REDS Midtown Tavern Online Review Analytics (WORKING TITLE)

Foundations of Data Science: Capstone Project Report

Antoine Beauchamp
March 25th, 2017

Contents

1 Introduction

("REDS Midtown Tavern," n.d.)

This is an edit comment I'm not sure if I should do proper bibliographic citations, or if I should just put hyperlinks in the text. Follow up on this.

The past decade has resulted in an explosion in the amount of data that we produce every day by way of our technological activity **cite and quantify this?**. This ever growing amount of data means that we are increasingly becoming saturated with information regarding any number of topics in a wide range of areas of human activity, from personalized healthcare to federal policy to corporate finance. Though this vast sea of data might seem daunting, with the right tools and the right approach, it can also be incredibly useful. Buried in the data is contained important information about various aspects of human and business activity. By manipulating and analyzing this data, we are able to uncover and solidify insights that previously may only have been hinted at on an intuitive level. The data, when analyzed properly, allows us to make decisions and recommendations based on concrete evidence, rather than a gut instinct or biased perception. In the context of business, this is an extremely useful advantage to leverage in order to promote growth and to allocate resources properly. One especially important source of data is the World Wide Web. The Internet provides a massive agglomeration of human opinion and behaviour from all areas of life. From a business perspective, the clientele's online behaviour often provides a wealth of information regarding patterns that may be influential to business activity. For businesses that operate within the service and hospitality industries, important data pertaining to customer sentiment is abundantly available on popular review and travel websites. In this report, I will demonstrate how we can use a variety of tools available in R to acquire and mine the data from such websites in order to uncover insights and make informed business recommendations. In particular, I will focus on gathering and analyzing the user review data for REDS Mditown Tavern, a restaurant in Toronto, Ontario. **Edit this sentence** The restaurant's website is located here: www.redsmidtowntavern.com. **Better to cite things here? Or is this fine?**

This report is divided into **???** primary sections. In Section 2, I will elaborate on the methods and tools I used to acquire customer review data for REDS using popular review websites. Once the data is collected and stored locally, I will discuss the process of transforming this raw data into a form suitable for analysis. This is done in Section 3. Finally, in Section 4, I will dive into the analysis of this customer review data in order to mine for actionable insights. Let's get started.

2 Data Acquisition

As mentioned in the Introduction, the aim of this project is to perform an analysis of online customer review data for REDS Midtown Tavern. For the purpose of this investigation, I have focussed my efforts on the following review and travel websites: https://www.yelp.com/toronto, https://www.opentable.com/toronto-restaurants, https://www.tripadvisor.ca/, and https://www.zomato.com/toronto. Though this data is readily available to see and read, we would like to acquire it in a more structured way, so as to facilitate processing and analysis. This can be done relatively simply using a variety of tools available online. More specifically, I gathered the review data from the aforementioned websites by using the `rvest` package in R (https://cran.r-project.org/web/packages/rvest/index.html) and the web scraping tool, SelectorGadget (http://selectorgadget.com/). **Proper citations here or is this fine?**

2.1 SelectorGadget and CSS Selectors

2.1.1 Identifying Content with SelectorGadget

The first step in acquiring data from the web is being able to identify exactly what content needs to be extracted. In practice, this is accomplished with the use of CSS selectors. CSS Selectors are what allows us to gather data from the web by using packages such as `rvest`. The difficulty arises in identifying which selectors are associated with what content on a webpage. Traditionally, one might attempt to identify a selector for specific content by scouring the HTML source code for a website. Though it is possible, this isn't exactly effortless. An alternative way of identifying the appropriate CSS selectors is to use a tool, such as SelectorGadget, that facilitates this process. SelectorGadget is an open source tool, developed by Hadley Wickham **cite?**, that allows us to identify CSS selectors from a website by clicking on select items of interest. SelectorGadget greatly facilitates the process of identifying CSS selectors by providing a point-and-click interface between the webpage content and the underlying selector. I will elaborate on the use of SelectorGadget to obtain CSS selectors using a relevant example.

An important first step **Used "first step" earlier in here** towards the ultimate goal of analysing online customer review data is to scrape popular websites for the customer reviews. In this example, I will work with the first page of reviews for REDS Midtown Tavern on Yelp. Here is the relevant URL: https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2. The reviews are located not too far down the page. This is displayed in Figure 1.

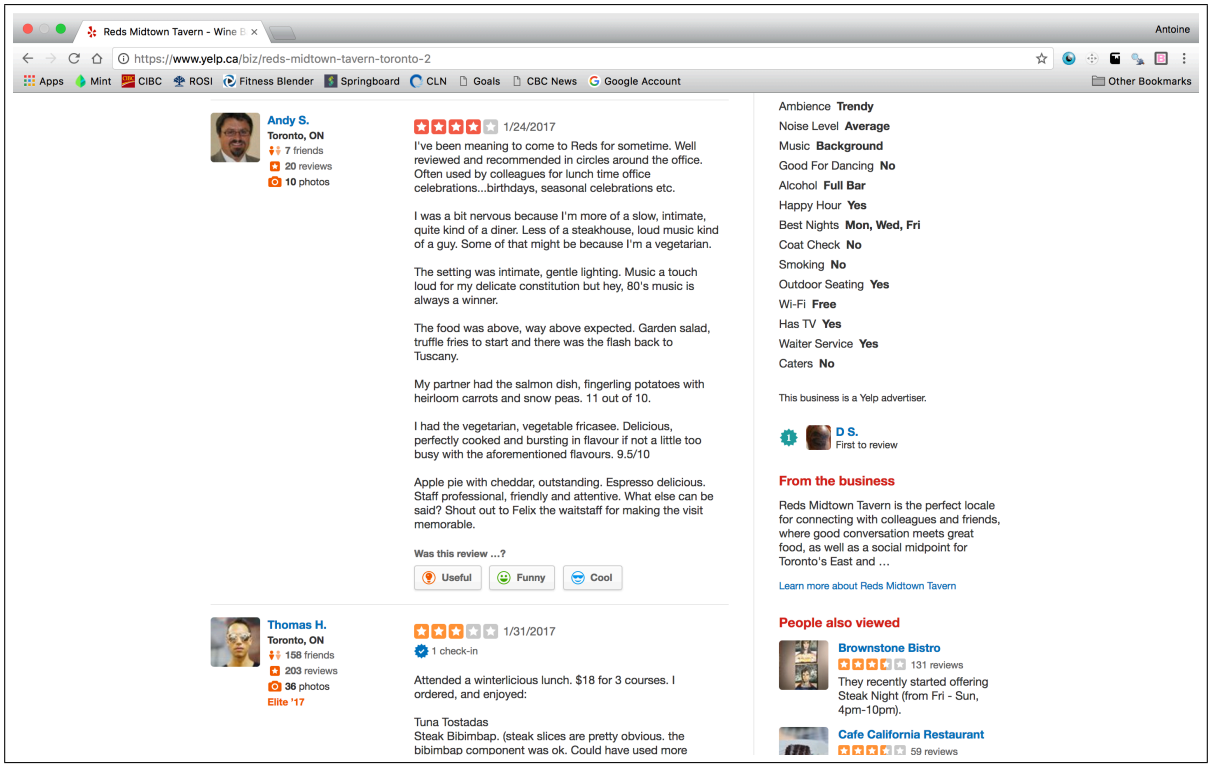


Figure 1: **Add caption**

The SelectorGadget interface manifests itself on the webpage as a grey search bar at the bottom of the page. To begin identifying the CSS Selector that corresponds to the customer review content, we can select the first customer review by clicking on it. The result of this action is shown in Figure

2.

We see that the SelectorGadget has returned the CSS Selector "`p`". The content that we have selected via point-and-click has been highlighted in green, while the content that matches that selector has been highlighted in yellow. This selector includes the customer reviews, which is desired, but it also appears to include additional content, such as the text "Was this review …?" and additional text under the heading "From the business". We don't need these elements of the webpage. This means that the selector "`p`" is returning too broad a selection. This is the initial stage in the filtering process. We can now choose to add additional content to our filter by clicking on content that is not yet highlighted, or we can remove content from our filter by clicking on content that has been highlighted. In this case, we want to click on the text below "From the business" to remove it from our selection. In doing so, we observe that content that had previously been highlighted but is now de-selected becomes red. This is shown in Figure 3. We also notice that the text "Was this review…?" is no longer included in our selection. It appears that the CSS selector "`.review-content p`" is the correct selector associated with the review content on this webpage. This can verified by noting that SelectorGadget has found 20 items matching this selector, which corresponds to the number of reviews on this page. Having identified the correct CSS Selector for the customer review content, we can import the data into R. This process will be described in detail below.

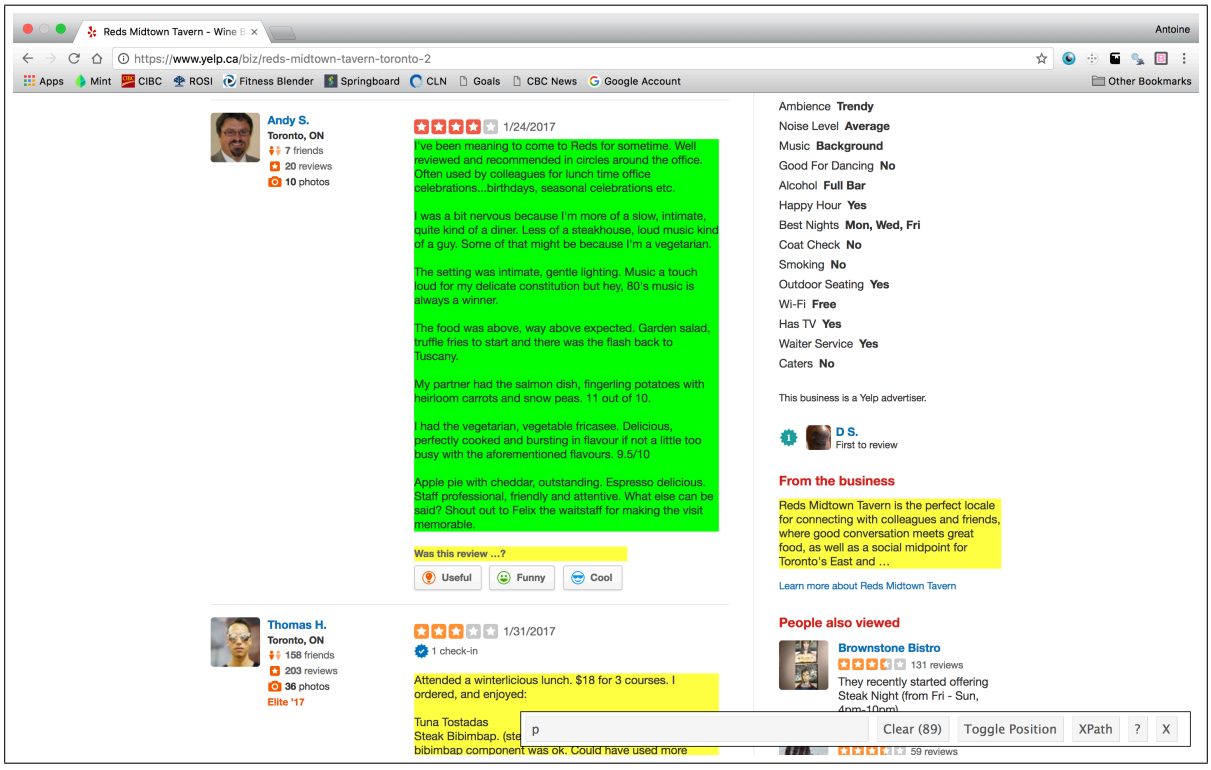


Figure 2: **Add caption**

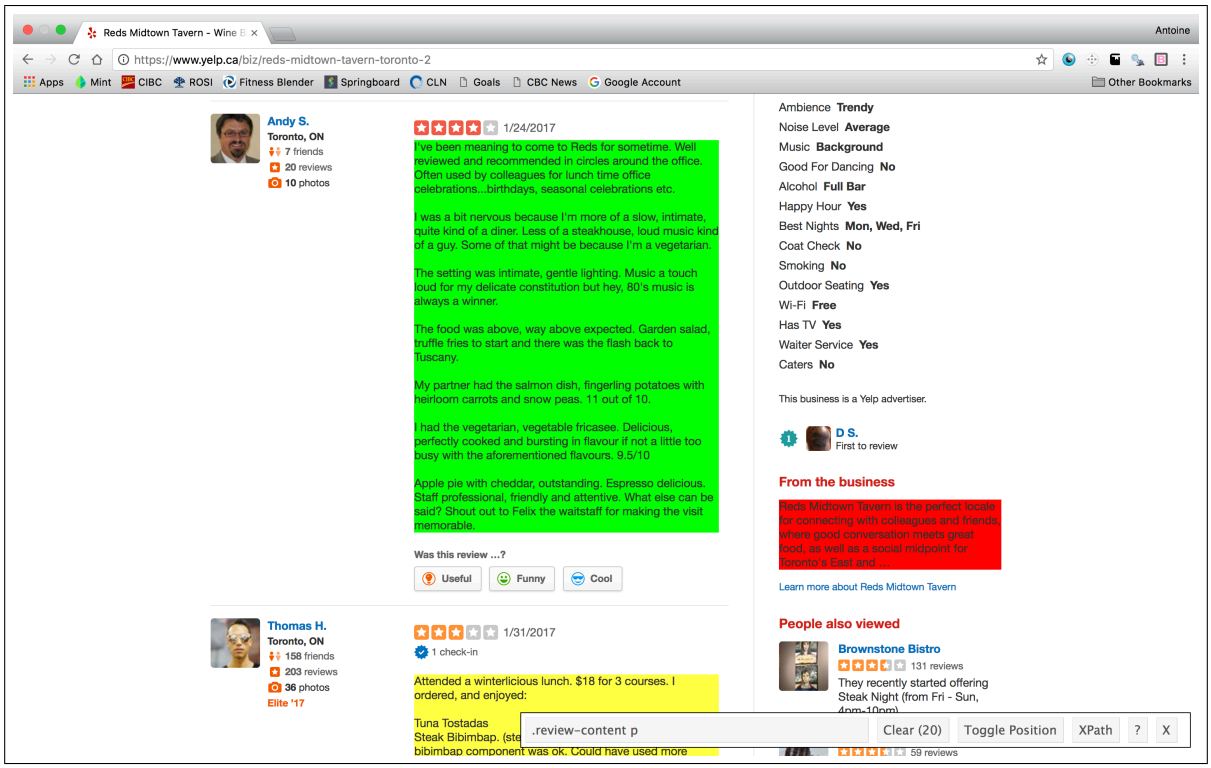


Figure 3: **Add caption**

2.1.2 CSS Selectors in Detail

When working with SelectorGadget, it is helpful to have some knowledge of CSS selectors. In essence, CSS selectors allow us to identify given "blocks" of content within an HTML source file. An excellent hands-on tutorial regarding CSS selectors is found at the following website: https://flukeout.github.io/ **Cite properly?**. This section will loosely follow the layout of this tutorial.

An HTML file is comprised of different blocks of content, which are identified by their **type**. For example, in HTML, `<p>` represents a paragraph block, `<a>` a hyperlink, `<h1>` a heading, and so on. A screenshot of the HTML source file for https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2 is included in Figure 4. To learn more about HTML formatting, visit this website: https://www.w3schools.com/html/default.asp

The most basic CSS selector is simply a selector of type. To obtain all content that is contained within a paragraph block, simply use the selector "`p`". In fact, this is what happened when we used SelectorGadget in Figure 2.2: the customer review content was contained in a parapraph block, but so was other content that we didn't need. Using the selector `p` identified all of this content.

In order to diversify content, HTML types can be associated with an **ID** or with a **class**. For example, `<div id="title">` or `<div class="section">`. These are both `<div>` blocks, but one has an ID called "title" and the other has a class called "section". In addition to type, content can be extracted from a webpage by selecting it based on ID or class. The CSS selector for an ID is a hashtag, `#`. E.g. if there exists a block defined by `<div id="title">` in the HTML file, we can select that element by using the selector `#title`. This will select all content with `id="title"`, including but not limited to the content that we want. Similarly, the selector for a class is a period, `.`. E.g. we can select `<div class="content">` using `.content`.

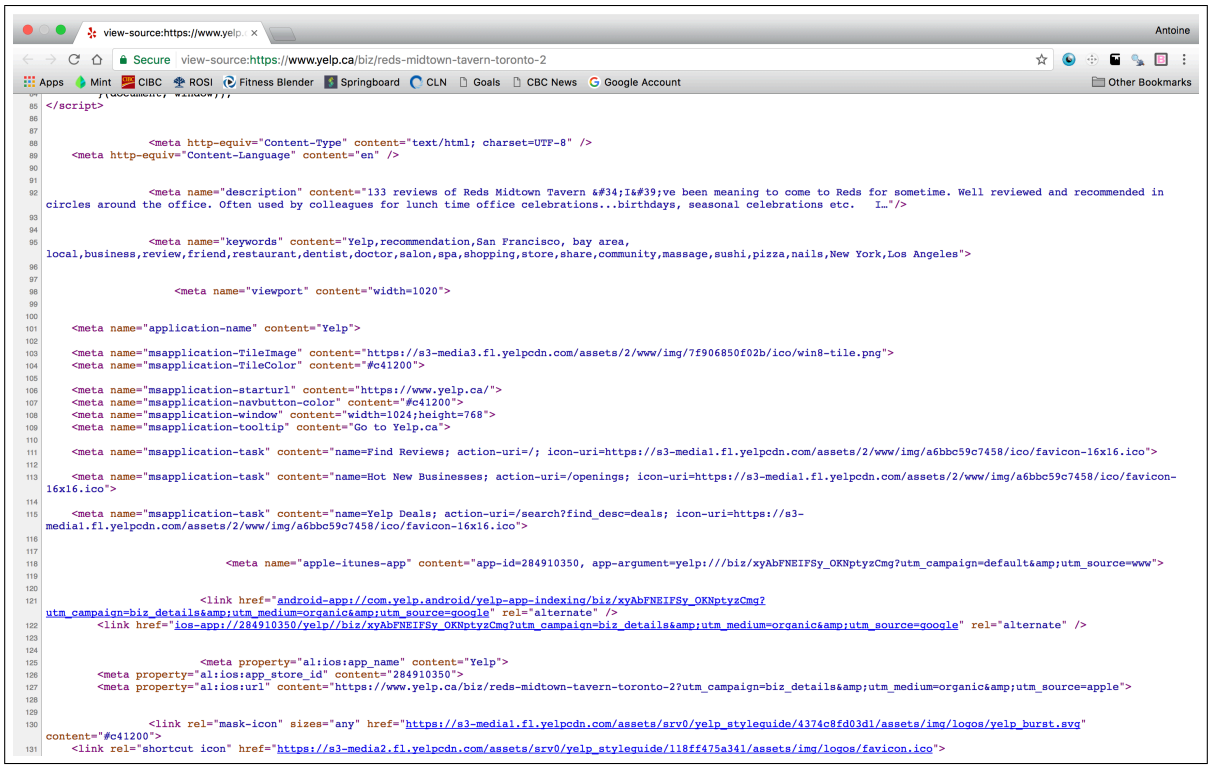


Figure 4: **Add caption**

Now that we have these basic selectors, we can combine them to access more specific content. One such way is using the **descendant selector**. This allows us to select content that is nested within a content block. The descendant selector consists of writting two selectors separated by a space. E.g. We could select `<div id="title"><p> Text </p></div>` with the selector `div p`. This selects paragraph content within a `<div>` block. We can also use the descendant selector with the ID or class selectors. E.g. `#title p` will select all paragraph content contained within a block of any type with an `id="title"`.

In Section 2.1.1, we used the selector `.review-content p` to obtain the customer review data from Yelp. With our new understanding of CSS selectors, we see that this uses a descendant selector with the class and type selectors. We are selecting the paragraph blocks, `<p>`, within blocks with `class="review-content"`.

Finally, specific class content can be selected by combining the class selector with the type selector, in the following way: `type.class`. E.g. if there exists both `<div class="section">` and `<span class="section">` withtin the HTML file, we can select the `<div>` block specifically using `div.section`, rather than just `.section`, which would additionally select the `<span>` block.

This covers the basics of CSS selectors. This should provide some context for what SelectorGadget is returning when clicking on webpage content. For a more detailed look at CSS selectors, follow the complete tutorial at https://flukeout.github.io/.

2.2 Importing Web Data with `rvest`

In the previous Section, we discussed how to work with SelectorGadget to identify content that we want to acquire from webpages. Once the correct CSS selectors are obtained for the content that we want to acquire, the next step is to load this data into R. This step of the data gathering

process was performed using Hadley Wickham's web data R package, `rvest`.

`rvest` allows us to gather data from the web using a few basic functions: `read_html()`, `html_nodes()`, `html_text()`, and `html_attrs()`.

Consider once again the problem of scraping reviews for REDS Midtown Tavern from Yelp. In Section 2.1.1, we identified that the correct CSS selector for this content was `.review-content p`. This is great, but what do we do with this? This is where `rvest` comes in.

Let's start by loading the package. If it isn't already installed, install it using `install.packages()`.

```
suppressMessages(library(rvest))
```

The first step in working with `rvest` is to provide R with the URL for the website that we want to scrape. This is done using the `read_html()` function.

```
YelpURL <- "https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2"
YelpURL_data <- read_html(YelpURL)
print(YelpURL_data)
```

```
## {xml_document}
## <html xmlns:fb="http://www.facebook.com/2008/fbml" class="no-js" lang="en">
## [1] <head>\n<script>            window.yPageStart = new Date().getTime() ...
## [2] <body id="yelp_main_body" class="jquery country-ca logged-out biz-de ...
```

This saves the information about the URL to our console. Next, we extract the content that we want using the CSS selector that we identified with SelectorGadget. Here we use `html_nodes()` with the arguments being the HTML documents and the CSS selector.

```
YelpReviews <- html_nodes(YelpURL_data, ".review-content p")
head(YelpReviews)
```

```
## {xml_nodeset (6)}
## [1] <p lang="en">The first time I started dining here was due to summerl ...
## [2] <p lang="en">Attended a winterlicious lunch. $18 for 3 courses. I or ...
## [3] <p lang="en">I've been meaning to come to Reds for sometime. Well re ...
## [4] <p lang="en">As a young barrister  I often visited Red's older, fanc ...
## [5] <p lang="en">I came here for dinner and had wine and entrees. I wish ...
## [6] <p lang="en">I had to try out Reds for Summerlicious! It was a 3 cou ...
```

We now have the customer reviews from Yelp stored in an object of class `xml_nodeset`. This isn't the easiest way to work with the data so the next step is to convert this to `character` class. The first way to do this is simply to use the `as.character()` function.

```
YelpReviews_char1 <- as.character(YelpReviews)
head(YelpReviews_char1, n = 2)
```

```
## [1] "<p lang=\"en\">The first time I started dining here was due to summerlicious. My friends and
## [2] "<p lang=\"en\">Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:<b
```

This converts all of the data to `character` class, including the HTML formatting tags. This can be useful if the formatting is needed. If we are solely interested in the text stored within the paragraph block, however, we can use `html_text()` to extract this data directly.

```
YelpReviews_char2 <- html_text(YelpReviews)
head(YelpReviews_char2, n = 2)
```

```
## [1] "The first time I started dining here was due to summerlicious. My friends and I all loved the
## [2] "Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:Tuna TostadasStea
```

In this case we have the text in character format, but without the HTML tags associated with the content. Note that `html_text()` only works if there is text to extract, obviously.

Other useful functions to extract data from these HTML nodes are the `html_attrs()` and `html_attr()` functions. Looking at `YelpReviews`, we see that the HTML tag is `<p lang="en">`. Thus the paragraph blocks are associated with an **attribute** called `lang`. We can pull all the attributes from our HTML data using `html_attrs()`.

```
head(html_attrs(YelpReviews), n = 3)
```

```
## [[1]]
## lang
## "en"
##
## [[2]]
## lang
## "en"
##
## [[3]]
## lang
## "en"
```

```
class(html_attrs(YelpReviews))
```

```
## [1] "list"
```

The output is a list containing the attribute (`lang`) and the value (`"en"`). We can further extract the value of a specific attribute using `html_attr()`.

```
head(html_attr(YelpReviews, "lang"))
```

```
## [1] "en" "en" "en" "en" "en" "en"
```

```
class(html_attr(YelpReviews, "lang"))
```

```
## [1] "character"
```

This is particularly useful when information that we want is stored within such attributes, rather than within the main content block. One example of this is the numerical rating of the restaurant, included in the customer reviews. On Yelp this is presented as a number of stars filled in with the colour orange. Let's extract this data and see what it looks like. The correct CSS selector is `.rating-large`.

```
YelpRatings <- html_nodes(YelpURL_data, ".rating-large")
as.character(YelpRatings)[1]
```

```
## [1] "<div class=\"i-stars i-stars--regular-4 rating-large\" title=\"4.0 star rating\">\n      <
```

Here we see that the value that we want, in this case "4.0 star rating", is actually contained within the `"title"` attribute of the `<div>` block. It can also be obtained from the `"alt"` attribute of the `<img>` block. Let's see what `html_attrs()` gives us.

```
head(html_attrs(YelpRatings), n = 3)
```

```
## [[1]]
##                                      class
## "i-stars i-stars--regular-4 rating-large"
##                                      title
##                         "4.0 star rating"
##
## [[2]]
##                                      class
## "i-stars i-stars--regular-3 rating-large"
##                                      title
##                         "3.0 star rating"
##
## [[3]]
##                                      class
## "i-stars i-stars--regular-4 rating-large"
##                                      title
##                         "4.0 star rating"
```

These are only the attributes of the `<div>` block. The reason for this is that our selector, `.rating-large`, only selects this block. To select the nested `<img>` block, we would use the descendant selector `.rating-large img`. From the `div` block we can extract the numeric rating by using the `title` attribute.

```
YelpRatings_clean <- html_attr(YelpRatings, "title")
head(YelpRatings_clean)
```

```
## [1] "4.0 star rating" "3.0 star rating" "4.0 star rating" "4.0 star rating"
## [5] "3.0 star rating" "3.0 star rating"
```

The next step here would be to use regular expressions to isolate the number and then convert the data to `numeric` class for quantitative analysis.

2.3 Gathering Capstone Data

With a basic understanding of CSS selectors, SelectorGadget, and `rvest`, we are equipped to gather our customer review data. As mentioned previously, the following websites were used as data sources: Yelp(https://www.yelp.com/toronto), OpenTable(https://www.opentable.com/toronto-restaurants), Trip Advisor(https://www.tripadvisor.ca/), and Zomato (https://www.zomato.com/toronto).

The data that we are going to acquire is as follows: customer reviews, customer ratings, and dates of reviews.

Note that the code presented in this section will not be executable due to the longer computation times.

Let's start by wiping our working environment and loading the necessary packages.

```
rm(list = ls())

# Load required libraries
suppressMessages(library(rvest))
suppressMessages(library(dplyr))
library(tidyr)
suppressMessages(library(readr))
```

I have coded the data gathering process into a number of functions, one for each website. The algorithm for each function is straightforward:

1. Read the HTML document using the correct URL
2. Gather review, rating and date data from the webpage. Additional data may be gathered as needed.
3. Append new data to vectors for each of the variables
4. Check to see if we have reached the end of the reviews
5. Increment counter and identify URL for next page of reviews

An important part of this process was identifying the structure of the URLs of the various review pages in order to go through them automatically. It was also important to see what data was available from each of the webpages and how to access that data.

Let's begin with Yelp.

2.3.1 Yelp Data

```
########################## Function: YelpScrape ################################ This function is
########################## to scrape review data from Yelp.com Arguments: BaseURL: URL to the fi
########################## page of reviews that you want to scrape

YelpScrape <- function(BaseURL) {

    ReviewCount <- 0  #Counter for the number of reviews. On the Yelp there are 20 per page
    # Empty character vectors for the reviews and ratings
    Reviews <- character(0)
    Ratings <- character(0)
```

```
    Dates <- character(0)
    PrevRev <- character(0)
    flag <- 1

    # Now let's iterate over the different Yelp review pages and scrape the
    # data.
    while (flag == 1) {

        # Yelp URL for the given review page
        page_url <- paste(BaseURL, "?start=", as.character(ReviewCount), sep = "")

        # Scrape the reviews and ratings from the current URL
        ReviewsNew <- read_html(page_url) %>% html_nodes(".review-content p") %>%
            html_text
        RatingsNew <- read_html(page_url) %>% html_nodes(".rating-large") %>%
            html_attr("title")
        DatesNew <- read_html(page_url) %>% html_nodes(".biz-rating-large .rating-qualifier") %
            html_text()
        PrevRevNew <- read_html(page_url) %>% html_nodes(".biz-rating-large .rating-qualifier")
            as.character()

        print(paste("Scraping Yelp page", ceiling(ReviewCount/20)))

        # Append new reviews/ratings to existing vectors
        Reviews <- c(Reviews, ReviewsNew)
        Ratings <- c(Ratings, RatingsNew)
        Dates <- c(Dates, DatesNew)
        PrevRev <- c(PrevRev, PrevRevNew)

        # Increment the review counter to move to the next page in the following
        # iteration
        ReviewCount = ReviewCount + length(ReviewsNew)

        # Loop ending condition
        flag <- if (length(ReviewsNew) == 0) {
            0
        } else {
            1
        }

    }

    return(list(Reviews = Reviews, Ratings = Ratings, Dates = Dates, PrevRev = PrevRev))
}
```

The main issue that arose when scraping Yelp was that there was a discrepancy in the number of reviews and the number of ratings. This occurs because the ratings data includes data from previous reviews that have since been updated, as well as the corresponding updated reviews. The

reviews data only picks up the new reviews. The variable `PrevRev` contains information about whether a review is considered a "previous review" or not. This will be used later on to identify which reviews are previous reviews.

2.3.2 OpenTable Data

```
########################## Function: OpenTableScrape ############################### This functi
########################## used to scrape review data from opentable.com Arguments: BaseURL: URL
########################## the review page from open table. Note that the URL must end with &pag
########################## without specifying the page number. Page number is specified in the
########################## function.


OpenTableScrape <- function(BaseURL) {

    # Parameters
    ReviewCount <- 1
    Reviews <- character(0)
    Ratings <- character(0)
    Dates <- character(0)
    flag <- 1

    while (flag == 1) {

        # Get URL for current page
        page_url <- paste(BaseURL, as.character(ReviewCount), sep = "")

        # Obtain data from page
        ReviewsNew <- read_html(page_url) %>% html_nodes("#reviews-results .review-content") %>
            html_text
        RatingsNew <- read_html(page_url) %>% html_nodes("#reviews-results .filled") %>%
            html_attr("title")
        DatesNew <- read_html(page_url) %>% html_nodes(".review-meta-separator+ .color-light") %
            html_text()

        # Append vectors
        Reviews <- c(Reviews, ReviewsNew)
        Ratings <- c(Ratings, RatingsNew)
        Dates <- c(Dates, DatesNew)

        print(paste("Scraping OpenTable page", ReviewCount))

        # Increment counter
        ReviewCount <- ReviewCount + 1

        # This condition checks whether we have reached the end of the reviews
        flag <- if (length(ReviewsNew) == 0) {
```

```
            0
        } else {
            1
        }
    }
    return(list(Reviews = Reviews, Ratings = Ratings, Dates = Dates))
}
```

The web scraping process for OpenTable is more straightforward than Yelp, though there are some hiccoughs in the data that we will clean in subsequent sections.

2.3.3 Trip Advisor Data

When scraping reviews from Trip Advisor, a difficulty arises in that the URL for each of the review pages does not appear to have an obvious pattern. In order to acquire all the reviews, I had to find a way to work around this. Luckily enough, the URL for the subsequent review page is actually contained within the HTML document of the current review page, as part of the link to the next page. Therefore to get the URL for the next page, all I had to do was identify the selector for this link, and extract the data.

Another feature that I've implemented is that the web scraping process actually begins at the landing page for REDS Midtown Tavern (https://www.tripadvisor.ca/Restaurant_Review-g155019-d5058760-Reviews-Reds_Midtown_Tavern-Toronto_Ontario.html), rather than at the first full review page. The function then jumps from this landing page to the first full review page by "clicking" on the title of the first available review. The reason for this is that, when attempting to go straight to the first full review page, I noticed that new reviews were not being included in this "first" page. The page was effectively dated to when I had first obtained the URL. Jumping from the landing page for REDS Midtown Tavern to the first full review page allows me to bypass this problem, since new reviews are included on the landing page.

```
########################## Function: TripAdScrape ################################ This function
########################## used to scrape review data from Trip Advisor Arguments: LandingURL:
########################## is the URL for the landing page of the restaurant you want to scrape
########################## It will be used to link to the full review pages

TripAdScrape <- function(LandingURL) {


    # This gets the links to the review pages, which are embedded in the review
    # titles
    ReviewTitleLink <- read_html(LandingURL) %>% html_nodes(".quote a") %>%
        html_attr("href")

    # The base URL to the first review page is
    BaseURL <- paste("https://www.tripadvisor.ca", ReviewTitleLink[1], sep = "")

    # Set parameters for data scraping.
    ReviewCount <- 1
    Reviews <- character(0)
```

```
Ratings <- character(0)
Dates1 <- character(0)
Dates2 <- character(0)
flag <- 1

while (flag == 1) {

    print(paste("Scraping Trip Advisor page", ReviewCount))

    # For the first page, the URL we want to use is just the base URL. For
    # subsequent iterations, we want to grab the hyperlink to the new page from
    # the page links in the previous page. E.g. page 1 carries a link to page 2
    # in its HTML, and so on.
    if (ReviewCount == 1) {

        page_url <- BaseURL

    } else {

        # Grab the page numbers for the links
        pagenum <- read_html(page_url) %>% html_nodes(".pageNum") %>% html_attr("data-page-
            as.numeric()
        # Grab the hyperlinks for the subsequent pages
        hyperlink <- read_html(page_url) %>% html_nodes(".pageNum") %>%
            html_attr("href") %>% as.character()
        # New URL
        page_url <- paste("https://www.tripadvisor.ca", hyperlink[pagenum ==
            ReviewCount], sep = "")

    }

    # Read in reviews and ratings from current page
    ReviewsNew <- read_html(page_url) %>% html_nodes("#REVIEWS p") %>% html_text()
    RatingsNew <- read_html(page_url) %>% html_nodes("#REVIEWS .rating_s_fill") %>%
        html_attr("alt")
    DatesNew1 <- read_html(page_url) %>% html_nodes(".relativeDate") %>%
        html_attr("title", default = NA_character_)
    DatesNew2 <- read_html(page_url) %>% html_nodes(".ratingDate") %>% html_text()


    # End loop condition
    flag <- if (length(ReviewsNew) == 0) {
        0
    } else {
        1
    }
```

```
        # Append new reviews/ratings
        Reviews <- c(Reviews, ReviewsNew)
        Ratings <- c(Ratings, RatingsNew)
        Dates1 <- c(Dates1, DatesNew1)
        Dates2 <- c(Dates2, DatesNew2)

        # Increment page count
        ReviewCount <- ReviewCount + 1

    }

    return(list(Reviews = Reviews, Ratings = Ratings, Dates1 = Dates1, Dates2 = Dates2))
}
```

In this function I have extract two sets of data for the dates, stored in `Dates1` and `Dates2`. The reason for this is that, for recent reviews, Trip Advisor presents its date information in the following form: "Dined ## days ago" or "Dined yesterday". This is not useful for analysis. However, the actual dates for these recent reviews can be obtained using the selector `.relativeDate`. The catch is that this selector does not select the dates for those older reviews that are not expressed in the form "Dined ## days ago". This format is only used to express the most recent dates. Older reviews are associated with a proper date format. So we need a combination of both the information gathered by the `.relativeDate` and `.ratingDate` selectors.

2.3.4 Zomato Data

The final website used to extract data is Zomato. The main problem I ran into with this website was that the reviews are not written onto different URL pages. Rather, the reviews are accessed via a sort of drop down menu on the landing page. Due to this, and in the interest of time, I wasn't able to extract the full set of reviews available on Zomato. Fortunately, the reviews on Zomato are fairly outdated and, though it would have been nice to have them as part of our data, the older reviews will not play a significant part in our analysis since the restaurant has changed many times since then.

```
########################## Function: ZomatoScrape ################################

ZomatoScrape <- function(BaseURL) {

    # Set parameters for data scraping.
    ReviewCount <- 1
    Reviews <- character(0)
    Ratings <- character(0)
    Dates <- character(0)
    flag <- 1

    Reviews <- read_html(BaseURL) %>% html_nodes(".rev-text-expand , .rev-text") %>%
        html_text()
    Ratings <- read_html(BaseURL) %>% html_nodes(".rev-text-expand div , .rev-text div") %>%
        html_attr("aria-label")
```

```
    Dates <- read_html(BaseURL) %>% html_nodes("time") %>% html_attr("datetime")

    return(list(Reviews = Reviews, Ratings = Ratings, Dates = Dates))

}
```

One thing to note is that, if a review on Zomato is long, it will be truncated and have an associated "read more" option on the webpage to show the full review. Reviews of this type are counted twice by SelectorGadget: once for the truncated version, and once for the full expanded version. Additionally, the format in which the ratings have been extracted is such that there will be double the amount of data, half of which consists of `NA` values. We will address these issues in the cleaning stage.

2.3.5 Web Scraping

With our web scraping functions defined, we can begin the data acquisition process. All we have to do is identify the URLs for the review pages and pass them to the functions.

```
# Yelp main review page URL
BaseURL_Yelp <- "https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2"

# OpenTable main review page URL
BaseURL_OpenTable <- "https://www.opentable.com/reds-midtown-tavern?covers=2&dateTime=2017-02-2

# Trip Advisor landing page
LandingURL_TripAd <- "https://www.tripadvisor.ca/Restaurant_Review-g155019-d5058760-Reviews-Red

# Zomato main review page URL
BaseURL_Zomato <- "https://www.zomato.com/toronto/reds-midtown-tavern-church-and-wellesley/revi

# Scrape data from websites
YelpData <- YelpScrape(BaseURL_Yelp)
OpenTableData <- OpenTableScrape(BaseURL_OpenTable)
TripAdData <- TripAdScrape(LandingURL_TripAd)
ZomatoData <- ZomatoScrape(BaseURL_Zomato)
```

Before we save our raw data to file, we need to do some basic cleaning of the Open Table date data. The reason for this is that some of the dates are expressed in "Dined ## days ago" format. We don't have the proper date data, so we have to create it by subtracting the number of days from the current date. This only works if we do this on the same day that we scraped the web data.

```
# Find all instances of dates in the form 'Dined ## days ago'
DatesLogic <- grepl("[0-9]+.*ago", OpenTableData$Dates)

# Subset date info to get the instances matching the above format
DatesTemp <- OpenTableData$Dates[DatesLogic]

# Create empty character vector of length equal to DatesTemp
```

```
DineDate <- character(length(DatesTemp))

# Extract the actual date information for these instances by comparing to
# the present date
for (i in 1:length(DatesTemp)) {
    # Extract the number of days ago that the review was posted.
    dineDay <- regmatches(DatesTemp, regexpr("[0-9]+", DatesTemp)) %>% as.numeric()
    # Grab today's date
    todayDate <- Sys.Date()
    # Subtract the number of days from today's date
    DineDate[i] <- todayDate - dineDay
}

# Replace the date entries with the proper dates. Note that these are not
# yet formatted as date class
OpenTableData$Dates[DatesLogic] <- DineDate
```

Finally, having finish the data gathering stage, we can save our raw data to file.

```
# Save raw data to file
save(YelpData, OpenTableData, TripAdData, ZomatoData, file = "./Data/CapstoneRawData.RData")
```

3 Data Cleaning

With our raw data in hand, the next step of the process is to clean the data. There are a number of things that we would like to do:

- Remove unnecessary "previous review" data from Yelp data
- Remove the unnecessary `NA` values from Zomato ratings data
- Remove the duplicate truncated reviews from Zomato
- Consolidate `Dates1` and `Dates2` variables for Trip Advisor
- Create vectors that describe which website the data belongs to
- Merge all customer review data into a data frame
- Clean up all dates and convert to `date` class
- Clean up all ratings and convert to `numeric` class
- Clean up reviews as needed

Let's get started.

Since the code written above is not executable, I will load the raw data from the "CapstoneRawData.RData" file.

```
load("./Data/CapstoneRawData.RData")
str(YelpData)
```

```
## List of 4
##  $ Reviews: chr [1:136] "The first time I started dining here was due to summerlicious. My friends
##  $ Ratings: chr [1:138] "4.0 star rating" "3.0 star rating" "4.0 star rating" "4.0 star rating" .
```

```
## $ Dates  : chr [1:138] "\n      3/19/2017\n    " "\n      1/31/2017\n    " "\n      1/24/2017\n    "
## $ PrevRev: chr [1:138] "<span class=\"rating-qualifier\">\n      3/19/2017\n    </span>" "<span
```

```
str(OpenTableData)
```

```
## List of 3
## $ Reviews: chr [1:309] "Bibimbap needs more veg and more sauce. Wouldn't order it again." "This w
##  $ Ratings: chr [1:309] "3" "5" "5" "3" ...
## $ Dates  : chr [1:309] "17242" "Dined on March 9, 2017" "Dined on March 9, 2017" "Dined on Februar
```

```
str(TripAdData)
```

```
## List of 4
## $ Reviews: chr [1:222] "\nGreat place to meet after work for drinks. Very beautiful setting, déc
## $ Ratings: chr [1:222] "4 of 5 bubbles" "2 of 5 bubbles" "4 of 5 bubbles" "4 of 5 bubbles" ...
## $ Dates1 : chr [1:6] "22 March 2017" "21 March 2017" "5 March 2017" "1 March 2017" ...
## $ Dates2 : chr [1:222] "Reviewed yesterday\nNEW " "Reviewed 2 days ago\nNEW " "Reviewed 2 weeks a
```

```
str(ZomatoData)
```

```
## List of 3
## $ Reviews: chr [1:11] "\n          Rated \n                        Had lunch at Reds on an extrem
##  $ Ratings: chr [1:22] "Rated 3.0" NA "Rated 5.0" NA ...
## $ Dates  : chr [1:10] "2016-08-01 16:22:56" "2016-04-20 05:25:03" "2016-01-15 23:20:46" "2015-1
```

Let's start with the Yelp data. We will remove "previous reviews", as mentioned above. We will also generate a vector that indicates that this is data from Yelp. Finally, we will merge these variables into a data frame specific to Yelp.

```
# Identify which reviews are not previous reviews and discard those that do
NoPrevRev <- grepl("has-previous-review", YelpData$PrevRev) == FALSE
YelpData$Ratings <- YelpData$Ratings[NoPrevRev]
YelpData$Dates <- YelpData$Dates[NoPrevRev]

# Create vector to describe the review category as Yelp
YelpVec <- rep("Yelp", length(YelpData$Reviews))

# Combine Yelp data vectors in DF
YelpDF <- data_frame(Reviews = YelpData$Reviews, Ratings = YelpData$Ratings,
    Dates = YelpData$Dates, Website = YelpVec)
```

That's it for the Yelp data. Let's move on to OpenTable. Since we have already cleaned the dates prior to saving the raw data to file, we only need to create the categorical vector and merge the variables to a data frame.

```
# Create vector to describe category of OpenTable
OpenTableVec <- rep("OpenTable", length(OpenTableData$Reviews))

# Create OpenTable data frame
OpenTableDF <- data_frame(Reviews = OpenTableData$Reviews, Ratings = OpenTableData$Ratings,
    Dates = OpenTableData$Dates, Website = OpenTableVec)
```

To clean the Zomato data, we will remove the unnecessary `NA` values as well as the duplicate truncated reviews.

```
# Remove the double values from the ratings, which take on NAs
ZomatoData$Ratings <- ZomatoData$Ratings[!is.na(ZomatoData$Ratings)]

# Remove duplicate reviews. These truncated duplicates have the regex 'read
# more' within them.
FullRev <- !grepl("read more", ZomatoData$Reviews)
ZomatoData$Ratings <- ZomatoData$Ratings[FullRev]
ZomatoData$Reviews <- ZomatoData$Reviews[FullRev]

# Create vector describe website category
ZomatoVec <- rep("Zomato", length(ZomatoData$Reviews))

# Merge to data frame
ZomatoDF <- data_frame(Reviews = ZomatoData$Reviews, Ratings = ZomatoData$Ratings,
    Dates = ZomatoData$Dates, Website = ZomatoVec)
```

Finally, we will clean the Trip Advisor data by consolidating the different date variables.

```
# Replace dates of the form 'Reviewed ## days ago' with the proper dates
TripAdData$Dates2[grepl("ago|yesterday|today", TripAdData$Dates2)] <- TripAdData$Dates1

# Create vector describing website
TripAdVec <- rep("TripAdvisor", length(TripAdData$Reviews))

TripAdDF <- data_frame(Reviews = TripAdData$Reviews, Ratings = TripAdData$Ratings,
    Dates = TripAdData$Dates2, Website = TripAdVec)
```

Now that the data from the individual websites is stored in a data frame, we can join these data frames together to have all of the data in one place.

```
# Merge all data frames
d1 <- suppressMessages(full_join(YelpDF, OpenTableDF))
d2 <- suppressMessages(full_join(d1, ZomatoDF))
CapstoneDF <- suppressMessages(full_join(d2, TripAdDF)) %>% group_by(Website)
str(CapstoneDF)
```

```
## Classes 'grouped_df', 'tbl_df', 'tbl' and 'data.frame':  677 obs. of  4 variables:
##  $ Reviews: chr  "The first time I started dining here was due to summerlicious. My friends and I a
```

```
## $ Ratings: chr  "4.0 star rating" "3.0 star rating" "4.0 star rating" "4.0 star rating" ...
## $ Dates  : chr  "\n      3/19/2017\n    " "\n      1/31/2017\n    " "\n      1/24/2017\n    " "\n
##  $ Website: chr  "Yelp" "Yelp" "Yelp" "Yelp" ...
##  - attr(*, "vars")=List of 1
##   ..$ : symbol Website
##  - attr(*, "drop")= logi TRUE
##  - attr(*, "indices")=List of 4
##   ..$ : int  136 137 138 139 140 141 142 143 144 145 ...
##   ..$ : int  455 456 457 458 459 460 461 462 463 464 ...
##   ..$ : int  0 1 2 3 4 5 6 7 8 9 ...
##   ..$ : int  445 446 447 448 449 450 451 452 453 454
##  - attr(*, "group_sizes")= int  309 222 136 10
##  - attr(*, "biggest_group_size")= int 309
##  - attr(*, "labels")='data.frame':   4 obs. of  1 variable:
##   ..$ Website: chr  "OpenTable" "TripAdvisor" "Yelp" "Zomato"
##   ..- attr(*, "vars")=List of 1
##   .. ..$ : symbol Website
##   ..- attr(*, "drop")= logi TRUE
```

```
summary(CapstoneDF)
```

```
##    Reviews             Ratings             Dates
##  Length:677         Length:677         Length:677
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##    Website
##  Length:677
##  Class :character
##  Mode  :character
```

This is a great start, but so far all of our data is of `character` class. Our next goal is to clean up the date and ratings data in order to express them in more quantitative terms.

3.1 Cleaning Up the Dates

Let's take a look at what the dates from Yelp look like.

```
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "Yelp"), n = 10)
```

```
##  [1] "\n        3/19/2017\n    "  "\n        1/31/2017\n    "
##  [3] "\n        1/24/2017\n    "  "\n        1/10/2017\n    "
##  [5] "\n        12/27/2016\n    " "\n        7/12/2016\n    "
##  [7] "\n        7/29/2016\n    "  "\n        3/10/2017\n    "
##  [9] "\n        11/10/2016\n    " "\n        6/17/2016\n    "
```

The first thing we need to do is get rid of the newline and space characters.

```r
# Remove newline characters and spaces
CapstoneDF$Dates <- gsub("\n *", "", CapstoneDF$Dates)
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "Yelp"), n = 10)
```

```
##  [1] "3/19/2017"  "1/31/2017"  "1/24/2017"  "1/10/2017"  "12/27/2016"
##  [6] "7/12/2016"  "7/29/2016"  "3/10/2017"  "11/10/2016" "6/17/2016"
```

That looks better. Next we look for data that doesn't fit this pattern.

```r
# Find data that doesn't fit Yelp pattern
UncleanDates <- CapstoneDF$Dates[!grepl("^[0-9].*[0-9]$", CapstoneDF$Dates)]
head(UncleanDates, n = 10)
```

```
##  [1] "3/5/2016Updated review"     "2/14/2014Updated review"
##  [3] "Dined on March 9, 2017"     "Dined on March 9, 2017"
##  [5] "Dined on February 27, 2017" "Dined on February 23, 2017"
##  [7] "Dined on February 19, 2017" "Dined on February 19, 2017"
##  [9] "Dined on February 19, 2017" "Dined on February 15, 2017"
```

Following this output, we see that we need to remove the "Updated review" phrase, as well as "Dined on".

```r
# Remove 'Updated review'
CapstoneDF$Dates <- gsub("Updated review.*$", "", CapstoneDF$Dates)

# Remove 'Dined on '
CapstoneDF$Dates <- gsub("Dined on ", "", CapstoneDF$Dates)
```

Now let's take a look at the date data from Open Table, Trip Advisor, and Zomato.

```r
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "OpenTable"), n = 20)
```

```
##  [1] "17242"             "March 9, 2017"     "March 9, 2017"
##  [4] "February 27, 2017" "February 23, 2017" "February 19, 2017"
##  [7] "February 19, 2017" "February 19, 2017" "February 15, 2017"
## [10] "February 12, 2017" "February 9, 2017"  "February 6, 2017"
## [13] "February 4, 2017"  "January 31, 2017"  "January 30, 2017"
## [16] "January 28, 2017"  "January 28, 2017"  "January 26, 2017"
## [19] "January 21, 2017"  "January 4, 2017"
```

```r
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "TripAdvisor"), n = 20)
```

```
##  [1] "22 March 2017"             "21 March 2017"
##  [3] "5 March 2017"              "1 March 2017"
##  [5] "20 February 2017"          "17 February 2017"
##  [7] "Reviewed 9 February 2017"  "Reviewed 6 February 2017"
##  [9] "Reviewed 2 February 2017"  "Reviewed 2 February 2017"
```

```
## [11] "Reviewed 31 January 2017"  "Reviewed 29 January 2017"
## [13] "Reviewed 28 January 2017"  "Reviewed 27 January 2017"
## [15] "Reviewed 24 January 2017"  "Reviewed 19 January 2017"
## [17] "Reviewed 13 January 2017"  "Reviewed 5 January 2017"
## [19] "Reviewed 1 January 2017"   "Reviewed 29 December 2016"
```

```
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "Zomato"), n = 20)
```

```
##  [1] "2016-08-01 16:22:56" "2016-04-20 05:25:03" "2016-01-15 23:20:46"
##  [4] "2015-12-23 00:15:16" "2015-11-19 01:28:48" "2015-10-12 04:06:25"
##  [7] "2015-07-10 08:57:14" "2015-06-29 05:47:53" "2015-06-21 06:23:44"
## [10] "2015-03-25 01:09:18"
```

The data from Open Table and Zomato looks good. All we have to do is remove "Reviewed" from the Trip Advisor data

```
# Remove 'Reviewed '
CapstoneDF$Dates <- gsub("Reviewed ", "", CapstoneDF$Dates)
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "TripAdvisor"), n = 20)
```

```
##  [1] "22 March 2017"    "21 March 2017"    "5 March 2017"
##  [4] "1 March 2017"     "20 February 2017" "17 February 2017"
##  [7] "9 February 2017"  "6 February 2017"  "2 February 2017"
## [10] "2 February 2017"  "31 January 2017"  "29 January 2017"
## [13] "28 January 2017"  "27 January 2017"  "24 January 2017"
## [16] "19 January 2017"  "13 January 2017"  "5 January 2017"
## [19] "1 January 2017"   "29 December 2016"
```

The dates data should now be stripped of unnecessary characters. The next step in our cleaning process is to express all of this data in terms of a `Date` class. This will be done in parts since the date is formatted differently for the different websites.

Start with Yelp.

```
# Yelp date format is as follows
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "Yelp"))
```

```
## [1] "3/19/2017"  "1/31/2017"  "1/24/2017"  "1/10/2017"  "12/27/2016"
## [6] "7/12/2016"
```

```
# grep for the Yelp dates
YelpDateRegex <- grep("^[0-9]+/.*[0-9]$", CapstoneDF$Dates)
# Given the dates, we will express them as date variables
CapstoneDF$Dates[YelpDateRegex] <- CapstoneDF$Dates[YelpDateRegex] %>% as.Date(format = "%m/%d/
```

Open Table:

```
# Open Table date format:
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "OpenTable"))
```

```
## [1] "17242"             "March 9, 2017"     "March 9, 2017"
## [4] "February 27, 2017" "February 23, 2017" "February 19, 2017"
```

```
# grep for Open Table dates and express as date variable
OpenTableDateRegex <- grep("^([Jj]|[Ff]|[Mm]|[Aa]|[Jj]|[Ss]|[Oo]|[Nn]|[Dd]).+[0-9]+$",
    CapstoneDF$Dates)
CapstoneDF$Dates[OpenTableDateRegex] <- CapstoneDF$Dates[OpenTableDateRegex] %>%
    as.Date(format = "%B %d, %Y")
```

Trip Advisor:

```
# Trip Advisor date format:
head(subset(CapstoneDF$Dates, CapstoneDF$Website == "TripAdvisor"))
```

```
## [1] "22 March 2017"    "21 March 2017"    "5 March 2017"
## [4] "1 March 2017"     "20 February 2017" "17 February 2017"
```

```
# grep for Trip Advisor dates and express as date variable
TripAdRegex <- grep("^[0-9]+ ([Jj]|[Ff]|[Mm]|[Aa]|[Jj]|[Ss]|[Oo]|[Nn]|[Dd]).+[0-9]+$",
    CapstoneDF$Dates)
CapstoneDF$Dates[TripAdRegex] <- CapstoneDF$Dates[TripAdRegex] %>% as.Date(format = "%d %B %Y")
```

Finally, since Zomato dates are already in POSIXct format, we can convert them trivially to dates

```
CapstoneDF$Dates[which(CapstoneDF$Website == "Zomato")] <- CapstoneDF$Dates[which(CapstoneDF$We
    "Zomato")] %>% as.Date()
```

Let's finish by imposing the `Date` class on the `Dates` variable of our data frame, just to be sure.

```
class(CapstoneDF$Dates) <- "Date"

str(CapstoneDF$Dates)
```

```
##  Date[1:677], format: "2017-03-19" "2017-01-31" "2017-01-24" "2017-01-10" ...
```

3.2 Numeric Ratings

Next, do the same with numerical ratings.
Yelp

```
# What do the Yelp ratings look like?
head(subset(CapstoneDF$Ratings, CapstoneDF$Website == "Yelp"))
```

```
## [1] "4.0 star rating" "3.0 star rating" "4.0 star rating" "4.0 star rating"
## [5] "3.0 star rating" "3.0 star rating"
```

```
# Get rid of 'star rating'
CapstoneDF$Ratings <- gsub("star rating", "", CapstoneDF$Ratings)
```

Open Table

```
# Open Table ratings
head(subset(CapstoneDF$Ratings, CapstoneDF$Website == "OpenTable"))
```

```
## [1] "3" "5" "5" "3" "5" "3"
```

Looks good already.
Trip Advisor

```
# TripAdvisor ratings
head(subset(CapstoneDF$Ratings, CapstoneDF$Website == "TripAdvisor"))
```

```
## [1] "4 of 5 bubbles" "2 of 5 bubbles" "4 of 5 bubbles" "4 of 5 bubbles"
## [5] "5 of 5 bubbles" "5 of 5 bubbles"
```

```
# Get rid of 'of 5 bubbles'
CapstoneDF$Ratings <- gsub("of [0-9] bubbles", "", CapstoneDF$Ratings)
```

Zomato

```
# Zomato ratings
head(subset(CapstoneDF$Ratings, CapstoneDF$Website == "Zomato"))
```

```
## [1] "Rated 3.0" "Rated 5.0" "Rated 4.5" "Rated 1.5" "Rated 4.0" "Rated 1.0"
```

```
# Get rid of 'Rated '
CapstoneDF$Ratings <- gsub("Rated ", "", CapstoneDF$Ratings)
```

Impose numeric class and print to check.

```
# Impose numeric class
class(CapstoneDF$Ratings) <- "numeric"

str(CapstoneDF$Ratings)
```

```
##  num [1:677] 4 3 4 4 3 3 4 1 1 3 ...
```

3.3 Final Touches

```
#
CapstoneDF$Website <- factor(CapstoneDF$Website, order = FALSE, levels = c("Yelp",
    "OpenTable", "Zomato", "TripAdvisor"))

str(CapstoneDF$Website)
```

```
##  Factor w/ 4 levels "Yelp","OpenTable",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
levels(CapstoneDF$Website)
```

```
## [1] "Yelp"        "OpenTable"   "Zomato"      "TripAdvisor"
```

```
# Let's remove newline characters from reviews
CapstoneDF$Reviews <- gsub("\n", "", CapstoneDF$Reviews)

# Clean up the Zomato reviews a bit by removing the 'Rated' at the beginning
head(subset(CapstoneDF$Reviews, CapstoneDF$Website == "Zomato"), n = 2)
```

```
## [1] "           Rated                       Had lunch at Reds on an extremely hot day.  Too hot t
## [2] "           Rated                       My wife & I were looking for place to try for the fir
```

```
CapstoneDF$Reviews <- gsub(" +Rated *", "", CapstoneDF$Reviews)

head(subset(CapstoneDF$Reviews, CapstoneDF$Website == "Zomato"), n = 2)
```

```
## [1] "                       Had lunch at Reds on an extremely hot day.  Too hot to sit on the pati
## [2] "                       My wife & I were looking for place to try for the first time nice plac
```

```
write_csv(CapstoneDF, "./Data/CapstoneCleanData.csv")
```

4 Data Analysis

4.1 Time Series Analysis of Ratings

```
# Load libraries
library(readr)
library(dplyr)
library(tidyr)
suppressMessages(library(tm))
suppressMessages(library(wordcloud))
suppressMessages(library(ggplot2))
suppressMessages(library(syuzhet))

# Read in clean data
```

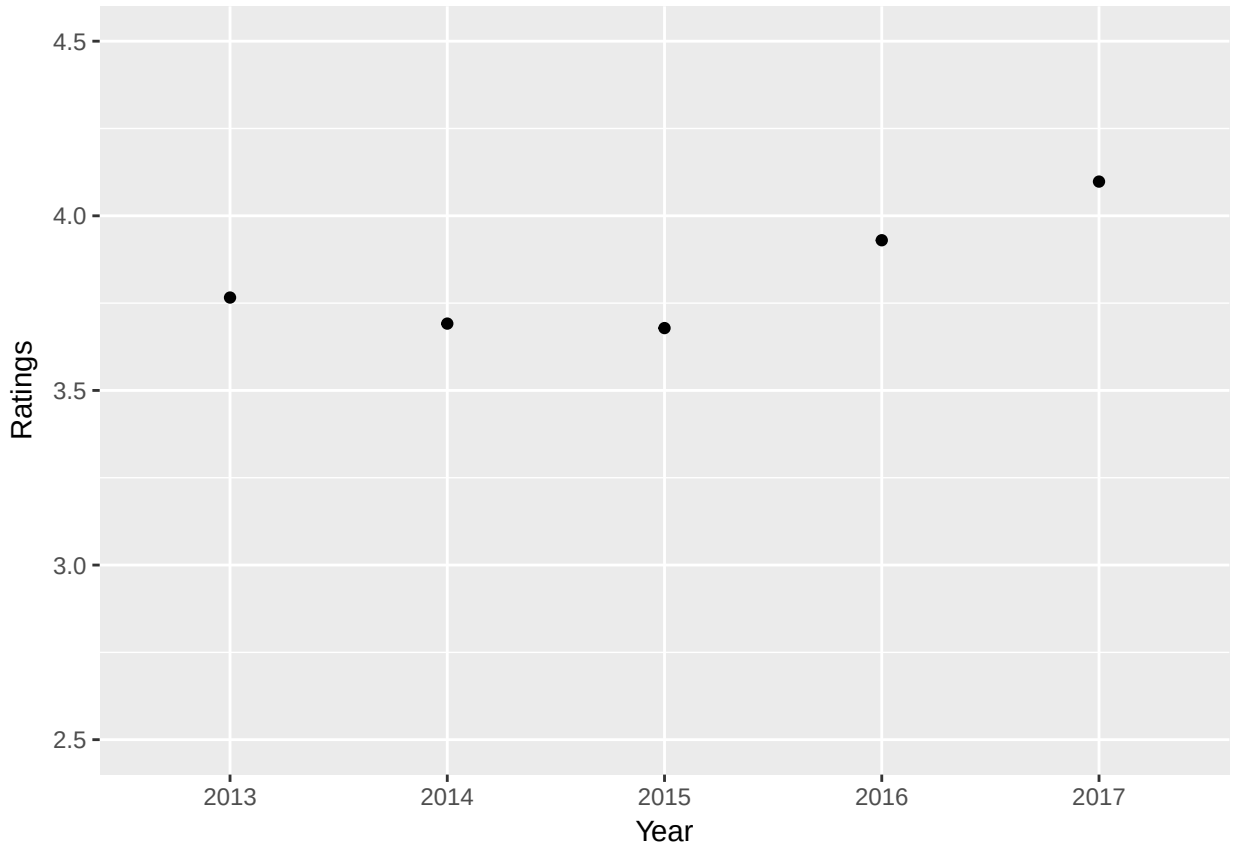
```r
CapstoneDF <- suppressMessages(read_csv("./Data/CapstoneCleanData.csv"))

# Summary of data frame
glimpse(CapstoneDF)
```

```
## Observations: 677
## Variables: 4
## $ Reviews <chr> "The first time I started dining here was due to summe...
## $ Ratings <dbl> 4, 3, 4, 4, 3, 3, 4, 1, 1, 3, 3, 4, 5, 4, 3, 4, 5, 5, ...
## $ Dates   <date> 2017-03-19, 2017-01-31, 2017-01-24, 2017-01-10, 2016-...
## $ Website <chr> "Yelp", "Yelp", "Yelp", "Yelp", "Yelp", "Yelp", "Yelp"...
```

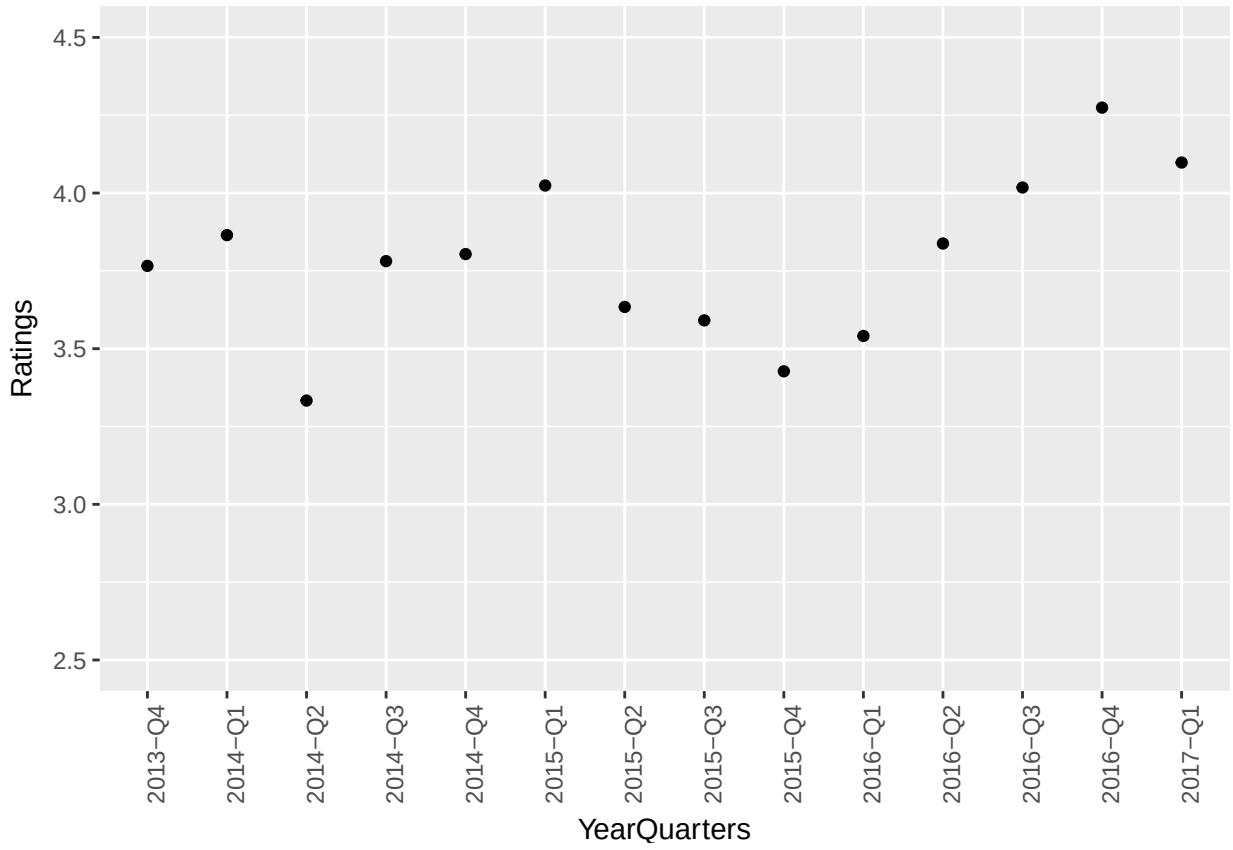
```r
# Create Quarters variable
CapstoneDF <- CapstoneDF %>% mutate(Quarters = quarters.Date(Dates))

# Separate date variable into Year, Month, Day
CapstoneDF <- CapstoneDF %>% separate(Dates, c("Year", "Month", "Day"))

# Create variable that describes month and year.
tempdf <- CapstoneDF %>% unite("YearMonth", Year, Month, sep = "-")
CapstoneDF$YearMonth <- tempdf$YearMonth

# Create variable that describes quarters and years
tempdf <- CapstoneDF %>% unite("YearQuarters", Year, Quarters, sep = "-")
CapstoneDF$YearQuarters <- tempdf$YearQuarters

# Convert Website and Quarters to factor class
CapstoneDF$Website <- factor(CapstoneDF$Website)
CapstoneDF$Quarters <- factor(CapstoneDF$Quarters)

# Take a look at the data
head(CapstoneDF[-1])
```

```
## # A tibble: 6 × 8
##   Ratings  Year Month   Day Website Quarters YearMonth YearQuarters
##     <dbl> <chr> <chr> <chr>  <fctr>   <fctr>     <chr>        <chr>
## 1       4  2017    03    19    Yelp       Q1   2017-03      2017-Q1
## 2       3  2017    01    31    Yelp       Q1   2017-01      2017-Q1
## 3       4  2017    01    24    Yelp       Q1   2017-01      2017-Q1
## 4       4  2017    01    10    Yelp       Q1   2017-01      2017-Q1
## 5       3  2016    12    27    Yelp       Q4   2016-12      2016-Q4
## 6       3  2016    07    12    Yelp       Q3   2016-07      2016-Q3
```

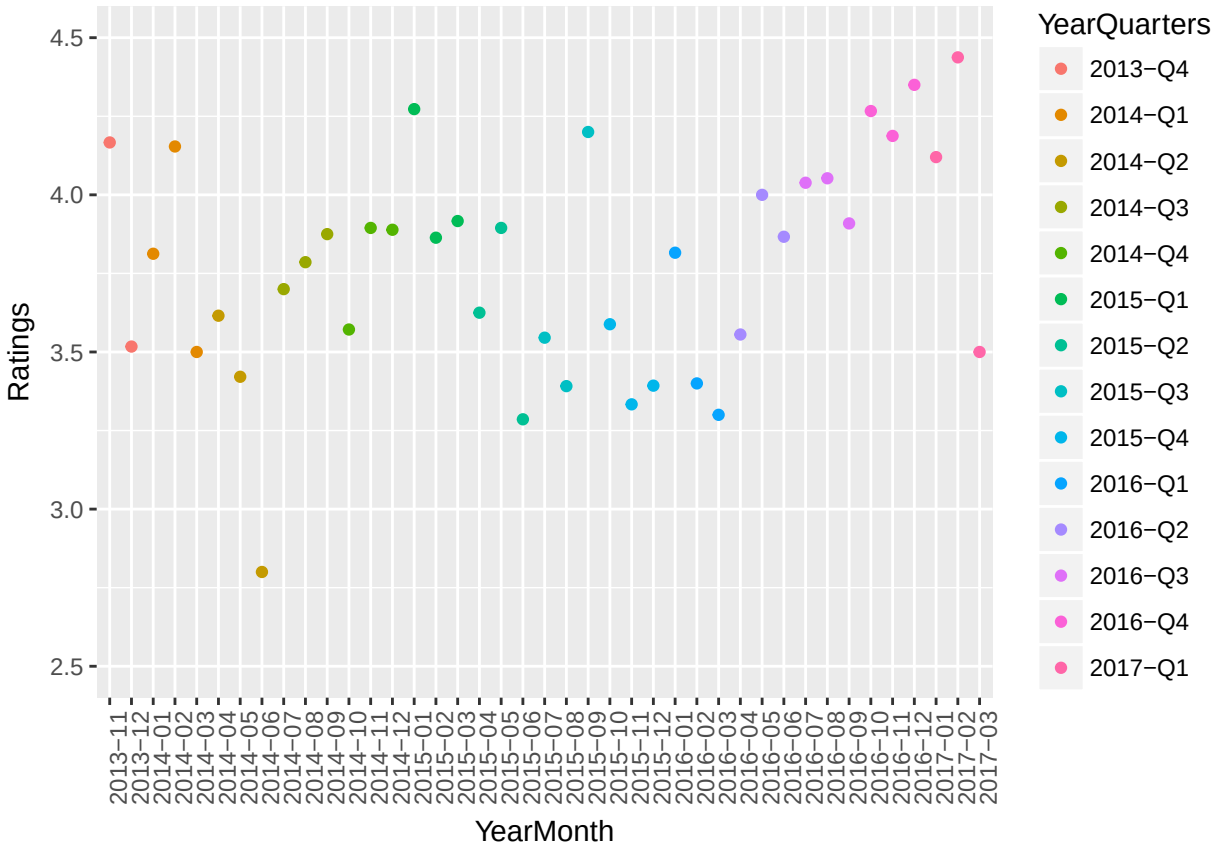
```r
# Ratings aggregated by Year
global_p1 <- ggplot(CapstoneDF, aes(x = Year, y = Ratings)) + stat_summary(fun.y = mean,
    geom = "point") + coord_cartesian(ylim = c(2.5, 4.5))
global_p1
```




```
# Ratings aggregated by Year and Quarters
global_p2 <- ggplot(CapstoneDF, aes(x = YearQuarters, y = Ratings)) + stat_summary(fun.y = mean
    geom = "point") + coord_cartesian(ylim = c(2.5, 4.5)) + theme(axis.text.x = element_text(an
global_p2
```




```
global_p3 <- ggplot(CapstoneDF, aes(x = YearMonth, y = Ratings, col = YearQuarters)) +
    stat_summary(fun.y = mean, geom = "point") + coord_cartesian(ylim = c(2.5,
    4.5)) + theme(axis.text.x = element_text(angle = 90))
global_p3
```



4.2 Customer Reviews Text Analysis

5 References

"REDS Midtown Tavern." n.d. www.redsmidtowntavern.com.