REDS Midtown Tavern Online Review Analytics

Foundations of Data Science: Capstone Project Report

Antoine Beauchamp
March 28th, 2017

Contents

1 Introduction

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.

Citation example:("REDS Midtown Tavern," n.d.) **Would rather have it be only numbered, and hyperlinked to the bibliography**

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 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: Yelp, OpenTable, TripAdvisor, and Zomato. 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 and the web scraping tool, SelectorGadget. **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 selector 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, as displayed in Figure 1.

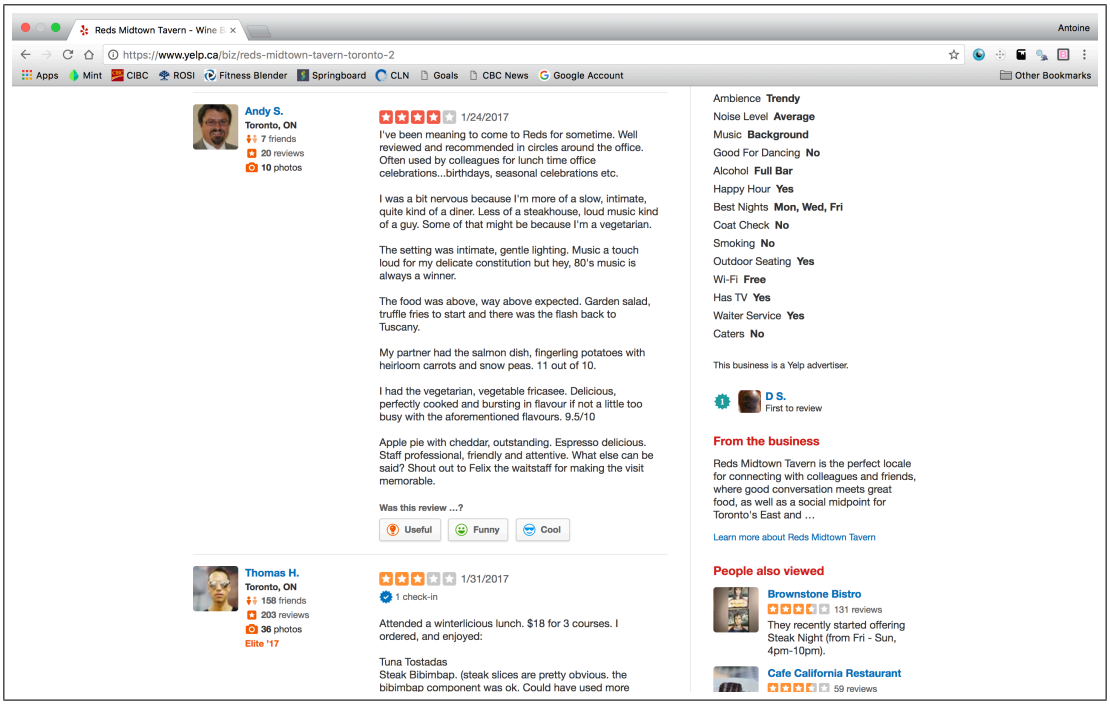

Figure 1: Customer reviews for REDS Midtown Tavern on Yelp

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. The selector "`p`" is returning too broad a selection. 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, so that only the customer reviews remain 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 (identified in parentheses on the SelectorGadget tool), which corresponds to the number of reviews on this particular 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 Section ????

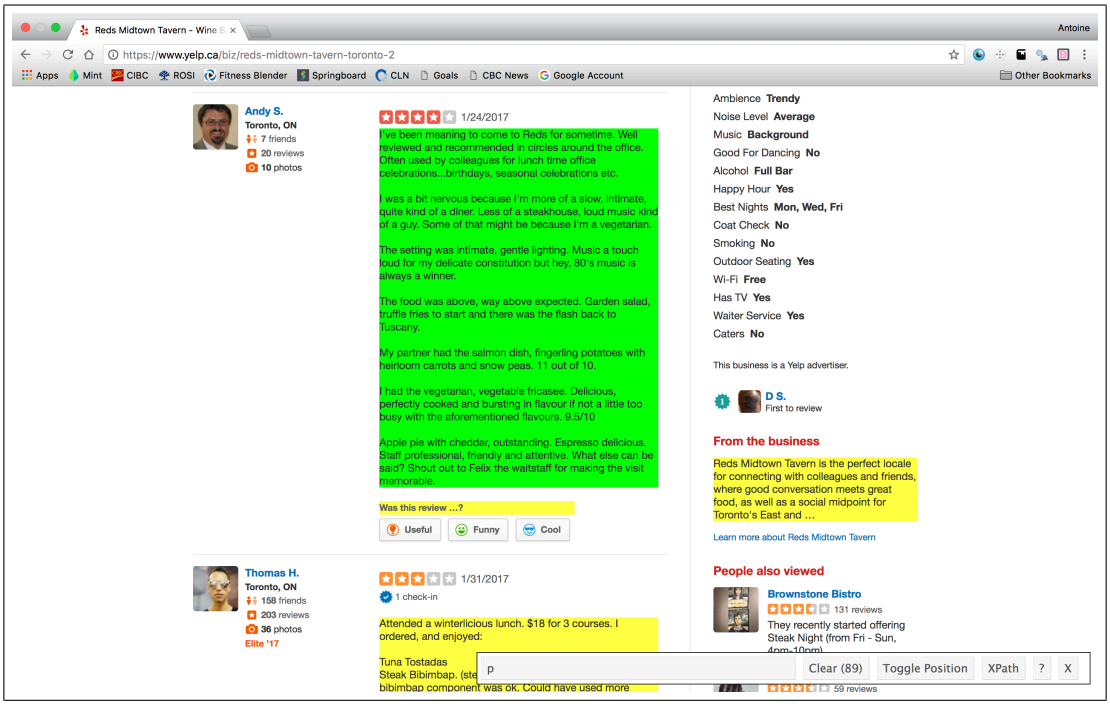

Figure 2: First attempt at selecting Yelp customer reviews with SelectorGadget

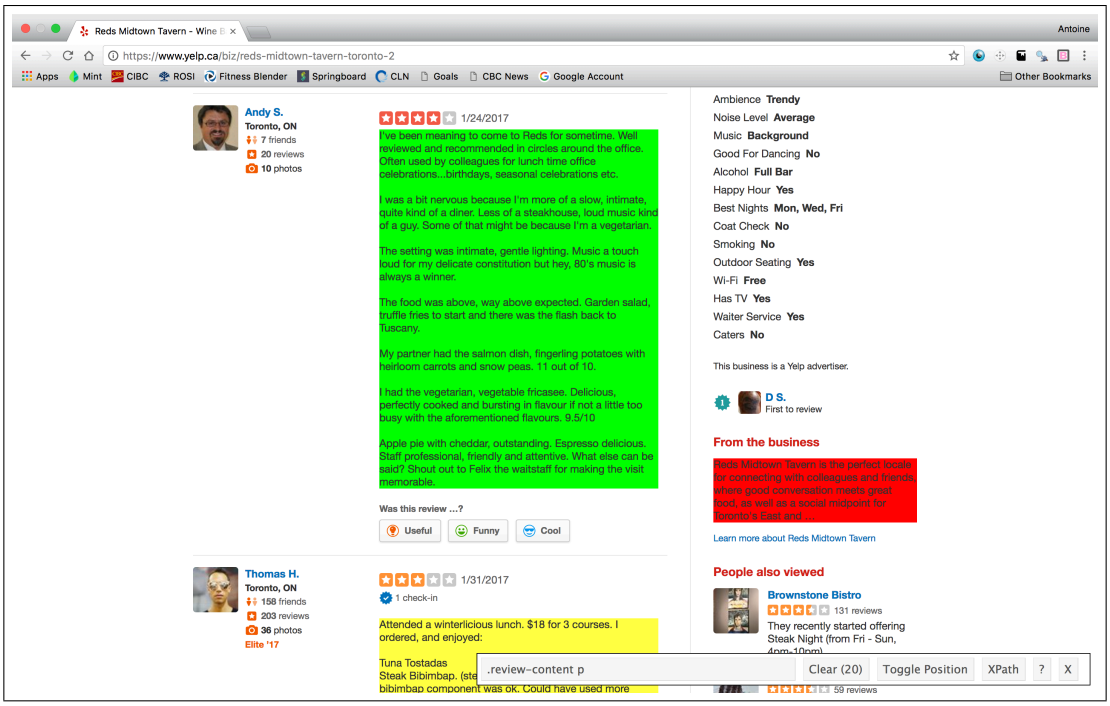

Figure 3: Correct use of SelectorGadget on Yelp

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 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 block of a certain type, simply use the indicator for that type. For example, to obtain content that is contained within a paragraph block, use the selector "`p`". In fact, this is what happened when we used SelectorGadget in Figure 2: the customer review content was contained within 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 basic type selection, 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. **Make sure I'm consistent when using "For example" or** if there exists a block defined by `<div`

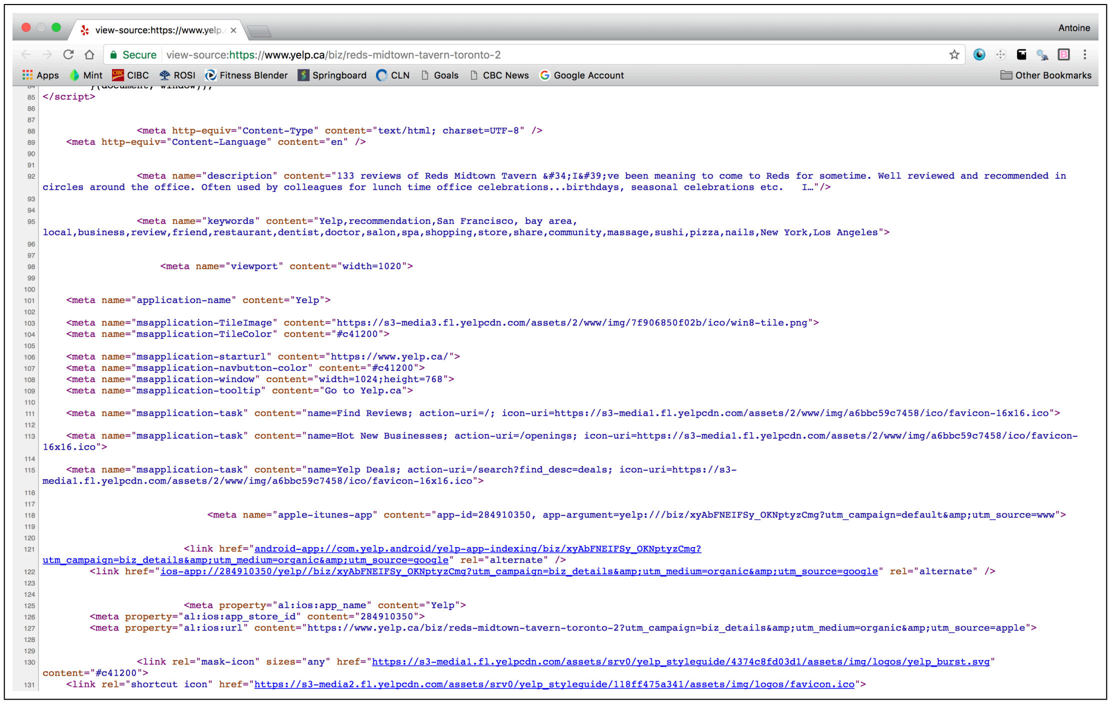

Figure 4: HTML source for REDS Midtown Tavern on Yelp

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 a block such as <div class="content"> using .content.

Now that we have these basic selectors, we can combine them to tailor our selection process and access more specific content. One way of doing this is to use the **descendant selector**. This allows us to select content that is nested within another content block. The descendant selector is expressed by writing two selectors separated by a space. E.g. We could select something like <div id="title"><p> Text </p></div> with the selector div p. This selects all paragraph content within a <div> block. We can also use the descendant selector in conjunction with the ID or class selectors. E.g. #title p will select all paragraph content contained within a block of any type with 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 is comprised of a descendant selector with the class and type selectors. We are selecting the paragraph blocks, <p>, within blocks of content 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"> within 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/ **Cite properly?**.

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 extract, the next step is to load this data into R for cleaning and processing. I performed this step of the data gathering process using Hadley Wickham's web data R package, `rvest`. `rvest` allows us to gather data from the web using a few basic functions in R: `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()`.

```
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)
```

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">Unbelievably bad service. I think everyone course had a ...
## [5] <p lang="en">As a young barrister  I often visited Red's older, fanc ...
## [6] <p lang="en">I came here for dinner and had wine and entrees. I wish ...
```

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=3)
```

```
## [1] "<p lang=\"en\">The first time I started dining here was due to summerlicious. My friends
## [2] "<p lang=\"en\">Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed
## [3] "<p lang=\"en\">I've been meaning to come to Reds for sometime. Well reviewed and recomme
```

```
class(YelpReviews_char1)
```

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

head() output here is exceeding page width

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
## [2] "Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:Tuna TostadasS
```

head() output here is exceeding page width

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, evidently. 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 its 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 extracting the numerical ratings that the customers have given to the restaurant. 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, obtained via SelectorGadget, is `.rating-large`.

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

```
## [1] "<div class=\"i-stars i-stars--regular-4 rating-large\" title=\"4.0 star r"
## [2] "ating\">\n      <img class=\"offscreen\" height=\"303\" src=\"https://s3-me"
## [3] "dia1.fl.yelpcdn.com/assets/srv0/yelp_design_web/41341496d9db/assets/im"
## [4] "g/stars/stars.png\" width=\"84\" alt=\"4.0 star rating\">\n</div>"
```

Here we see that the value that we want, namely "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 specifically, 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" "2.0 star rating"
## [5] "4.0 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. Now that we have a foundation in how to gather web data using SelectorGadget and `rvest`, let's acquire the full data set **Be consistent for data set vs dataset** of online customer reviews for REDS Midtown Tavern.

2.3 Gathering Online Customer Review Data

With a basic understanding of CSS selectors, SelectorGadget, and `rvest`, we are equipped to gather our customer review data for REDS Midtown Tavern. As mentioned previously, the following websites were used as data sources: Yelp, OpenTable, TripAdvisor, and Zomato. Specifically, our data set will be comprised of the written customer reviews, the numerical customer ratings, and the review dates. Note that the code presented in this section will not be executable due to the longer computation times. Let's start by loading the necessary R packages.

```
#Load required libraries
library(rvest)
library(dplyr)
library(tidyr)
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 using CSS Selectors. 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

Maybe mod this algorithm

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.

2.3.1 Yelp Data

```
## Function: YelpScrape ##

# This function is used to scrape review data from Yelp.com
# Arguments:
# BaseURL: URL to the first page of reviews
YelpScrape <- function(BaseURL) {

  #Review counter. Yelp = 20 per page
  ReviewCount <- 0
  #Vector intialization
  Reviews <- character(0)
  Ratings <- character(0)
  Dates <- character(0)
  PrevRev <- character(0)
  flag <- 1

  #Iterate over review pages and scrape data
  while(flag==1){

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

    #Scrape reviews, ratings, dates from 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()
    #Additional data identifying previous/updated reviews
    PrevRevNew <- read_html(page_url) %>%
      html_nodes(".biz-rating-large .rating-qualifier") %>%
      as.character()

    #Print iteration count identifier to std.out
    print(paste("Scraping Yelp page",ceiling(ReviewCount/20)))

    #Append new data to existing vectors
    Reviews <- c(Reviews,ReviewsNew)
    Ratings <- c(Ratings,RatingsNew)
```

```
    Dates <- c(Dates, DatesNew)
    PrevRev <- c(PrevRev,PrevRevNew)

    #Increment counter
    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 primary 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 text review 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 function is used to scrape review data from OpenTable
# Arguments:
# BaseURL: URL to the review page from OpenTable.
# Note that the URL must end with &page=
# Page number is specified in the function.
OpenTableScrape <- function(BaseURL) {

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

  #Iterate over review pages and scrape data
  while(flag==1) {

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

```
    #Scrape reviews, ratings, dates from current URL
    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 iteration count identifier to std.out
    print(paste("Scraping OpenTable page",ReviewCount))

    #Increment counter
    ReviewCount <- ReviewCount+1

    #Loop ending condition
    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 Section 3.

2.3.3 TripAdvisor Data

When scraping reviews from TripAdvisor, a difficulty arises in that the URL for each of the review pages does not appear to have an obvious pattern. Fortunately, the URL for the subsequent review page is contained within the HTML source of the current review page, as part of the link to the next page. Therefore to get the URL for the next page, I had to identify the selector for the link to the following page, and extract the data.

Another feature that I've implemented is that the web scraping process for TripAdvisor actually begins at the landing page for REDS Midtown Tavern, rather than at the first full review page. The scraping 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 recognized that new reviews were not being included in this "first" page. The review 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 bypasses this problem, since new reviews are included on the landing page.

```
## Function: TripAdScrape ##

# This function is used to scrape review data from TripAdvisor
# Arguments:
# LandingURL: URL for the landing page of the restaurant to scrape
# It will be used to link to the full review pages
TripAdScrape <- function(LandingURL) {


  #Get link to full review pages. These are embedded in review titles
  ReviewTitleLink <- read_html(LandingURL) %>%
    html_nodes(".quote a") %>%
    html_attr("href")

  #Base URL of first review page
  BaseURL <- paste("https://www.tripadvisor.ca",ReviewTitleLink[1],sep="")

  #Parameters/vector initialization
  ReviewCount <- 1
  Reviews <- character(0)
  Ratings <- character(0)
  Dates1 <- character(0)
  Dates2 <- character(0)
  flag <- 1

  #Iterate over review pages and scrape data
  while(flag==1){

    #Print iteration count identifier
    print(paste("Scraping TripAdvisor page",ReviewCount))

    #For the first page, the URL is just the base URL.
    #For subsequent iterations, grab hyperlink to the new page
    # from the page links.
    #E.g. page 1 carries a link to page 2 in its HTML, and so on.
    if(ReviewCount == 1){
      page_url <- BaseURL
    } else {
      #Get page numbers for the links
      pagenum <- read_html(page_url) %>%
        html_nodes(".pageNum") %>%
        html_attr("data-page-number") %>%
```

```
        as.numeric()
      #Get hyperlinks for 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="")
    }

    #Scrape reviews, ratings, dates from current URL
    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()

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

    #Append vectors
    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 extracted two sets of data for the dates, stored in `Dates1` and `Dates2`. The reason for this is that, for recent reviews, TripAdvisor 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 forms "Dined ## days ago" and so on. This format is only used to express the most recent dates. Older reviews are associated with a proper date format. Consequently, 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 primary 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.

```
## Function: ZomatoScrape ##

# This function is used to scrape review data from Zomato
# Arguments:
# BaseURL: URL to the review page from Zomato.
ZomatoScrape <- function(BaseURL) {

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

  #Scrape reviews, ratings, dates
  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 longer reviews on Zomato will be truncated and have an associated "read more" option 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 data cleaning stage.

2.3.5 Web Scraping

With the web scraping functions defined, we can finish the data acquisition process. All that is left to do is to identify the URLs for the different 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 <-
  paste("https://www.opentable.com/reds-midtown-tavern?",
        "covers=2&dateTime=2017-02-22+19%3A00%23reviews&page=")
#TripAdvisor landing page
LandingURL_TripAd <-
  paste("https://www.tripadvisor.ca/Restaurant_Review-g155019-d5058760",
        "-Reviews-Reds_Midtown_Tavern-Toronto_Ontario.html")
#Zomato main review page URL
BaseURL_Zomato <-
  paste("https://www.zomato.com/toronto/",
        "reds-midtown-tavern-church-and-wellesley/reviews")

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

Before saving the raw data to file, we need to do some basic cleaning of the OpenTable date data. The reason for this is that some of the dates are expressed in "Dined ## days ago" format. The proper date data is not available, so we have to create it by subtracting the number of days from the current date. This only works if it is done on the same day that the web data was acquired.

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

```
#Extract number of days ago that reviews were posted.
dineDay <-
  regmatches(DatesTemp,regexpr("[0-9]+",DatesTemp)) %>%
  as.numeric()

#Today's date
todayDate <- Sys.Date()

#Subtract number of days from today's date
DineDate <- todayDate - dineDay

#Replace date entries with the proper dates
OpenTableData$Dates[DatesLogic] <- DineDate
```

Finally, having completed 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

In the previous section, we developed a number of functions to scrape the customer review data for REDS Midtown Tavern from a number of popular review and travel websites. With the raw data in hand, the next step of the process is to clean the data. There are a number of things that need to be done:

- Remove unnecessary "previous review" data from Yelp data set
- Remove the unnecessary `NA` values from Zomato ratings data
- Remove the duplicate truncated reviews from Zomato
- Consolidate `Dates1` and `Dates2` variables for TripAdvisor
- Create vectors that describe which websites 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 begin.

3.1 Preliminary Cleaning

Since the code written in Section 2 is not executable, we will load the raw data from the "CapstoneRawData.RData" file.

```
load("./Data/CapstoneRawData.RData")
# Examine Yelp data
str(YelpData, width = 80, strict.width = "cut")
```

```
## List of 4
##  $ Reviews: chr [1:136] "The first time I started dining here was due to summ"..
##  $ Ratings: chr [1:138] "4.0 star rating" "3.0 star rating" "4.0 star rating""..
##  $ Dates  : chr [1:138] "\n      3/19/2017\n    " "\n      1/31/2017\n   "..
##  $ PrevRev: chr [1:138] "<span class=\"rating-qualifier\">\n      3/19/2017"..
```

```
# Examine OpenTable data
str(OpenTableData, width = 80, strict.width = "cut")
```

```
## List of 3
##  $ Reviews: chr [1:309] "Bibimbap needs more veg and more sauce. Wouldn't ord"..
##   $ Ratings: chr [1:309] "3" "5" "5" "3" ...
##  $ Dates  : chr [1:309] "17242" "Dined on March 9, 2017" "Dined on March 9, 2"..
```

```
# Examine TripAdvisor data
str(TripAdData, width = 80, strict.width = "cut")
```

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

```
# Examine Zomato data
str(ZomatoData, width = 80, strict.width = "cut")
```

```
## List of 3
##  $ Reviews: chr [1:11] "\n              Rated \n                          "..
##   $ 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" "2016-01-1"..
```

Beginning with the Yelp data, we will remove "previous reviews", as mentioned in Section 2.3.1. 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.

```
#Which reviews are not previous reviews?
NoPrevRev <- !grepl("has-previous-review",YelpData$PrevRev)
YelpData$Ratings <- YelpData$Ratings[NoPrevRev]
YelpData$Dates <- YelpData$Dates[NoPrevRev]
```

```
#Vector, Yelp identifier
YelpVec <- rep("Yelp",length(YelpData$Reviews))

#Combine to data frame
YelpDF <- data_frame(Reviews=YelpData$Reviews,
                     Ratings=YelpData$Ratings,
                     Dates=YelpData$Dates,
                     Website=YelpVec)
```

Moving on to OpenTable, we only need to create the categorical vector and merge the variables to a data frame, since the dates were cleaned in Section 2.3.5

```
#Vector, OpenTable identifier
OpenTableVec <- rep("OpenTable", length(OpenTableData$Reviews))

#Merge to 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 in the ratings data, as well as the duplicate truncated reviews.

```
#Remove NAs from ratings
ZomatoData$Ratings <- ZomatoData$Ratings[!is.na(ZomatoData$Ratings)]

#Remove duplicate reviews. These truncated duplicates can be
# identified with the regex "read more"
FullRev <- !grepl("read more",ZomatoData$Reviews)
ZomatoData$Ratings <- ZomatoData$Ratings[FullRev]
ZomatoData$Reviews <- ZomatoData$Reviews[FullRev]

#Vector, Zomato identifier
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 TripAdvisor 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

##Vector, TripAdvisor identifier
TripAdVec <- rep("TripAdvisor",length(TripAdData$Reviews))

#Merge to data frame
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 <- full_join(YelpDF,OpenTableDF)
d2 <- full_join(d1,ZomatoDF)
CapstoneDF <- full_join(d2,TripAdDF) %>% group_by(Website)
str(CapstoneDF, width=80, strict.width = "cut", give.attr = FALSE)
```

```
## 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 summerlicio"..
##  $ Ratings: chr  "4.0 star rating" "3.0 star rating" "4.0 star rating" "4.0 s"..
##  $ Dates  : chr  "\n      3/19/2017\n    " "\n      1/31/2017\n    " "\n "..
##  $ Website: chr  "Yelp" "Yelp" "Yelp" "Yelp" ...
```

```
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 the data is of `character` class. In the following sections, we will clean up the date and ratings data in order to express them in more quantitative terms.

3.2 Cleaning Up the Dates

We will begin the date cleaning stage by taking a look at the date data from Yelp.

```
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 that needs to be done is to get rid of the newline and space characters.

```
#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"
```

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

```
#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"
```

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

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

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

With that done, let's take a look at the date data from OpenTable, TripAdvisor, and Zomato.

```
#OpenTable
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"
```

```
#TripAdvisor
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"
```

```
#Zomato
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 OpenTable and Zomato is already expressed in formats that we can manipulate. All we have to do is remove "Reviewed" from the TripAdvisor data.

```
#Remove "Reviewed " from TripAdvisor data
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.

Yelp:

```
#Yelp date format
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)
#Express dates as date variables
CapstoneDF$Dates[YelpDateRegex] <-
  CapstoneDF$Dates[YelpDateRegex] %>%
  as.Date(format="%m/%d/%Y")
```

OpenTable:

```
#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 OpenTable dates and express as date
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")
```

TripAdvisor:

```
#TripAdvisor 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 TripAdvisor 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$Dates[which(CapstoneDF$Website == "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" ...
```

3.3 Numeric Ratings

In the previous section, we removed unnecessary characters from the date data and converted the data to a common `Date` class. Here we will do the same with the ratings data.

Yelp:

```
#Yelp ratings
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)
```

OpenTable:

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

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

This data is already in a workable format.

TripAdvisor:

```
#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)
```

Having removed unnecessary characters, we will now impose the `Numeric` class on the data.

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

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

This completes the cleaning process for the numerical ratings.

3.4 Final Touches

Before moving on to data analysis, we will finalize the data cleaning process by making some small adjustments to the data. The first adjustment is to convert the `Website` variable of the data frame to `Factor` class, rather than a simple `character` class **Decide if you want to capitalize classes or not**.

```
#Convert Websites to factor
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"
```

Additionally, we will remove any newline characters from the review data, and remove some unnecessary text in the Zomato review data.

```
# Remove newline characters from reviews
CapstoneDF$Reviews <- gsub("\n", "", CapstoneDF$Reviews)

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

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

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

We will finish off the section by writing our newly-cleaned data to file.

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

This completes the cleaning stage.

4 Data Analysis

Having completed the gritty process of first acquiring the customer review data from the web and subsequently cleaning it, we can now dive into the analysis stage. In the previous sections, we built a data frame with four variables: `Reviews`, `Ratings`, `Dates`, and `Website`. The variables that we will use primarily to extract insights are the `Reviews` and `Ratings` variables, containing respectively the written customer reviews and the associated numerical ratings. These variables contain two very different types of data, i.e. text data and numerical data, and so they will be manipulated in different ways. We will begin the analytic process by performing a time series analysis of the ratings data, and further move on to a text analysis of the review data. Let's take a moment to load some relevant libraries and read in the clean data.

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

```
library(ggplot2)
library(syuzhet)
#Read in clean data
CapstoneDF <- 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"...
```

We have data for 677 customer reviews from the four aformentioned websites. Before diving into the analysis, we will do some basic data wrangling to reformat the structure of the data frame.

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

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

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

#Create variable that describes Quarters and Year together: YearQuarters
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)


t1 <- CapstoneDF %>% group_by(Year) %>% summarise(countYear=n())
CapstoneDF <- left_join(CapstoneDF,t1, by="Year")

t2 <- CapstoneDF %>% group_by(YearQuarters) %>% summarise(countQuarters=n())
CapstoneDF <- left_join(CapstoneDF,t2, by="YearQuarters")

t3 <- CapstoneDF %>% group_by(YearMonth) %>% summarise(countMonth=n())
CapstoneDF <- left_join(CapstoneDF,t3, by="YearMonth")
```

```
head(CapstoneDF[-1])
```

```
## # A tibble: 6 × 11
##   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
## # ... with 3 more variables: countYear <int>, countQuarters <int>,
## #   countMonth <int>
```

We are ready to begin our analysis.

4.1 Time Series Analysis of Ratings

The first part of our data analysis process will be focussed on examining the "dynamics" of the `Ratings` variable over time, aggregated at various levels of ganularity. As a first step, let's compute the mean value of the ratings for REDS Midtown Tavern.

```
mean(CapstoneDF$Ratings)
```

```
## [1] 3.79099
```

```
CapstoneDF$RatingsAvg <-
  rep(mean(CapstoneDF$Ratings), length(CapstoneDF$Ratings))
```

The mean values of the numerical customer ratings is 3.79. We can think of this as the ratings aggregated at the lowest possible level of granularity, i.e. all of the data is aggregated using the `mean()` function. This provides some insight into the global performance of the restaurant over its entire lifespan **More insight here?**. We can mine for further insight by increasing the granularity of the data. First, let's aggregate the mean values by year.

For all of these plots, going to need to fix the themes for presentation. Get the analysis done first though.

```
#Ratings aggregated by Year
ggplot(CapstoneDF, aes(x=Year, y=Ratings)) +
  geom_line(aes(y=RatingsAvg, group=1), linetype="dashed",
            colour="red", alpha=0.5) +
  stat_summary(aes(size=countYear), fun.y=mean, geom="point") +
  #coord_cartesian(ylim=c(2.5,4.5)) +
  scale_size_continuous(range=c(3,7)) +
  coord_cartesian(ylim=c(3.0,4.5))
```

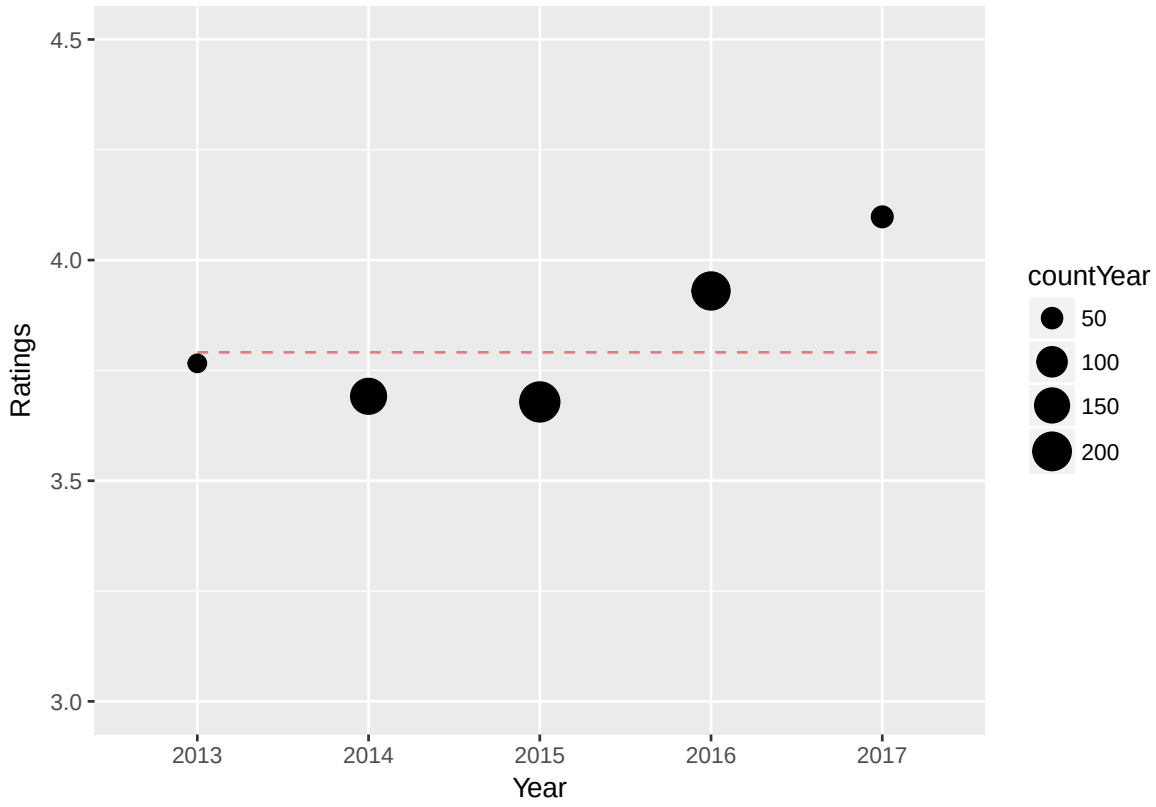

Figure 5: Customer Ratings Averaged by Year

Not sure what to do about the scale here. I'd like to make the scale consistent for Year, QUarters and Months. But it makes the data harder to read for Years

Reference to 5

By computing the mean value of the customer ratings for each year, we begin to see some of the dynamics of the ratings. In particular, we can observe that from 2013 to 2015, the restaurant experienced a decreasing trend in its ratings. 2015 was the worst year in terms of quality for the restaurant, as reported by the clientele. Interestingly, since then the ratings have been rising steadily. Now one quarter into 2017, the restaurant's ratings are at an all time high, with a mean value of 4.10. This is a great sign for the coming future. In order to better understand why the ratings were so low in 2015, and how they have improved since then, we will mine deeper into the data by looking at the mean ratings for individual quarters.

```
#Ratings aggregated by Year and Quarters
global_p2 <- ggplot(CapstoneDF, aes(x=YearQuarters, y=Ratings,col=Year)) +
  geom_line(aes(y=RatingsAvg, group=1), linetype="dashed",
            colour="black", alpha=0.5) +
  stat_summary(aes(size=countQuarters), fun.y=mean, geom="point") +
  #coord_cartesian(ylim=c(2.5,4.5)) +
  coord_cartesian(ylim=c(3.0,4.5)) +
  scale_size_continuous(range=c(3,9)) +
  theme(axis.text.x =element_text(angle=90))
global_p2
```

In this plot we are able to see some interesting details in the data. In particular, the data can be subdivided into three periods of interest, as shown in Figure **?????**. The first period is comprised of the restaurant's first year in business, i.e. from 2013-Q4 to 2014-Q4. During this period, the quarterly ratings are mainly scattered around the global mean of 3.79. The exception is an unexpectedly low value for the ratings in 2014-Q2, which can effectively be treated as an outlier to the corresponding group. In this first year, the restaurant experienced no growth in terms of quality. The first quarter of 2015 appeared to promise that things would be different in 2015, but the data from the rest of the year shows that this isn't the case. This forms the second group of interest, which spans from 2015 to the beginning of 2016. During this period, REDS Midtown Tavern experienced a steady decline in ratings, reaching bottom in the fourth quarter of 2015. **I'm not sure if this is the proper way to organize this data, by including 2016-Q1 in this "poor" group. Another way to think about it would be to include this quarter in the final group, as the beginning of the growth period. Could also just group data by year? Flat in 2014, decreasing in 2015, increasing in 2016.** What is rather remarkable is that, since the beginning of 2016, the restaurant has experienced a tremendous improvement in the quality of the ratings it has received. The ratings reached their earlier 2014 values in the second quarter of 2016, and continued on to an all-time high in 2016-Q4. This period of growth implies that the restaurant has undergone some significant changes in 2016. The final piece to examine is that, since the beginning of 2017, the ratings have dipped slightly, from 4.27 in 2016-Q4 to 4.10.

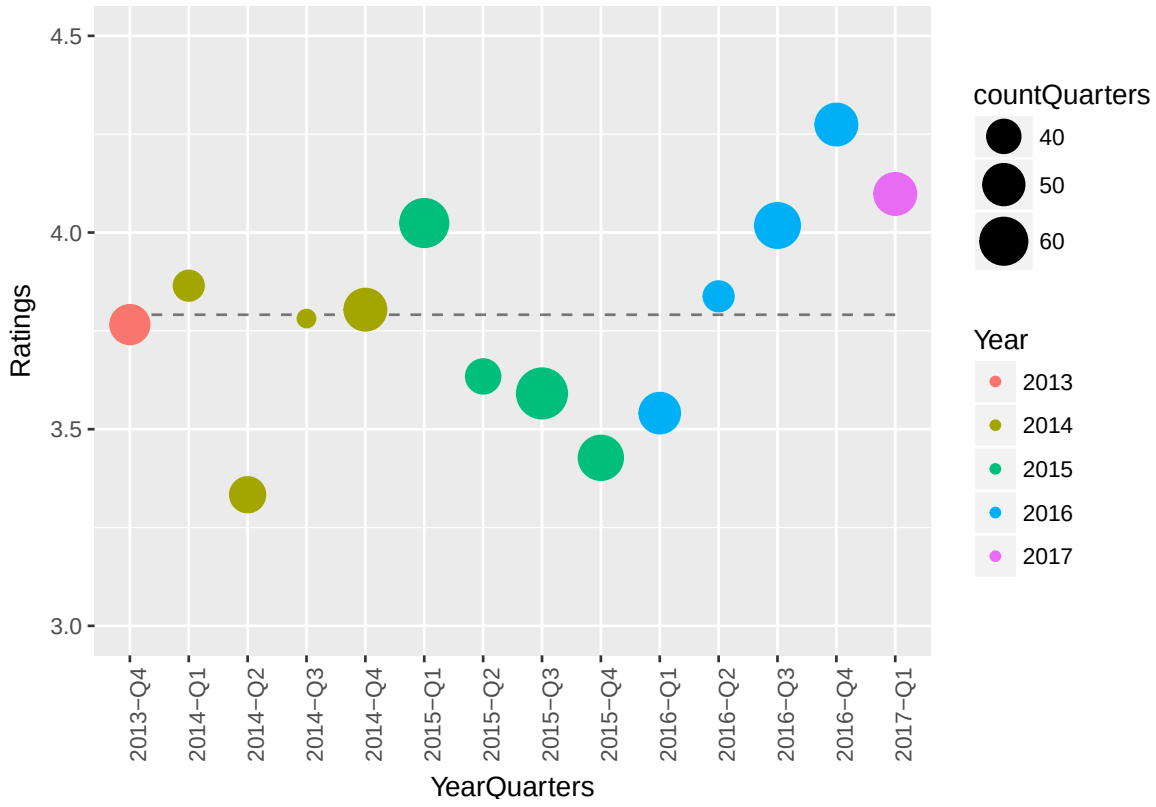

Figure 6: Customer Ratings Averaged by Yearly Quarters

```
CapstoneDF$QuarterGroup <- rep(0, dim(CapstoneDF)[1])
CapstoneDF$QuarterGroup[CapstoneDF$YearQuarters %in% c("2013-Q4", "2014-Q1", "2014-Q3", "20
CapstoneDF$QuarterGroup[CapstoneDF$YearQuarters %in% c("2015-Q2", "2015-Q3", "2015-Q4", "20
CapstoneDF$QuarterGroup[CapstoneDF$YearQuarters %in% c("2016-Q2", "2016-Q3", "2016-Q4", "20
CapstoneDF$QuarterGroup[CapstoneDF$YearQuarters %in% c("2014-Q2", "2015-Q1")] <- "Outliers"
CapstoneDF$QuarterGroup <- factor(CapstoneDF$QuarterGroup)
```

```
#Ratings aggregated by Year and Quarters
global_p3 <- ggplot(CapstoneDF, aes(x=YearQuarters, y=Ratings,col=QuarterGroup)) +
  geom_line(aes(y=RatingsAvg, group=1), linetype="dashed",
            colour="black", alpha=0.5) +
  stat_summary(aes(size=countQuarters),fun.y=mean, geom="point") +
  #coord_cartesian(ylim=c(2.5,4.5)) +
  coord_cartesian(ylim=c(3.0,4.5)) +
  scale_size_continuous(range=c(3,9)) +
  theme(axis.text.x =element_text(angle=90))
global_p3
```

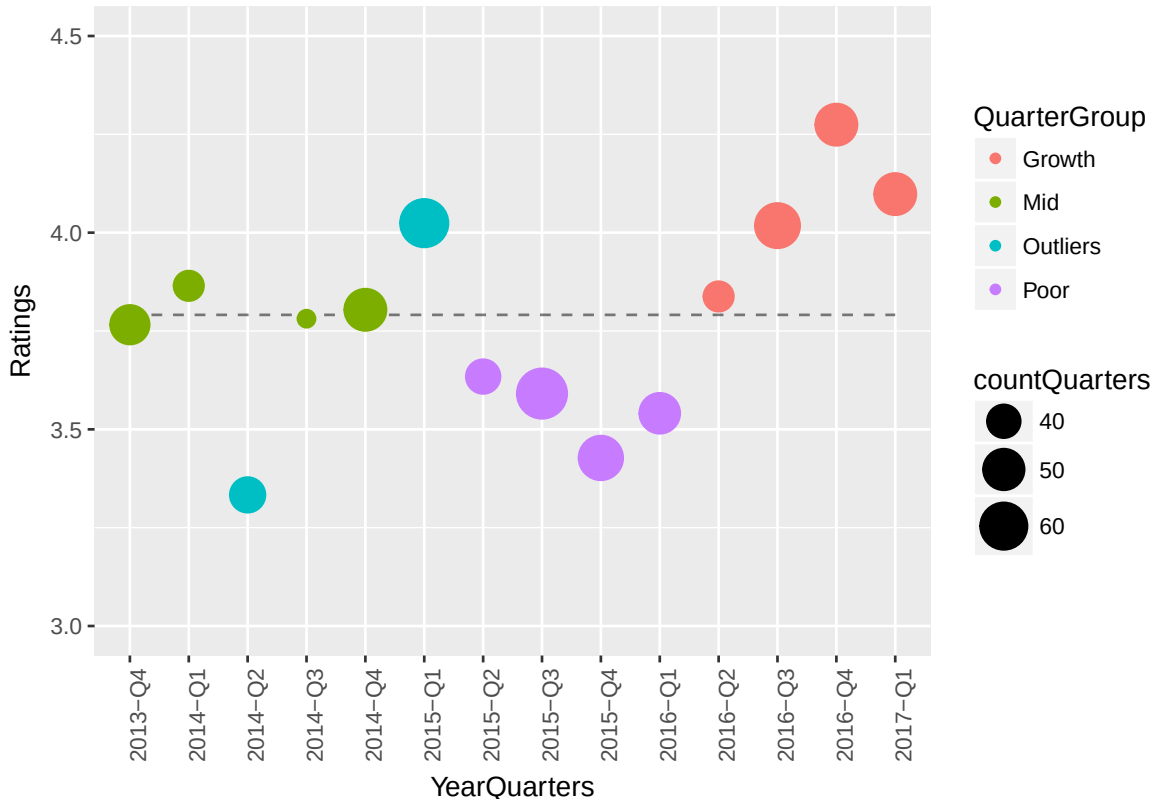

Figure 7: Customer Ratings Averaged by Yearly Quarters, Grouped

```
global_p4 <- ggplot(CapstoneDF, aes(x=YearMonth, y=Ratings, col=Year)) +
    geom_line(aes(y=RatingsAvg, group=1), linetype="dashed",
              colour="black", alpha=0.5) +
  stat_summary(aes(size=countMonth),fun.y=mean, geom="point", alpha=0.8) +
  coord_cartesian(ylim=c(2.5,4.5)) +
  scale_size_continuous(range=c(3,8)) +
  theme(axis.text.x =element_text(angle=90))
global_p4
```

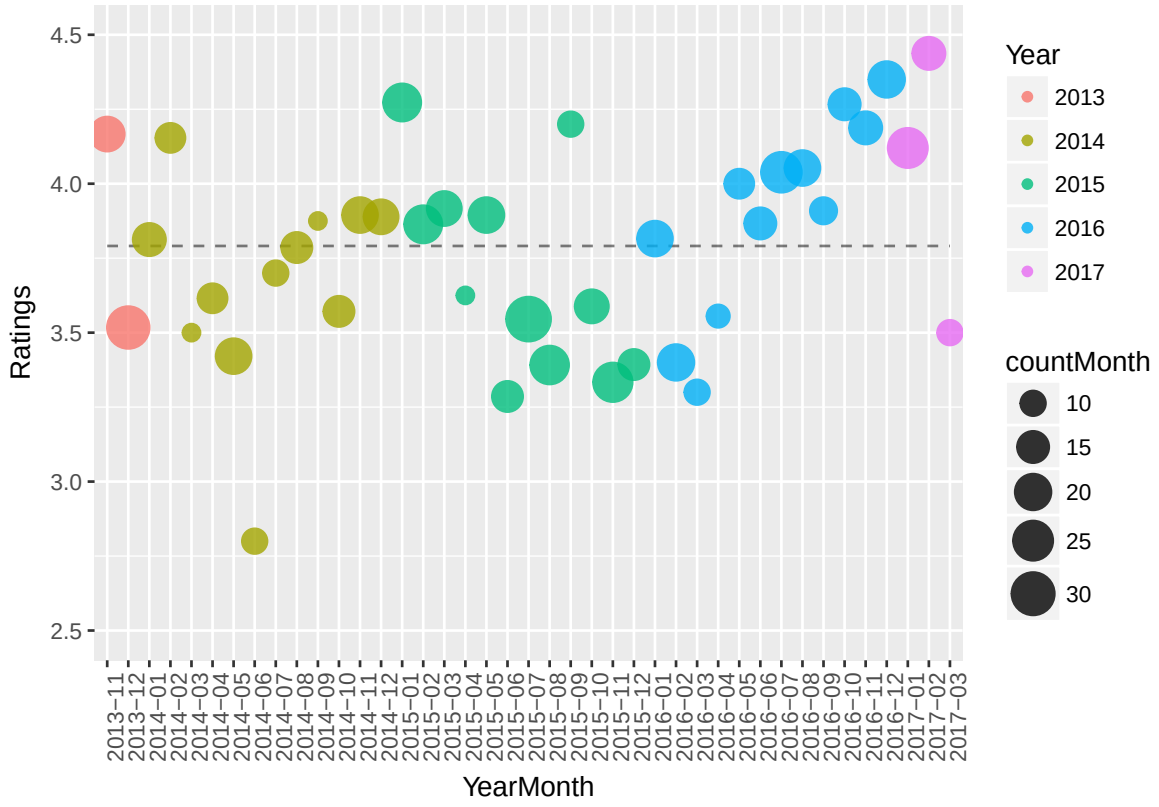

Figure 8: Customer Ratings Averaged by Month

4.2 Customer Reviews Text Analysis

5 References

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