard to gain an understanding of customer sentiment: what products are

popular, why are they popular, and what features do people discuss most often? Amazon reviews are certainly not the only place where this information is located but I would make a strong argument that Amazon reviews should be considered. The reviewers are there to speak about the products, so less resources are needed to determine the subject matter of the text itself.

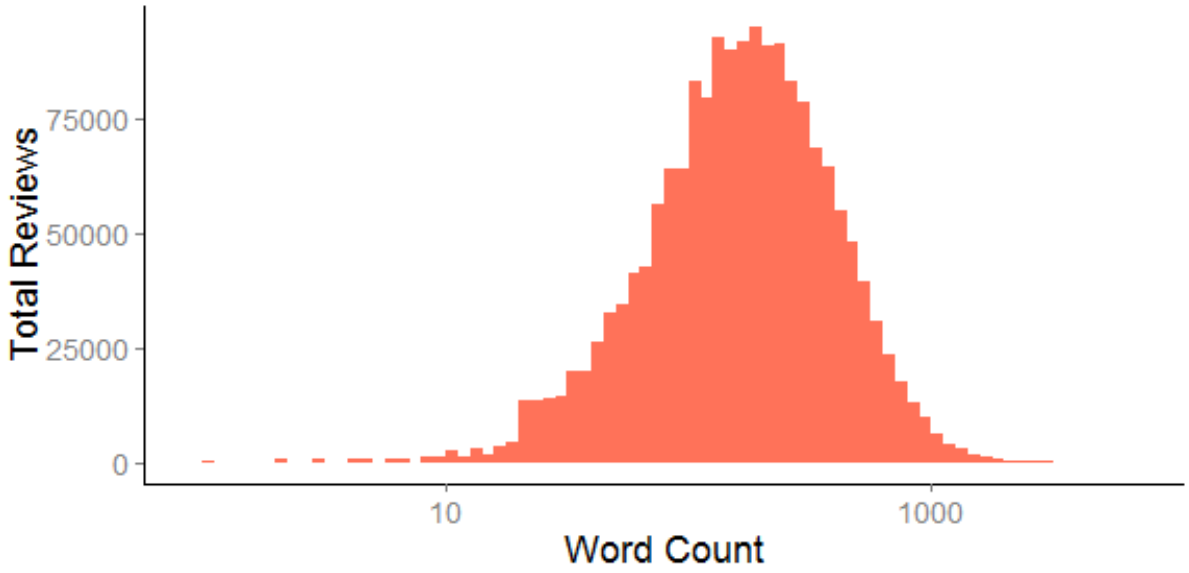
Current State of Modeling

As mentioned in previous sections, the data set presents challenges to **R**'s in-memory processing. If I have a machine dedicated to analysis, then code runtime is not much of a concern. However, if you're planning to process the data set in it's current form and then perform any type of modeling on a general-purpose PC you lose the ability to perform other tasks, which as a busy graduate student is not always possible. In addition, the data set is still growing and will be appended with price and category in the near future. As a result the modeling completed thus far is descriptive in nature and lacks the sophistication and accuracy of the Random Forest algorithm.

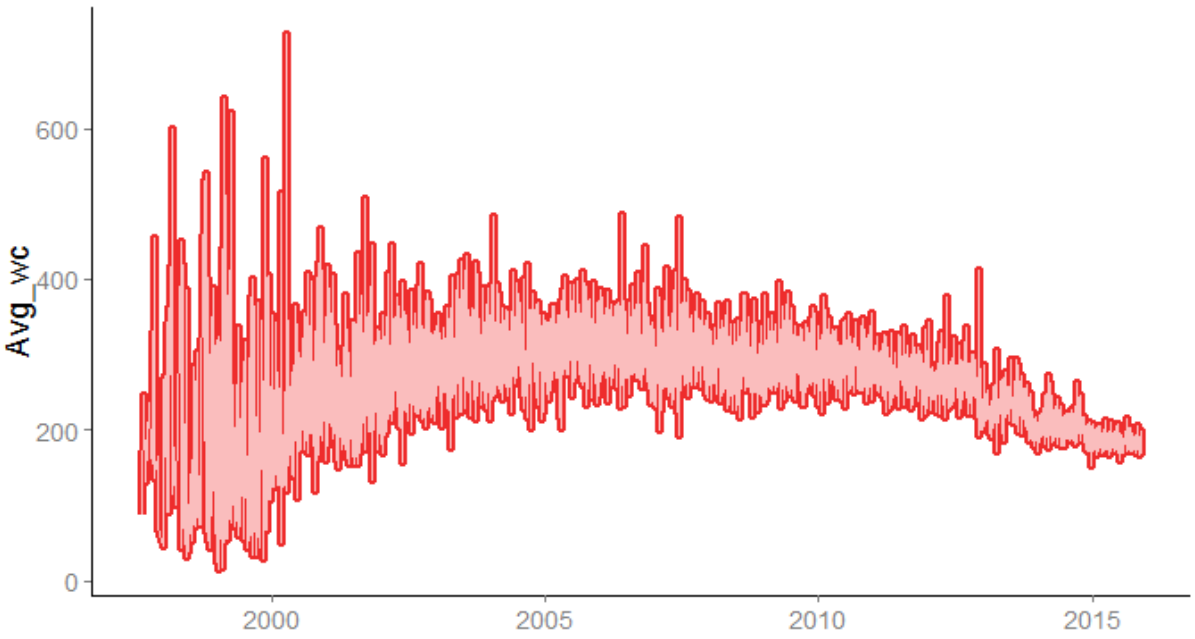
First, let's turn to the distribution of the reviews per Amazon reviewer. The average is 179 reviews per reviewer, but the distribution is highly skewed, as indicated by the plot below (log scale)



Another aspect of the data set we're interested in is the word count of the reviews. The distribution of review word count is skewed as well. The average is 229 words per review but the maximum is 5,992. Let's take a look at the plot of the distribution (log scale)



This view of the word counts does not take time into account. So, let's take a look to see if word count varies over time



As you can see, the average is volatile early on because there are not many reviews, but as time progresses, there are more reviews, so the estimate of the average is more precise. Still, taking the spread into account,

there seems to be a visible trend. Since 2005, the average length of a review has declined from roughly three hundred words to approximately two hundred words. I do not have a hypothesis of why this is the case so it remains a possible source for further research. Now, I'll turn to a quick discussion of how I accomplished the word counts. I used the `word_count` function from the `qdap` package. To make this calculation faster from the user's standpoint I used a parallell computing package, `snow`. The parallel functions are `makeSOCKcluster` and `parSapply` and I demonstrate their use below in a code snippet.

```
library(snow)
library(qdap)

# start 8 local clusters (using all 8 cores)
cl <- makeSOCKcluster(rep("localhost", 8))


# apply the word_count function in parallel
wc_par <- parSapply(cl, total_reviews$text, word_count)
```

Text Processing and Stemming

I've taken initial steps towards preparing the review texts for modeling and further text analysis. To process the data I need to convert to all lower case, remove stop words, and remove punctuation. I also would like to stem the words, which converts words to a common word-stem. For example, `think`, `thinking`, and `thinker` would all appear as `think` after stemming. With 1.7 millions reviews, averaging 229 words, there are roughly 390M words to stem. This is a large data set to process on a PC with 8Gb for RAM, so I have to turn to parallel computing again. I wrote a function, `sen_tok`, to process the text and stem each word properly. In order for the function to be called with in the parallel version of `lapply`, the required packages need to be loaded within the function.

First, here is the code for `sen_tok`

```
# define function for stemming and cleaning ---------------------------------
sen_tok <- function(sen){
  # load packages for parallel processing
  library(plyr)
  library(dplyr)
  library(lubridate)
  library(tm)
  library(SnowballC)
  library(stringr)
  library(snow)
  library(parallel)
  library(qdap)

  tok <- sen %>%
    str_to_lower() %>%
    removeWords(stopwords("en")) %>%
    removePunctuation() %>%
    str_to_lower() %>%
    removeWords(stopwords("en")) %>%
    str_split(" ") %>%
    lapply(wordStem) %>%
    lapply(function(x) str_c(x, collapse = " ")) %>%
    unlist() %>%
```

```
    removeWords(stopwords("en"))

  return(tok)
  x_samp <- sample(100, 1)
  if(x_samp < 11) invisible(gc())
}
```

Now, let's illustrate how to call this function within a parellel process

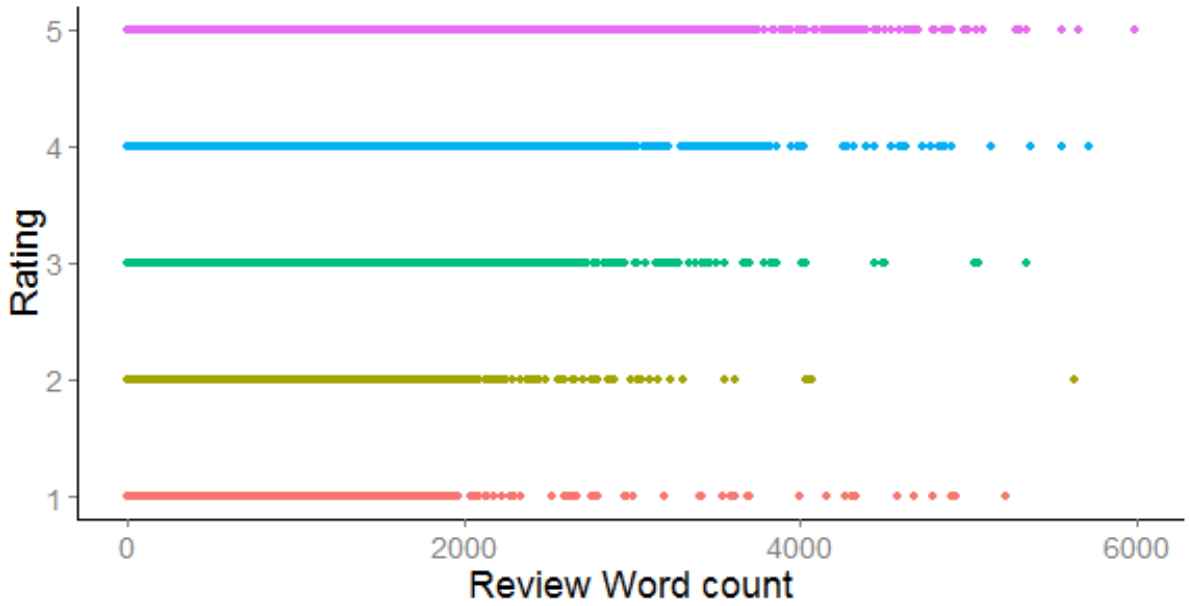
```
# get word count using parallel computing
cl <- makeSOCKcluster(rep("localhost", 8))



review_text_test <- parLapply(cl, review_text, sen_tok) %>%
  unlist()
```

Ratings vs. Word Count

Earlier I mentioned a relationship between the rating given by an Amazon reviewer and how many words the reviewer uses to describe the product. Judging from the plot below, the relationship appears to be positive, where longer reviews are associated with higher ratings.



Conclusions

To work with such a large dataset, R, running on local machine becomes very limited in terms of analysis. This project is just as much about the technology to support the analysis as it is the analysis itself. I will need to consider using a remote machines to process the data and/or a new language such as Python. I most likely will use Amazon Web Services (AWS), the EC2 product line specifically, to process and model the data. I have learned more about **R** during this project than I have during any other project I've had during graduate school. Needless to say, I really enjoyed this challenge and the problem solving required to gather millions of records from remote servers.

Next Steps

Use Amazon's Product Advertising API to get Product information which was not gathered during web scraping, see `amazon_prod_api.R` in the Appendix section.

Appendix: Code

reviewers.R

This script defines the functions which are later used to procure the list of Top Reviewers and the links to their reviews.

```
# load required packages
amazon_packages <- function() {

  # dplyr for data frame manipulation
  if(require(dplyr) == F) {install.packages("dplyr")
    library(dplyr)}

  # for string manipulation and regex
  if(require(stringr) == F) {install.packages("stringr")
    library(stringr)}

  # html gathering and node selection
  if(require(rvest) == F) {install.packages("rvest")
    library(rvest)}

  if(require(RCurl) == F) {install.packages("RCurl")
    library(RCurl)}

  if(require(httr) == F) {install.packages("httr")
    library(httr)}
}


# grab html values: links, text, and attributes
review_href <- function(page, pause = .1) {

  # grab html
  html <- html(page)    #httr::GET(page, use_proxy("198.108.245.243", 8080)))


  # get the href value
  review_links <- html %>%
    html_nodes(xpath = "//tr[contains(@id, 'reviewer')]/*/div/a") %>%
    html_attr("href") %>%
    as.data.frame.character(stringsAsFactors = F)


  # name used for url building
  url_name <- html %>%
    html_nodes(xpath = "//tr[contains(@id, 'reviewer')]/td[2]/a[2]") %>%
    html_attr("name") %>%
    as.data.frame.character(stringsAsFactors = F)


  # user name
  user_name <- html %>%
    html_nodes(xpath = "//tr[contains(@id, 'reviewer')]/td[3]/a/b") %>%
```

```
    html_text() %>%
    as.data.frame.character(stringsAsFactors = F)


  # get number of reviews, votes, and % helpful
  metrics <- html %>%
    html_nodes(xpath = "//tr[contains(@id, 'reviewer')]/td[contains(@class, 'crNum')]") %>%
    html_text() %>%
    matrix(nrow = 10, byrow = T) %>%
    as.data.frame.matrix(stringsAsFactors = F)


  # rename the columns to meanful names
  colnames(metrics) <- c("Rank", "Reviews", "Helpful_votes", "%_helpful")


  # create a larger dataframe of all the values scraped
  # need to rename the columns first
  cols <- list(url_name = url_name,
               user_name = user_name,
               review_links = review_links)


  # rename the columns and bind them together
  cols <- Map(setNames, cols, names(cols)) %>% bind_cols()


  # bind everything now
  review_links <- bind_cols(cols, metrics)

  # pause
  Sys.sleep(pause)


  return(review_links)
}


# run through number of pages
review_pages <- function(url, pages) {

  # initialize list
  links <- vector(mode = "list", length = pages)


  # get the urls
  pages <- paste(url, 1:pages, sep = "")


  # grab the links
  links <- lapply(pages, review_href)
```

```
  return(links)
}


# create url for the review page
get_review_url <- function(user, page_num){

  url <- paste("http://www.amazon.com/gp/cdp/member-reviews/",
               user,
               "?ie=UTF8&display=public&page=",
               page_num,
               "&sort_by=MostRecentReview",
               sep = "")
  return(url)
}


# create all initial last page url's: estimation
last_page <- function(reviews){

  # starting page number
  page_num <- mapply(floor, reviews/10)

  return(page_num)
}
```

reviewers_script.R

This script calls the functions defined in the previous section.

```
# set working directory and source functions
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/3_Code/R")
source("reviewers.R")

# load the required packages
amazon_packages()


# time the process
start <- Sys.time()


# where do we want to start
pages <- "http://www.amazon.com/review/top-reviewers/ref=cm_cr_tr_link_2?ie=UTF8&page="


# grab Reviewer's review metrics
pages <- review_pages(pages, 1000)


# bind each data frame together: output should have 10,000 distinct url_names
pages <- bind_rows(pages)


# finish timer
(end <- Sys.time() - start)



# Convert numeric ---------------------------------
pages$Reviews <- pages$Reviews %>%
  str_replace_all(",", "") %>%
  as.numeric()


pages$Helpful_votes <- pages$Helpful_votes %>%
  str_replace_all(",", "") %>%
  as.numeric()


pages$X._helpful <- pages$X._helpful %>%
  str_replace_all("%","") %>%
  as.numeric()


# get the last page of reviews (estimated)
last_page <- last_page(pages$Reviews)
last_page <- data.frame(last_page = last_page)

# bind with pages
```

```
pages <- bind_cols(pages, last_page)
```

data_last_step.R

Functions to create list review page url's and HTML parsing for each component of a review.

```
# load packages ------------------------------------------------------------
library(stringr)
library(lubridate)
library(plyr)
library(dplyr)
library(rvest)
library(httr)
library(ggplot2)


# grab the links and last page of each reviewer ----------------------------
# from the number above, decide the last page to peruse of reviews ...
# ... either 10 or floor(num reviews/10)
.create_link <- function(user, page){

  pages <- seq.int(page)
  base_url <- "http://www.amazon.com/gp/cdp/member-reviews/"
  base_url <- str_c(base_url, user)
  base_url <- str_c(base_url, "?ie=UTF8&display=public&page=")
  base_url <- paste(base_url, pages, sep = "")
  base_url <- paste(base_url, "&sort_by=MostRecentReview", sep = "")

  return(base_url)
}


# using the page number and link function above, perform for all users
.create_links <- function(users, pages) {

  # create list of pages to scrape for each person
  links <- Map(function(x, y) .create_link(x, y),
               x = users,
               y = pages) %>%
    unlist()

  # get the url names
  url_name <- Map(rep, users, pages) %>%
    unlist()


  # create a data.frame
  links <- data.frame(URLs = links,
                      reviewer = url_name,
                      stringsAsFactors = F,
                      row.names = NULL)

  # return the data.frame
  return(links)
}
```

```r
# Reviews ----------------------------------------------------------------------
# review text
.get_review_text <- function(html) {

  # grab the text
  text <- html %>%
    html_nodes(xpath = "//div[@class = 'reviewText']") %>%
    html_text()


  # video javascript garbage?
  JS <- str_detect(text, "amznJQ.onReady")


  # have the Length:: element?
  v_length <- str_detect(text, "Length::")


  # js text with length
  js_text <- text[JS & v_length] %>%
    str_replace_all("\\n", "") %>%
    str_split("Length::") %>%
    sapply(`[[`, 2)


  # js text with no length
  text <- str_replace_all(text,
                          "\\n",
                          "")


  # piec text back together
  clean_text <- ifelse(JS & v_length, js_text, text)


  # return text
  return(clean_text)
}


# review rating
.get_review_rating <- function(html){

  # create xpaths
  xpath <- "//span[@style='margin-left: -5px;']/img"


  # grab rating
  rating <- html %>%
    html_nodes(xpath = xpath) %>%
    html_attr("title")
```

```
  # return the review rating
  return(rating)
}


# review Date
.get_review_date <- function(html){

  # get the date of review
  date <- html %>%
    html_nodes("nobr") %>%
    html_text()


  # return the review rating
  return(date)
}


# review product id
.get_review_pid <- function(html){

  # defin the xpath: where is the information located ...
  # ... in the htlm doc?
  xpath <- "'padding-top: 10px; clear: both; width: 100%;'" %>%
    str_c("//div[@style=", ., "]/a[1]", sep = "")


  # get the product id as listed in the review
  p_id <- html %>%
    html_nodes(xpath = xpath) %>%
    html_attr("href")


  # return the product id
  return(p_id)
}


# review id
.get_review_id <- function(html){

  # define xpath
  xpath <- "//td[@colspan='7' and @align='left' and @class='small']" %>%
    str_c("/a[1]")


  # get the review id
  review_id <- html %>%
    html_nodes(xpath = xpath) %>%
    html_attr("name")
```

```
  # return the review id
  return(review_id)
}


# primary function for review page ----------------------------------------
get_page <- function(link , user){

  # rate throttle timer
  pause <- runif(1, 0, .75)


  # sleep system
  Sys.sleep(pause)


  # parse get the html doc from the link provided
  try({
    html <- read_html(link)
  },
  silent = T)



  # create list of review components
  review_components <- tryCatch(
    {
      text <- .get_review_text(html)
      rating <- .get_review_rating(html)
      date <- .get_review_date(html)
      p_id <- .get_review_pid(html)
      id <- .get_review_id(html)

      comps <- list(text = text,
                    rating = rating,
                    date = date,
                    p_id = p_id,
                    id = id)
    },
    error = function(cond){

      text <- character(0)
      rating <- character(0)
      date <- character(0)
      p_id <- character(0)
      id <- character(0)

      comps <- list(text = text,
                    rating = rating,
                    date = date,
                    p_id = p_id,
                    id = id)
```

```
    return(comps)
  },
  warning = function(cond){

  }
)


# get length of each component
i_length <- sapply(review_components, length)


# how many reviews should I have
n_reviews <- i_length  %>%
  max()


# are they all the same length?
complete_reviews <- i_length %>%
  all(. == n_reviews)


# are we banned, indicated by empty components
banned <- length(i_length[i_length == 0]) > 1

text <- review_components$text
rating <- review_components$rating
date <- review_components$date
p_id <- review_components$p_id
id <- review_components$id


# create the data frame we'll use later
tryCatch(
  {
    df <- data.frame(text = text,
                     rating = rating,
                     review_date = date,
                     product_id = p_id,
                     review_id = id,
                     reviewer = user,
                     review_page = link,
                     trouble = "correct",
                     stringsAsFactors = F
    )


    # return the succesful data frame
    return(df)
  },
  error = function(cond) {

    # why did we get the error
```

```
      if(banned){
        banned <- "banned"
      } else {
        banned <- "diff_length"
      }


      # create the contents of failed data frame
      e_rror <- c(i_length["text"],
                  i_length["rating"],
                  i_length["date"],
                  i_length["p_id"],
                  i_length["id"],
                  user,
                  link,
                  banned)


      # convert contents to dataframe
      df <- data.frame(t(e_rror),
                       stringsAsFactors = F)


      # rename to same names as succesful data frame
      colnames(df) <- c("text",
                        "rating",
                        "review_date",
                        "product_id",
                        "review_id",
                        "reviewer",
                        "review_page",
                        "trouble")


      # return unsuccessful data and appropriate error: ...
      # ... either banned or lengths of the attempted components
      return(df)
    },
    warning = function(cond) {
      message(paste("URL caused a warning:", link))

    }
  )
}
```

call_last_step.R

This script calls the functions from `data_last_step.R` and perform exploratory analysis of the data, including plots.

```
# where are my functions ----------------------------------------------------
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/3_Code/R")
source("data_last_step.R")


# load reviews list ---------------------------------------------------------
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data")


# load the list of top reviewers --------------------------------------------
reviewers <- read.csv("reviewers_list.csv", stringsAsFactors = F)


# what page to scrape to for reviews
reviewers <- reviewers %>%
  mutate(
    page_num = ceiling(Reviews/10),
    page_num = ifelse(page_num <= 10, page_num, 10)
  )


# grab the url for each person I'm assigned
reviewers <- reviewers %>%
  filter(page_num > 0)


# create the review links
review_links <- .create_links(reviewers$url_name,
                              reviewers$page_num)


# call and store the results ------------------------------------------------
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data/new_data")

start <- Sys.time()
indices <- seq(40001, 45000, by = 500)
times <- c(1:length(indices))
j <- 0

for(i in indices){
  start_loop <- Sys.time()
  Sys.sleep(rexp(1, 5))

  index_df <- seq(from = i,
                  to = i + 499)

  links <- review_links$URLs[index_df]
  users <- review_links$reviewer[index_df]
```

```
  test2 <- Map(get_page,
               as.list(links),
               as.list(users))


  test2 <- bind_rows(test2)

  file_name <- str_c(i,".csv")

  write.csv(test2, file_name, row.names = F)
  rm(test2)

  end_loop <- Sys.time() - start_loop
  j <- j + 1
  times[j] <- end_loop
}

end <- Sys.time() - start


# load one master file and save ---------------------------------------------
# list the files
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data/data_aws")
files <- dir(getwd(), pattern = "\\.csv")

start <- Sys.time()
# read them in, return as data.frame
review_large <- suppressWarnings(ldply(files,
                                       read.csv,
                                       stringsAsFactors = F)
                                 )


# filter down before saving back to disk
review_large_correct <- review_large %>%
  filter(trouble == "correct")


# look at review distribution
cust_rev <- review_large_correct %>%
  group_by(reviewer) %>%
  summarise(
    count = n()
  )


# rename
total_reviews <- review_large_correct


# rm everything except total reviews, just in case, clear garbage ...
# ... clear console, and get time
rm(list = setdiff(ls(), c("total_reviews", "start")))
```

```
gc()
cat("\014")
(end <- Sys.time() - start)


# ratings
total_reviews <- total_reviews %>%
  mutate(
    rating = str_extract(rating, "\\d{1}\\.?\\d{1}"),
    rating = as.numeric(rating)
  )


# product ID
total_reviews <- total_reviews %>%
  mutate(
    product_id = str_extract(product_id,
                             "(?<=ASIN\\=).*(?=\\#wasThisHelpful)"),
    product_id = str_c(product_id, "remove after loading")
  )


# date
total_reviews <- total_reviews %>%
  mutate(
    review_date = guess_formats(review_date, "Bdy") %>%
      as.Date(review_date, format = .)
  )


# take only the unique reviews
total_reviews <- total_reviews %>%
  distinct(review_id)


# take a quick look at a sample
total_reviews_sample <- total_reviews %>%
  sample_n(100)


# unique products
unique_products <- unique(total_reviews$product_id)


# look at how many reviews over time of the top reviewers
reviews_by_date <- total_reviews %>%
  group_by(review_date) %>%
  summarise(
    review_count = n()
  )


# plot the results
```

```r
plot(review_count ~ review_date,
     reviews_by_date,
     pch = 20,
     col = rgb(red = 0.2,
               green = 0.2,
               blue = 0.5,
               alpha = 0.5)
     )


# look at april 5, 2013
april_5_13 <- total_reviews %>%
  filter(review_date == "2013-04-05")


# what product was most reviewed on this date
april_5_13_prod <- april_5_13 %>%
  group_by(product_id) %>%
  summarise(
    review_count = n(),
    reviewers = n_distinct(reviewer),
    avg_rating = mean(rating, na.rm = T) %>%
      round(2)
  )


# load review list and see how many of the top reviewers were accounted for
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data")


# read file
reviewers <- read.csv("reviewers_list.csv", stringsAsFactors = F)


# grab the url for each person I'm assigned
reviewers <- reviewers %>%
  filter(Reviews > 0)


# how many made the cut
reviewers_retrieved <- intersect(reviewers$url_name, total_reviews$reviewer) %>%
  length()/nrow(reviewers)
reviewers_retrieved <- round(reviewers_retrieved, 2)


# who had the most reviews on any single day
most_reviews <- total_reviews %>%
  group_by(reviewer, review_date) %>%
  summarise(
    review_count = n()
  ) %>%
  ungroup() %>%
  filter(review_count == max(review_count))
```

```r
# get the daily review statistics
most_reviews_daily <- total_reviews %>%
  group_by(reviewer, review_date) %>%
  summarise(
    review_count = n()
  ) %>%
  ungroup()


# plot the results of daily review stats
ggplot(most_reviews_daily, aes(x = review_count)) +
  scale_x_log10() +
  geom_histogram(alpha = .8,
                 binwidth = 0.075,
                 fill = "slateblue") +
  geom_vline(data = most_reviews_daily,
             aes(xintercept = mean(review_count)),
             colour = "orange") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  labs(
    x = "Daily Review Count",
    y = "Total Reviewers"
  ) +
  theme(text = element_text(size = 18))


# time series of daily reviews
ggplot(most_reviews_daily, aes(as.Date(review_date), review_count)) +
  geom_line(aes(group = reviewer), alpha = 0.5) +
  geom_smooth(aes(group = 1),
              size = 1.5) +
  xlab("") +
  ylab("Daily Reviews") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  theme(text = element_text(size = 18))


# get the total reviews for each reviewer
reviews_per_reviewer <- total_reviews %>%
  group_by(reviewer) %>%
  summarise(
    review_count = n()
  )
```

```r
# join this with the number of reviews listed on the site
reviews_per_reviewer <- left_join(reviews_per_reviewer,
                                  reviewers[c("url_name", "Reviews")],
                                  by = c("reviewer" = "url_name"))


# create % scraped
reviews_per_reviewer <- reviews_per_reviewer %>%
  mutate(
    perc_retrieved = review_count / Reviews,
    perc_retrieved = round(perc_retrieved, 2)
  )


# histogram for reviews per review
ggplot(reviews_per_reviewer, aes(x = review_count)) +
  scale_x_log10() +
  geom_histogram(alpha = .8,
                 binwidth = 0.015,
                 fill = "olivedrab3") +
  geom_vline(data = reviews_per_reviewer,
             aes(xintercept = mean(review_count)),
             colour = "orange") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  labs(
    x = "Review Count",
    y = "Total Reviewers"
  ) +
  theme(text = element_text(size = 24))


# Clean text field ----------------------------------------------------------
# does the text field contain javascript
total_reviews <- total_reviews %>%
  mutate(
    javascript = str_detect(text, "amznJQ.onReady")
  )


# take a quick look at a sample
total_reviews_sample <- total_reviews %>%
  sample_n(100)


# filter only the those that contain javascript
total_reviews_javascript <- total_reviews %>%
  filter(javascript) %>%
  mutate(
    text = str_replace_all(text, ".*(?=\\}\\}\\)\\;\\}\\)\\;)", "") %>%
```

```
      str_replace_all("(\\}\\}\\)\\;\\}\\)\\;)", "")
  )


# I want cleaned javascript text binded with rest of reviews
total_reviews <- total_reviews %>%
  filter(!javascript) %>%
  bind_rows(total_reviews_javascript)


# video, yes or no
start <- Sys.time()
total_reviews$vid <- sapply(total_reviews$text,
                            str_detect,
                            "\\d{1,2}:\\d{1,2}\\s?Mins")
(end <- Sys.time() - start)


# load again to do some plotting ---------------------------------------------
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data")


# load
total_reviews <- read.csv("total_reviews_aws.csv",
                          stringsAsFactors = F,
                          nrows = 10)


# get classes to use when loading
review_col_classes <- sapply(total_reviews, class)


# reload
start <- Sys.time()
total_reviews <- read.csv("total_reviews_aws.csv",
                          stringsAsFactors = F,
                          colClasses = review_col_classes)
(end <- Sys.time() - start)

# change date
total_reviews$review_date <- ymd(total_reviews$review_date)


# get the daily review statistics
most_reviews_daily <- total_reviews %>%
  group_by(reviewer, review_date) %>%
  summarise(
    review_count = n()
  ) %>%
  ungroup()


# plot the results of daily review stats
```

```r
ggplot(most_reviews_daily, aes(x = review_count)) +
  scale_x_log10() +
  geom_histogram(alpha = .8,
                 binwidth = 0.075,
                 fill = "slateblue") +
  geom_vline(data = most_reviews_daily,
             aes(xintercept = mean(review_count)),
             colour = "orange") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  labs(
    x = "Daily Review Count",
    y = "Total Reviewers"
  ) +
  theme(text = element_text(size = 18))


# time series of daily reviews
p <- ggplot(most_reviews_daily, aes(as.Date(review_date), review_count)) +
  geom_line(aes(group = reviewer), alpha = 0.1) +
  xlab("") +
  ylab("Daily Reviews") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  theme(text = element_text(size = 18))


# plot time series
p + stat_summary(fun.y = mean, geom = "line", colour = "red")


# get the monthly review statistics
most_reviews_month <- total_reviews %>%
  mutate(
   year = year(review_date),
   month = month(review_date)
  ) %>%
  group_by(reviewer, year, month) %>%
  summarise(
    review_count = n()
  ) %>%
  ungroup() %>%
  mutate(
    yr_mo = paste(year, month, sep = "-")
  )
```

```r
# time series of monthly reviews
p <- ggplot(most_reviews_month, aes(yr_mo, review_count)) +
  geom_line(aes(group = reviewer), alpha = 0.1) +
  xlab("") +
  ylab("Monthly Reviews") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  theme(text = element_text(size = 18))


# plot time series
p + stat_summary(fun.y = mean, geom = "line", colour = "red")


# get the total reviews for each reviewer
reviews_per_reviewer <- total_reviews %>%
  group_by(reviewer) %>%
  summarise(
    review_count = n()
  )


# join this with the number of reviews listed on the site
reviews_per_reviewer <- left_join(reviews_per_reviewer,
                                  reviewers[c("url_name", "Reviews")],
                                  by = c("reviewer" = "url_name"))


# create % scraped
reviews_per_reviewer <- reviews_per_reviewer %>%
  mutate(
    perc_retrieved = review_count / Reviews,
    perc_retrieved = round(perc_retrieved, 2)
  )


# histogram for reviews per review
ggplot(reviews_per_reviewer, aes(x = review_count)) +
  scale_x_log10() +
  geom_histogram(alpha = .8,
                 binwidth = 0.015,
                 fill = "olivedrab3") +
  geom_vline(data = reviews_per_reviewer,
             aes(xintercept = mean(review_count)),
             colour = "orange") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
```

```
  labs(
    x = "Review Count",
    y = "Total Reviewers"
  ) +
  theme(text = element_text(size = 24))


# get the unique products and save them to disk
products_reviewed <- total_reviews %>%
  select(product_id) %>%
  distinct()
```

text_mining.R

This script performs fundament text mining — stemming and word counts — in addition to further exploratory analysis.

```
# load packages ------------------------------------------------------------
library(plyr)
library(dplyr)
library(lubridate)
library(tm)
library(SnowballC)
library(stringr)
library(snow)
library(parallel)
library(qdap)
library(ggplot2)


# load the data and review length: wc and char count -----------------------
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data")

start <- Sys.time()
# load
total_reviews <- read.csv("total_reviews_aws.csv",
                          stringsAsFactors = F,
                          nrows = 10)


# get classes to use when loading
review_col_classes <- sapply(total_reviews, class)


# reload
total_reviews <- read.csv("total_reviews_aws.csv",
                          stringsAsFactors = F,
                          colClasses = review_col_classes)


# change date
total_reviews$review_date <- ymd(total_reviews$review_date)


# get string length
start <- Sys.time()
total_reviews$review_length <- total_reviews$text %>%
  lapply(str_length) %>%
  unlist()
(end <- Sys.time() - start)


# get word count using parallel computing
cl <- makeSOCKcluster(rep("localhost", 8))
wc_par <- parSapply(cl, total_reviews$text, word_count)
(end <- Sys.time() - start)
```

```r
# add word count to total reviews
total_reviews$wc_review <- wc_par


# remove cluster, clear garbage and console
rm(cl, wc_par);gc();cat("\014")


# define function for stemming and cleaning ---------------------------------
sen_tok <- function(sen){
  # load packages for parallel processing
  library(plyr)
  library(dplyr)
  library(lubridate)
  library(tm)
  library(SnowballC)
  library(stringr)
  library(snow)
  library(parallel)
  library(qdap)

  tok <- sen %>%
    str_to_lower() %>%
    removeWords(stopwords("en")) %>%
    removePunctuation() %>%
    str_to_lower() %>%
    removeWords(stopwords("en")) %>%
    str_split(" ") %>%
    lapply(wordStem) %>%
    lapply(function(x) str_c(x, collapse = " ")) %>%
    unlist() %>%
    removeWords(stopwords("en"))

  return(tok)
  x_samp <- sample(100, 1)
  if(x_samp < 11) invisible(gc())
}


# get the reviews and remove the data frame due to memory constraints -------
review_text <- total_reviews$text
names(review_text) <- total_reviews$review_id # in case I need the review id

# remove anything except sentok function and review_text
rm(list = setdiff(ls(), c("review_text", "sen_tok")))


# garbage collection and clear console
gc();cat("\014")


# apply the function to the review text -------------------------------------
```

```r
# stem and clean
# get word count using parallel computing
cl <- makeSOCKcluster(rep("localhost", 8))
start <- Sys.time()
review_text_test <- parLapply(cl, review_text, sen_tok) %>%
  unlist()
(end1 <- Sys.time() - start)

rm(cl, review_text);gc()


# reload all columns except for text ----------------------------------------
total_reviews <- read.csv("total_reviews_aws.csv",
                          stringsAsFactors = F,
                          nrows = 10)

# get the classes
col_classes <- sapply(total_reviews, class)
col_classes[1] <- "NULL"


# skip text field because I'm going to replace with it with the cleaned ...
# ... stemmed text field
total_reviews <- read.csv("total_reviews_aws.csv",
                          stringsAsFactors = F,
                          colClasses = col_classes)


# add the stemmed text and save to disk
total_reviews$text <- review_text_test


# rerrange for consistency
total_reviews <- total_reviews %>%
  select(text, rating:review_length)


# plots of word count -------------------------------------------------------
# reload
setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data")
total_reviews <- read.csv("total_reviews_stem.csv",
                          stringsAsFactors = F,
                          nrows = 10)


# get the classes
col_classes <- sapply(total_reviews, class)


# use classes to load
total_reviews <- read.csv("total_reviews_stem.csv",
                          stringsAsFactors = F,
                          colClasses = col_classes)
```

```r
# convert date
total_reviews$review_date <- as.Date(ymd(total_reviews$review_date))


# histogram
ggplot(total_reviews, aes(x = wc_review)) +
  scale_x_log10() +
  geom_histogram(alpha = .9,
                 binwidth = 0.05,
                 fill = "tomato") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  labs(
    x = "Word Count",
    y = "Total Reviews"
  ) +
  theme(text = element_text(size = 18))


# get average word count by day
wc_by_day <- total_reviews %>%
  group_by(review_date) %>%
  summarise(
    avg_wc = mean(wc_review, na.rm = T),
    avg_wc = round(avg_wc, 2)
  ) %>%
  ungroup()


# timeseries
ggplot(wc_by_day, aes(review_date, avg_wc)) +
  geom_line(size = 2, colour = "firebrick2") +
  geom_line(size = 0.01, colour = "snow1", alpha = 0.7) +
  xlab("") +
  ylab("Avg_wc") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  theme(text = element_text(size = 18))


# get average rating by day
rating_by_day <- total_reviews %>%
  group_by(review_date) %>%
  summarise(
    avg_rating = mean(rating, na.rm = T),
    avg_rating = round(avg_rating, 2)
```

```
  ) %>%
  ungroup()


# plot avg rating by day
ggplot(rating_by_day, aes(review_date, avg_rating)) +
  geom_line(size = 2, colour = "olivedrab3") +
  geom_line(size = 0.01, colour = "snow1", alpha = 0.7) +
  xlab("") +
  ylab("Avg_rating") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  theme(text = element_text(size = 18))


# histogram of rating
ggplot(total_reviews, aes(x = rating)) +
  geom_histogram(alpha = .9,
                 binwidth = 0.5,
                 fill = "darkseagreen3") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  labs(
    x = "Rating",
    y = "Total Reviews"
  ) +
  theme(text = element_text(size = 18))


# review length and rating related? ----------------------------------------
# plot it
ggplot(total_reviews, aes(wc_review, rating)) +
  geom_point(aes(colour = factor(rating)))+
  ylab("Rating") +
  xlab("Review Word count") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none") +
  theme(text = element_text(size = 18))




# inputs in model
```

```
total_reviews <- total_reviews %>%
  mutate(
    year = as.factor(year(review_date)),
    vid = as.factor(vid)
  )


# build model
rating_model <- lm(wc_review ~ as.factor(rating),
                   data = total_reviews)


# get summary
mod_summary <- summary(rating_model)
```

amazon_prod_api.R

This script is still in active development but demonstrates how to interface with Amazon's Product Advertising API via **R**.

```
# load the required packages ------------------------------------------------
library(httr)
library(jsonlite)
library(lubridate)
library(plyr)
library(dplyr)
library(digest)
library(stringr)
library(RCurl)
library(data.table)
library(XML)



setwd("C:/Users/Ryan/Dropbox/RACHEL_RYAN/2_Data")

products_reviewed <- read.csv("products_reviewed.csv",
                              stringsAsFactors = F)

# remove string that protects leading zeros
products_reviewed$product_id <- products_reviewed$product_id %>%
  str_replace_all("remove after loading", "")

# test product
test_prod <- products_reviewed$product_id[1]


access_key <- "AWS Access Key here"
secret_key <- "Secret Key here"
associate_tag <- "Associate ID here"
time_stamp <- as.POSIXlt(Sys.time(), tz = "UTC") %>%
  as.character() %>%
  str_replace(" ", "T") %>%
```

```
  str_c("Z")


# end point
endpoint <- "http://ecs.amazonaws.com/onca/xml?"



nvp <- list(
  "Service" = "AWSECommerceService",
  "AssociateTag" = associate_tag,
  "Operation" = "ItemLookup",
  "Timestamp" = time_stamp,
  "AWSAccessKeyId" = access_key,
  "ItemID" = "B00008OE6I",
  "IdType" = "ASIN",
  "ResponseGroup" = "ItemAttributes"
)


bytes <- function(chr){
  as.data.frame(t(as.numeric(charToRaw(chr))))
}


b <- lapply(names(nvp), bytes)
b <- data.table::rbindlist(b, fill=TRUE)


nvp <- nvp[do.call(order, as.list(b))]
nvp <- sapply(nvp, URLencode, reserved = T)
nvp <- paste0(names(unlist(nvp)), "=",unlist(nvp))
nvp <- str_c(nvp, collapse = "&") %>% URLencode()

sign_begin <- "GET\necs.amazonaws.com\n/onca/xml"

query <- str_c(sign_begin, nvp, sep = "\n")

signature_api <- hmac(secret_key, query, "sha256")

url <- str_c(endpoint, nvp, "&Signature=", signature_api)

get_test <- GET(url)

get_test$content %>% rawToChar()

content_raw <- rawToChar(get_test$content)

content_clean <- xmlTreeParse(content_raw)



# https://associates-amazon.s3.amazonaws.com/signed-requests/helper/index.html
```

```
# this is the input
# http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService
# &AssociateTag=prodreview04c-20
# &Operation=ItemLookup
# &Timestamp=2015-12-02T23:53:43Z
# &AWSAccessKeyId=[acces_key]
# &ItemId=B00008OE6I
# &IdType=ASIN
# &ResponseGroup=ItemAttributes



# use this to make sure I can get the same signature
curlEscape(
  base64(hmac(enc2utf8((secret_key)),
                        enc2utf8(string_to_sign),
                        algo = 'sha256',
                        serialize = FALSE,
                        raw = TRUE))
  )
```