Title

Capstone Project Intermediate Report

Antoine Beauchamp

March 8th, 2017

Contents

1 Introduction

In this report, I intend to elaborate on how we can use tools available in R to acquire and manipulate data from the web. This is incredibly useful if we are interested in performing any sort of text analysis using webpages as our source. I intend to use the tools and methods introduced here to ultimately perform a sentiment analysis of user reviews for REDS Midtown Tavern in Toronto, Ontario. The restaurant's website is located here: www.redsmidtowntavern.com. This report is divided into two primary sections: data gathering and data cleaning. In the former, I discuss the methods and tools I used to gather customer review data for REDS from popular review websites. In the second section, I go into detail about the process I used to clean the raw data obtained from these websites.

Before we dive in, let's load some libraries and set the working directory.

```
CapstoneDir = "/Users/Antoine/Documents/Work/DataScience/Springboard/FoundationsofDataScience/CapstoneP

setwd(CapstoneDir)

rm(list=ls())
```

2 Data Acquisition

As mentioned in the Introduction, the aim of this project is to perform a sentiment analysis of REDS Midtown Tavern, based on available customer reviews. It is well known that there is a wealth of such information on the internet, contained on review and travel websites. For the purpose of this investigation, I have focussed my efforts on the following 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 for us 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 fairly 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/).

2.1 SelectorGadget and CSS Selectors

2.1.1 Identifying Content with SelectorGadget

Before I begin discussing the web scraping process in depth, I want to spend some time discussing SelectorGadget. SelectorGadget is a wonderful open source tool that allows us to obtain CSS selectors on a website by clicking on select items of interest. 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 which 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 doable, this isn't exactly effortless. SelectorGadget greatly improves this process by providing a point-and-click interface between the webpage content and the underlying selector.

I will elaborate on this using a relevant example. Here is the Yelp webpage for REDS Midtown Tavern: https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2. Our goal is to scrape this webpage for the customer reviews, which are located not too far into the page:

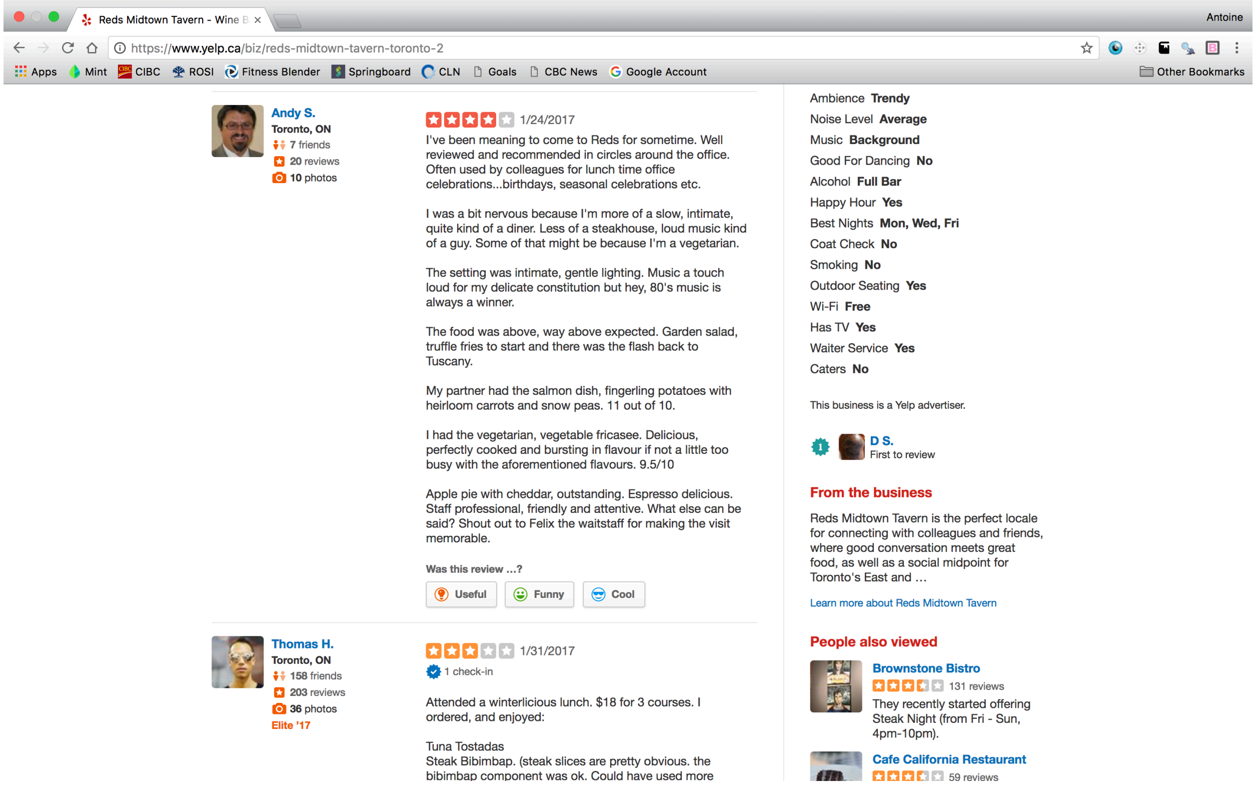
By running the SelectorGadget, which manifests itself as the grey search bar at the bottom of the page, we can select the first review to start identifying the relevant CSS Selector:

We see that the SelectorGadget has returned the CSS Selector "`p`". This selector includes the customer reviews, which is what we want, but it also includes additional content, such as "Was this review …?" and the text under the heading "From the business". We don't need these elements of the webpage. The content that we have selected via point-and-click has been coloured in green, while the content that matches that selector has been highlighted in yellow. This is the initial stage in our 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.

We see that content that had previously been highlighted but is now de-selected becomes red. We also notice that the text "Was this review…?" is now 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. We can verify this by noting that SelectorGadget has found 20 items matching this selector, which corresponds to the number of reviews on this page. Now that we have the selector that we want, we can import the data into R. This will be done in detail below.

2.1.2 CSS Selectors in Detail

When working with SelectorGadget, it is helpful to understand a little about CSS selectors. In essence, CSS selectors allow us to identify given "blocks" of content within an HTML source file. An excellent hands-on tutorial that allows you to get a feel for CSS selectors and how to work with them is found at the following website: https://flukeout.github.io/. This section will loosely follow the layout of this tutorial.



Andy S.
Toronto, ON
7 friends
20 reviews
10 photos

1/24/2017

I've been meaning to come to Reds for sometime. Well reviewed and recommended in circles around the office. Often used by colleagues for lunch time office celebrations...birthdays, seasonal celebrations etc.

I was a bit nervous because I'm more of a slow, intimate, quite kind of a diner. Less of a steakhouse, loud music kind of a guy. Some of that might be because I'm a vegetarian.

The setting was intimate, gentle lighting. Music a touch loud for my delicate constitution but hey, 80's music is always a winner.

The food was above, way above expected. Garden salad, truffle fries to start and there was the flash back to Tuscany.

My partner had the salmon dish, fingerling potatoes with heirloom carrots and snow peas. 11 out of 10.

I had the vegetarian, vegetable fricasee. Delicious, perfectly cooked and bursting in flavour if not a little too busy with the aforementioned flavours. 9.5/10

Apple pie with cheddar, outstanding. Espresso delicious. Staff professional, friendly and attentive. What else can be said? Shout out to Felix the waitstaff for making the visit memorable.

Was this review …?

Thomas H.
Toronto, ON
158 friends
203 reviews
36 photos
Elite '17

1/31/2017
1 check-in

Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:

Tuna Tostadas
Steak Bibimbap. (steak slices are pretty obvious. the bibimbap component was ok. Could have used more

Ambience **Trendy**
Noise Level **Average**
Music **Background**
Good For Dancing **No**
Alcohol **Full Bar**
Happy Hour **Yes**
Best Nights **Mon, Wed, Fri**
Coat Check **No**
Smoking **No**
Outdoor Seating **Yes**
Wi-Fi **Free**
Has TV **Yes**
Waiter Service **Yes**
Caters **No**

This business is a Yelp advertiser.

D S.
First to review

From the business

Reds Midtown Tavern is the perfect locale for connecting with colleagues and friends, where good conversation meets great food, as well as a social midpoint for Toronto's East and …

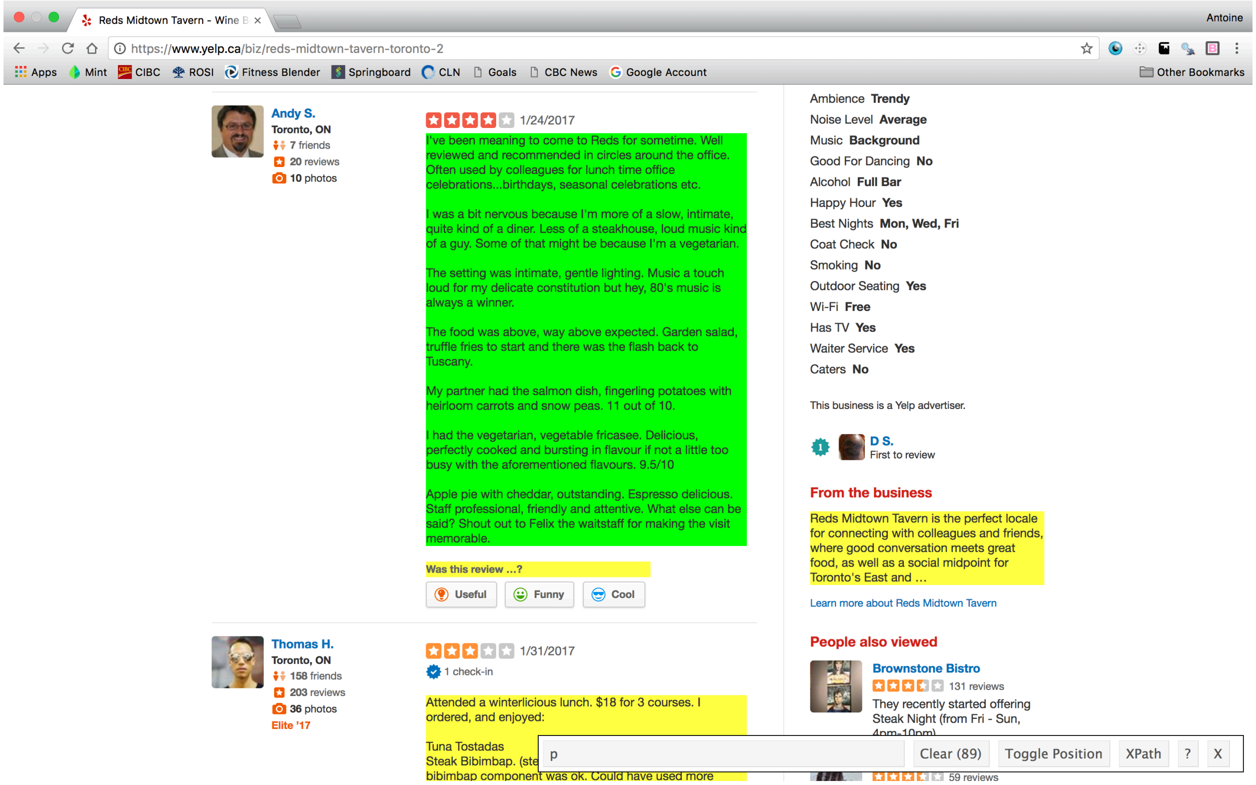
Learn more about Reds Midtown Tavern

People also viewed

Brownstone Bistro
131 reviews
They recently started offering Steak Night (from Fri - Sun, 4pm-10pm).

Cafe California Restaurant
59 reviews

Figure 1: Figure 2.1



Andy S.
Toronto, ON
7 friends
20 reviews
10 photos

1/24/2017

I've been meaning to come to Reds for sometime. Well reviewed and recommended in circles around the office. Often used by colleagues for lunch time office celebrations...birthdays, seasonal celebrations etc.

I was a bit nervous because I'm more of a slow, intimate, quite kind of a diner. Less of a steakhouse, loud music kind of a guy. Some of that might be because I'm a vegetarian.

The setting was intimate, gentle lighting. Music a touch loud for my delicate constitution but hey, 80's music is always a winner.

The food was above, way above expected. Garden salad, truffle fries to start and there was the flash back to Tuscany.

My partner had the salmon dish, fingerling potatoes with heirloom carrots and snow peas. 11 out of 10.

I had the vegetarian, vegetable fricasee. Delicious, perfectly cooked and bursting in flavour if not a little too busy with the aforementioned flavours. 9.5/10

Apple pie with cheddar, outstanding. Espresso delicious. Staff professional, friendly and attentive. What else can be said? Shout out to Felix the waitstaff for making the visit memorable.

Was this review …?

Useful Funny Cool

Thomas H.
Toronto, ON
158 friends
203 reviews
36 photos
Elite '17

1/31/2017
1 check-in

Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:

Tuna Tostadas
Steak Bibimbap. (ste
bibimbap component was ok. Could have used more

Ambience Trendy
Noise Level Average
Music Background
Good For Dancing No
Alcohol Full Bar
Happy Hour Yes
Best Nights Mon, Wed, Fri
Coat Check No
Smoking No
Outdoor Seating Yes
Wi-Fi Free
Has TV Yes
Waiter Service Yes
Caters No

This business is a Yelp advertiser.

D S.
First to review

From the business

Reds Midtown Tavern is the perfect locale for connecting with colleagues and friends, where good conversation meets great food, as well as a social midpoint for Toronto's East and …

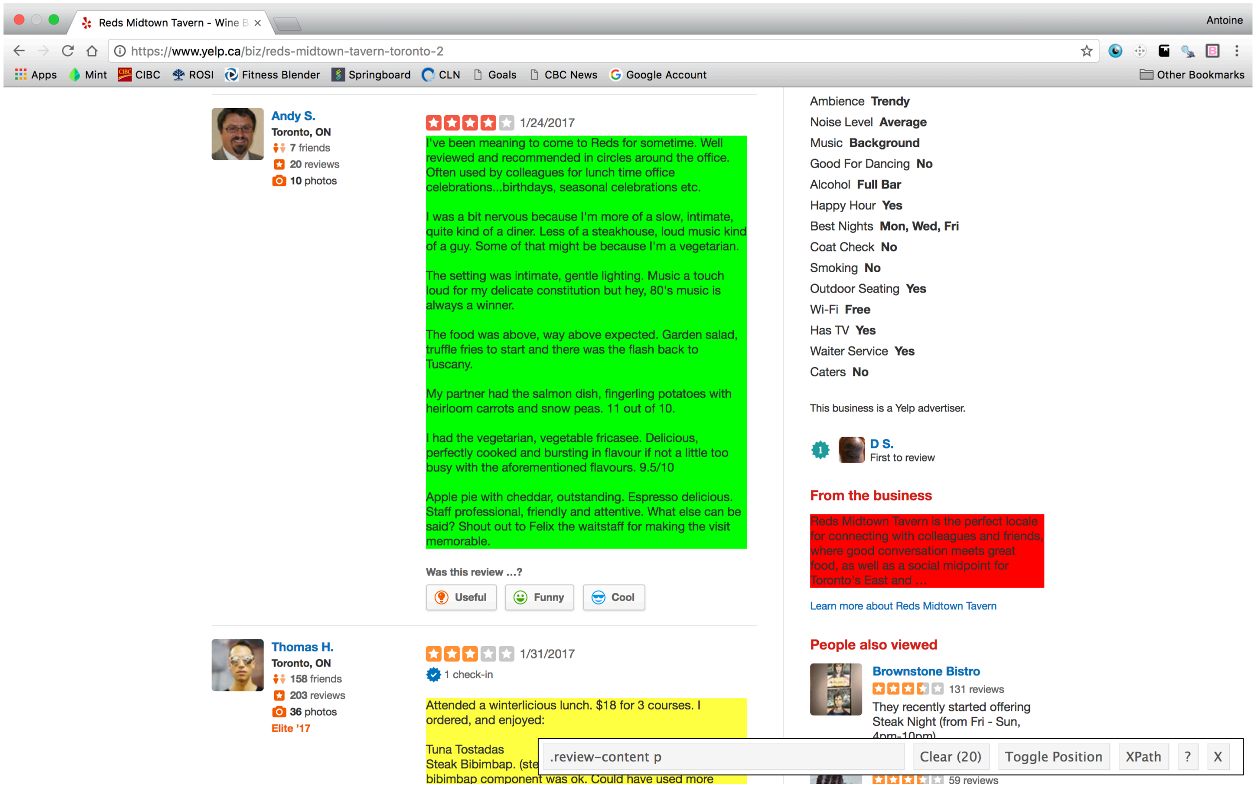
Learn more about Reds Midtown Tavern

People also viewed

Brownstone Bistro
131 reviews
They recently started offering Steak Night (from Fri - Sun, 4pm-10pm).

59 reviews

Figure 2: Figure 2.2



Andy S.
Toronto, ON
7 friends
20 reviews
10 photos

1/24/2017

I've been meaning to come to Reds for sometime. Well reviewed and recommended in circles around the office. Often used by colleagues for lunch time office celebrations...birthdays, seasonal celebrations etc.

I was a bit nervous because I'm more of a slow, intimate, quite kind of a diner. Less of a steakhouse, loud music kind of a guy. Some of that might be because I'm a vegetarian.

The setting was intimate, gentle lighting. Music a touch loud for my delicate constitution but hey, 80's music is always a winner.

The food was above, way above expected. Garden salad, truffle fries to start and there was the flash back to Tuscany.

My partner had the salmon dish, fingerling potatoes with heirloom carrots and snow peas. 11 out of 10.

I had the vegetarian, vegetable fricasee. Delicious, perfectly cooked and bursting in flavour if not a little too busy with the aforementioned flavours. 9.5/10

Apple pie with cheddar, outstanding. Espresso delicious. Staff professional, friendly and attentive. What else can be said? Shout out to Felix the waitstaff for making the visit memorable.

Thomas H.
Toronto, ON
158 friends
203 reviews
36 photos
Elite '17

1/31/2017
1 check-in

Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:

Tuna Tostadas
Steak Bibimbap. (ste
bibimbap component was ok. Could have used more

Ambience	**Trendy**
Noise Level	**Average**
Music	**Background**
Good For Dancing	**No**
Alcohol	**Full Bar**
Happy Hour	**Yes**
Best Nights	**Mon, Wed, Fri**
Coat Check	**No**
Smoking	**No**
Outdoor Seating	**Yes**
Wi-Fi	**Free**
Has TV	**Yes**
Waiter Service	**Yes**
Caters	**No**

This business is a Yelp advertiser.

D S.
First to review

From the business

Reds Midtown Tavern is the perfect locale for connecting with colleagues and friends, where good conversation meets great food, as well as a social midpoint for Toronto's East and ...

Learn more about Reds Midtown Tavern

People also viewed

Brownstone Bistro
131 reviews
They recently started offering Steak Night (from Fri - Sun, 4pm-10pm)

59 reviews

.review-content p

Figure 3: Figure 2.3

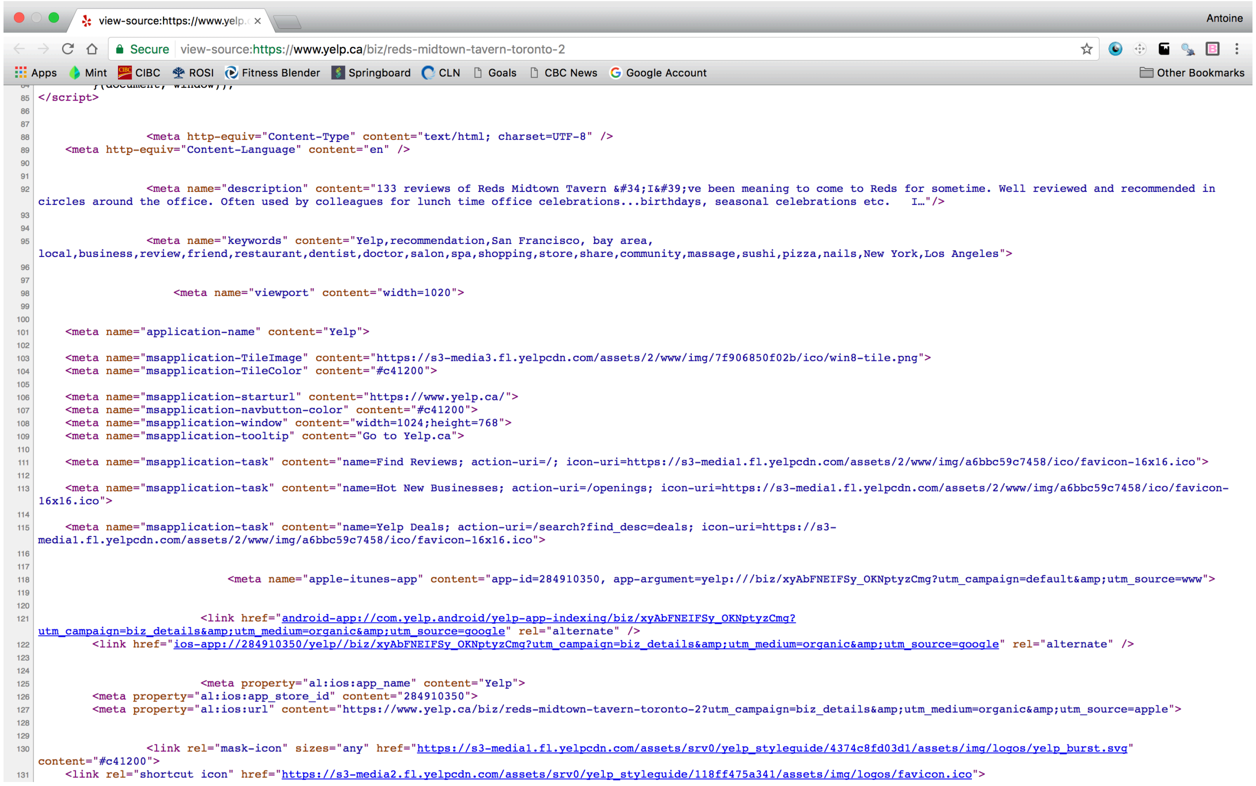
An HTML file is comprised of different blocks of content, which are identified by their **type**. For example, `<p>` represents a paragraph block, `<a>` a hyperlink, `<h1>` a heading, and so on. An screenshot of the HTML source file for https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2 is included in Figure 2.4. 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`.

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"`.



```
        })(document, window));
</script>


                <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="Content-Language" content="en" />


                <meta name="description" content="133 reviews of Reds Midtown Tavern &#34;I&#39;ve been meaning to come to Reds for sometime. Well reviewed and recommended in
circles around the office. Often used by colleagues for lunch time office celebrations...birthdays, seasonal celebrations etc.    I…"/>


                <meta name="keywords" content="Yelp,recommendation,San Francisco, bay area,
local,business,review,friend,restaurant,dentist,doctor,salon,spa,shopping,store,share,community,massage,sushi,pizza,nails,New York,Los Angeles">


                    <meta name="viewport" content="width=1020">


    <meta name="application-name" content="Yelp">

    <meta name="msapplication-TileImage" content="https://s3-media3.fl.yelpcdn.com/assets/2/www/img/7f906850f02b/ico/win8-tile.png">
    <meta name="msapplication-TileColor" content="#c41200">

    <meta name="msapplication-starturl" content="https://www.yelp.ca/">
    <meta name="msapplication-navbutton-color" content="#c41200">
    <meta name="msapplication-window" content="width=1024;height=768">
    <meta name="msapplication-tooltip" content="Go to Yelp.ca">

    <meta name="msapplication-task" content="name=Find Reviews; action-uri=/; icon-uri=https://s3-media1.fl.yelpcdn.com/assets/2/www/img/a6bbc59c7458/ico/favicon-16x16.ico">

    <meta name="msapplication-task" content="name=Hot New Businesses; action-uri=/openings; icon-uri=https://s3-media1.fl.yelpcdn.com/assets/2/www/img/a6bbc59c7458/ico/favicon-
16x16.ico">

    <meta name="msapplication-task" content="name=Yelp Deals; action-uri=/search?find_desc=deals; icon-uri=https://s3-
media1.fl.yelpcdn.com/assets/2/www/img/a6bbc59c7458/ico/favicon-16x16.ico">


                            <meta name="apple-itunes-app" content="app-id=284910350, app-argument=yelp:///biz/xyAbFNEIFSy_OKNptyzCmg?utm_campaign=default&amp;utm_source=www">


                        <link href="android-app://com.yelp.android/yelp-app-indexing/biz/xyAbFNEIFSy_OKNptyzCmg?
utm_campaign=biz_details&amp;utm_medium=organic&amp;utm_source=google" rel="alternate" />
        <link href="ios-app://284910350/yelp//biz/xyAbFNEIFSy_OKNptyzCmg?utm_campaign=biz_details&amp;utm_medium=organic&amp;utm_source=google" rel="alternate" />


                        <meta property="al:ios:app_name" content="Yelp">
        <meta property="al:ios:app_store_id" content="284910350">
        <meta property="al:ios:url" content="https://www.yelp.ca/biz/reds-midtown-tavern-toronto-2?utm_campaign=biz_details&amp;utm_medium=organic&amp;utm_source=apple">


                <link rel="mask-icon" sizes="any" href="https://s3-media1.fl.yelpcdn.com/assets/srv0/yelp_styleguide/4374c8fd03d1/assets/img/logos/yelp_burst.svg"
content="#c41200">
    <link rel="shortcut icon" href="https://s3-media2.fl.yelpcdn.com/assets/srv0/yelp_styleguide/118ff475a341/assets/img/logos/favicon.ico">
```

Figure 4: Figure 2.4

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()`.

```r
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">I've been meaning to come to Reds for sometime. Well re ...
## [2] <p lang="en">Attended a winterlicious lunch. $18 for 3 courses. I or ...
## [3] <p lang="en">As a young barrister  I often visited Red's older, fanc ...
## [4] <p lang="en">I came here for dinner and had wine and entrees. I wish ...
## [5] <p lang="en">I had to try out Reds for Summerlicious! It was a 3 cou ...
## [6] <p lang="en">Their gin and tonic nights are amazing and come in the  ...
```

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\">I've been meaning to come to Reds for sometime. Well reviewed and recommended in circl
## [2] "<p lang=\"en\">Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:<br><br>Tun
```

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] "I've been meaning to come to Reds for sometime. Well reviewed and recommended in circles around the of
## [2] "Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:Tuna TostadasSteak Bibimbap
```

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"
```

```r
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()`.

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

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

```r
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`.

```r
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      <img class
```

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.

```r
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" "3.0 star rating"
## [5] "3.0 star rating" "4.0 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 used to scrape review data from Yelp.com
# Arguments:
# BaseURL: URL to the first 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.chara

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

```r
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_tex

    #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 is used to scrape review data from Trip Advisor
# Arguments:
# LandingURL: This is the URL for the landing page of the restaurant you want to scrape for.
# 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-number") %>% as
      #Grab the hyperlinks for the subsequent pages
      hyperlink <- read_html(page_url) %>% html_nodes(".pageNum") %>% html_attr("href") %>% as.characte
      #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_c
    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("a
  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.

```r
#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=2017-02-22+19%3A0

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

#Zomato main review page URL
BaseURL_Zomato <- "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 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.

```r
#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 [1:133] "I've been meaning to come to Reds for sometime. Well reviewed and recommended in c
##  $ Ratings: chr [1:135] "4.0 star rating" "3.0 star rating" "4.0 star rating" "3.0 star rating" ...
##  $ Dates  : chr [1:135] "\n      1/24/2017\n   " "\n      1/31/2017\n   " "\n      1/10/2017\n   " "\n
##  $ PrevRev: chr [1:135] "<span class=\"rating-qualifier\">\n      1/24/2017\n   </span>" "<span class=\"
```

```
str(OpenTableData)
```

```
## List of 3
##  $ Reviews: chr [1:306] "Good food and service. Nice atmosphere. Nothing particularly outstanding but stil
##   $ Ratings: chr  [1:306] "3" "5" "3" "5" ...
##  $ Dates  : chr [1:306] "Dined on February 27, 2017" "Dined on February 23, 2017" "Dined on February 19, 201
```

```
str(TripAdData)
```

```
## List of 4
##  $ Reviews: chr [1:220] "\nThis place is deservedly well-known for its gin drinks, which we started with, c
##  $ Ratings: chr [1:220] "4 of 5 bubbles" "4 of 5 bubbles" "5 of 5 bubbles" "5 of 5 bubbles" ...
##  $ Dates1 : chr [1:9] "5 March 2017" "1 March 2017" "20 February 2017" "17 February 2017" ...
##  $ Dates2 : chr [1:220] "Reviewed yesterday\nNEW " "Reviewed 5 days ago\nNEW " "Reviewed 2 weeks ago\n" "Re
```

```
str(ZomatoData)
```

```
## List of 3
##  $ Reviews: chr [1:11] "\n           Rated \n                     Had lunch at Reds on an extremely hot d
##   $ 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" "2015-12-23 00:1
```

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.

```r
#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=Yel
```

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.

```r
# 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=OpenTable
```

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

```r
#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, W
```

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

```r
#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,We
```

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.

```r
#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':  669 obs. of  4 variables:
## $ Reviews: chr  "I've been meaning to come to Reds for sometime. Well reviewed and recommended in circles a
## $ Ratings: chr  "4.0 star rating" "3.0 star rating" "4.0 star rating" "3.0 star rating" ...
## $ Dates  : chr "\n      1/24/2017\n   " "\n      1/31/2017\n   " "\n      1/10/2017\n   " "\n      12/27/2
##  $ Website: chr  "Yelp" "Yelp" "Yelp" "Yelp" ...
##  - attr(*, "vars")=List of 1
##   ..$ : symbol Website
##  - attr(*, "drop")= logi TRUE
##  - attr(*, "indices")=List of 4
##   ..$ : int  133 134 135 136 137 138 139 140 141 142 ...
##   ..$ : int  449 450 451 452 453 454 455 456 457 458 ...
##   ..$ : int  0 1 2 3 4 5 6 7 8 9 ...
##   ..$ : int  439 440 441 442 443 444 445 446 447 448
##  - attr(*, "group_sizes")= int  306 220 133 10
##  - attr(*, "biggest_group_size")= int 306
##  - 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:669         Length:669         Length:669        
##  Class :character   Class :character   Class :character  
##  Mode  :character   Mode  :character   Mode  :character  
##    Website         
##  Length:669        
##  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        1/24/2017\n    "  "\n        1/31/2017\n    " 
##  [3] "\n        1/10/2017\n    "  "\n        12/27/2016\n    "
##  [5] "\n        7/12/2016\n    "  "\n        7/29/2016\n    " 
##  [7] "\n        11/10/2016\n    " "\n        6/17/2016\n    " 
##  [9] "\n        11/15/2016\n    " "\n        11/6/2016\n    "
```

The first thing we need to do is 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] "1/24/2017"  "1/31/2017"  "1/10/2017"  "12/27/2016" "7/12/2016" 
##  [6] "7/29/2016"  "11/10/2016" "6/17/2016"  "11/15/2016" "11/6/2016"
```

That looks better. 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 February 27, 2017" "Dined on February 23, 2017"
##  [5] "Dined on February 19, 2017" "Dined on February 19, 2017"
##  [7] "Dined on February 19, 2017" "Dined on February 15, 2017"
##  [9] "Dined on February 12, 2017" "Dined on February 9, 2017"
```

Following 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)
```

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

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

```
##  [1] "February 27, 2017" "February 23, 2017" "February 19, 2017"
##  [4] "February 19, 2017" "February 19, 2017" "February 15, 2017"
##  [7] "February 12, 2017" "February 9, 2017"  "February 6, 2017"
## [10] "February 4, 2017"  "January 31, 2017"  "January 30, 2017"
## [13] "January 28, 2017"  "January 28, 2017"  "January 26, 2017"
## [16] "January 21, 2017"  "January 4, 2017"   "January 2, 2017"
## [19] "January 1, 2017"   "January 1, 2017"
```

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

```
##  [1] "5 March 2017"              "1 March 2017"
##  [3] "20 February 2017"          "17 February 2017"
##  [5] "9 February 2017"           "6 February 2017"
##  [7] "2 February 2017"           "2 February 2017"
##  [9] "31 January 2017"           "Reviewed 29 January 2017"
## [11] "Reviewed 28 January 2017"  "Reviewed 27 January 2017"
## [13] "Reviewed 24 January 2017"  "Reviewed 19 January 2017"
## [15] "Reviewed 13 January 2017"  "Reviewed 5 January 2017"
## [17] "Reviewed 1 January 2017"   "Reviewed 29 December 2016"
## [19] "Reviewed 26 December 2016" "Reviewed 8 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] "5 March 2017"     "1 March 2017"     "20 February 2017"
##  [4] "17 February 2017" "9 February 2017"  "6 February 2017"
##  [7] "2 February 2017"  "2 February 2017"  "31 January 2017"
## [10] "29 January 2017"  "28 January 2017"  "27 January 2017"
## [13] "24 January 2017"  "19 January 2017"  "13 January 2017"
## [16] "5 January 2017"   "1 January 2017"   "29 December 2016"
## [19] "26 December 2016" "8 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] "1/24/2017"  "1/31/2017"  "1/10/2017"  "12/27/2016" "7/12/2016"
## [6] "7/29/2016"
```

```
#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/%Y")
```

Open Table:

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

```
## [1] "February 27, 2017" "February 23, 2017" "February 19, 2017"
## [4] "February 19, 2017" "February 19, 2017" "February 15, 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,
```

Trip Advisor:

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

```
## [1] "5 March 2017"     "1 March 2017"     "20 February 2017"
## [4] "17 February 2017" "9 February 2017"  "6 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$Dates[which(CapstoneDF$Website ==
```

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[1:669], format: "2017-01-24" "2017-01-31" "2017-01-10" "2016-12-27" ...
```

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" "3.0 star rating"
## [5] "3.0 star rating" "4.0 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" "3" "5" "5" "5"
```

Looks good already.

Trip Advisor

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

```
## [1] "4 of 5 bubbles" "4 of 5 bubbles" "5 of 5 bubbles" "5 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 [1:669] 4 3 4 3 3 4 1 3 4 5 ...
```

3.3 Final Touches

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

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 to sit on t
## [2] "          Rated                      My wife & I were looking for place to try for the first time ni
```

```
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 patio even.  T
## [2] "                      My wife & I were looking for place to try for the first time nice place seating
```

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

4 Exploratory Data Analysis

```
library(readr)
library(dplyr)
library(tm)
```

```
## Loading required package: NLP
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

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

```
## Parsed with column specification:
## cols(
##   Reviews = col_character(),
##   Ratings = col_double(),
##   Dates = col_date(format = ""),
##   Website = col_character()
## )
```

```
CapstoneCorpus <- CapstoneDF$Reviews %>% VectorSource() %>% Corpus()
inspect(CapstoneCorpus[1:2])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 2
##
## [1] I've been meaning to come to Reds for sometime. Well reviewed and recommended in circles around the off
## [2] Attended a winterlicious lunch. $18 for 3 courses. I ordered, and enjoyed:Tuna TostadasSteak Bibimbap
```

```
CapstoneCorpus <- CapstoneCorpus %>% tm_map(removePunctuation)
CapstoneCorpus <- CapstoneCorpus %>% tm_map(removeWords, stopwords("english"))
inspect(CapstoneCorpus[1:2])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 2
##
## [1] Ive  meaning  come  Reds  sometime Well reviewed  recommended  circles around  office Often used  collea
## [2] Attended  winterlicious lunch 18  3 courses I ordered  enjoyedTuna TostadasSteak Bibimbap steak slices
```

```r
TermDocMatrix <- TermDocumentMatrix(CapstoneCorpus)
inspect(TermDocMatrix)
```

```
## <<TermDocumentMatrix (terms: 4957, documents: 669)>>
## Non-/sparse entries: 25710/3290523
## Sparsity           : 99%
## Maximal term length: 32
## Weighting          : term frequency (tf)
## Sample             :
##          Docs
## Terms     117 120 17 26 3 593 631 652 87 90
##   back      6   0  0  0 1   6   1   1  0  2
##   food      2   0  2  0 2   2   2   2  0  1
##   good      2   0  0  2 0   2   1   0  1  5
##   great     0   1  0  0 0   0   2   3  2  0
##   menu      0   2  1  1 1   0   0   4  3  1
##   nice      0   0  3  1 0   0   1   1  0  2
##   place     0   0  3  0 0   0   0   0  1  0
##   reds      2   1  1  0 4   2   1   2  5  2
##   service   2   0  0  1 0   2   0   0  0  0
##   the       1   1  5 19 3   1   6   5 11  0
```

```r
wordmatrix <- as.matrix(TermDocMatrix)

wordfreqs = sort(rowSums(wordmatrix), decreasing=TRUE)
head(wordfreqs)
```

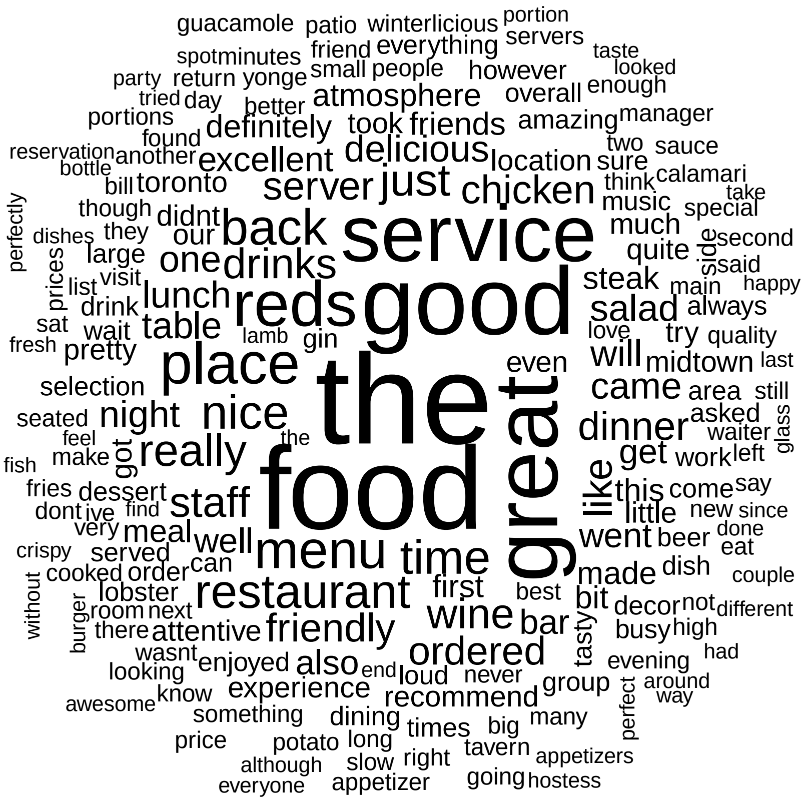
```
##     the    food    good   great service    reds
##     561     509     403     374     319     236
```

```r
# create a data frame with words and their frequencies
WordFreqDF = data_frame(word=names(wordfreqs), freq=wordfreqs)

head(WordFreqDF)
```

```
## # A tibble: 6 × 2
##      word  freq
##     <chr> <dbl>
## 1     the   561
## 2    food   509
## 3    good   403
## 4   great   374
## 5 service   319
## 6    reds   236
```

```r
wordcloud(CapstoneCorpus, min.freq=2,scale=c(4,0.5), max.word=200, random.order=F)
```



```
findAssocs(TermDocMatrix, "good", .20)
```

```
## $good
##     thursday surprisingly
##         0.21         0.20
```

```
findAssocs(TermDocMatrix, "service", .18)
```

```
## $service
##    missing       food apologized       face   appeared     assume
##       0.22       0.21       0.20       0.20       0.19       0.19
##    apology
##       0.18
```

```
findAssocs(TermDocMatrix,"food",0.20)
```

```
## $food
##    came service  drinks
##    0.22    0.21    0.21
```

```
findAssocs(TermDocMatrix, "great", .20)
```

```
## $great
##          right       college           say       article     awkwardly
##           0.23          0.23          0.22          0.21          0.21
##       blasting       goodbye       idiotic       lawyers          nook
##           0.21          0.21          0.21          0.21          0.21
##    obligingly       outcast         plant       roomour   salmontotal
##           0.21          0.21          0.21          0.21          0.21
## smallishgroup           sol        space2         those       though1
##           0.21          0.21          0.21          0.21          0.21
##    underneath         usour          were         jessy         lower
##          0.21          0.21          0.21          0.21          0.21
```

```
findAssocs(TermDocMatrix, "drinks", .25)
```

```
## $drinks
##      accross         apon       be3ing     differnt drinkgetting
##         0.31         0.31         0.31         0.31         0.31
##     elephent         fith     freezing   horriblewe    inbetween
##         0.31         0.31         0.31         0.31         0.31
##       jumper      mission      nocking        shirt      singlet
##         0.31         0.31         0.31         0.31         0.31
##   snowjacket         ther     thurmals       pushed        girls
##         0.31         0.31         0.31         0.26         0.25
##    appealing
##         0.25
```

```
findAssocs(TermDocMatrix, "dinner", .33)
```

```
## $dinner
##      restaurants             outi            leave              160
##             0.40             0.36             0.34             0.33
##              710         absolute       additional            angry
##             0.33             0.33             0.33             0.33
##      apologizing              are        argentina          capping
##             0.33             0.33             0.33             0.33
##           caring           cellar             chin       choicethey
##             0.33             0.33             0.33             0.33
##             corp           cotedu      cotedurhone             crop
##             0.33             0.33             0.33             0.33
## disappointmentwe          emailed         european          excuses
##             0.33             0.33             0.33             0.33
##              few            forth          freebie        genuinely
##             0.33             0.33             0.33             0.33
##            hands           injury         insisted           insult
##             0.33             0.33             0.33             0.33
##            italy           kermit           leaned            learn
##             0.33             0.33             0.33             0.33
##            lynch          minimum           munute             owns
##             0.33             0.33             0.33             0.33
##        proceeded           region         regional       reiterated
##             0.33             0.33             0.33             0.33
##            rhone       rightfully          selling            smirk
##             0.33             0.33             0.33             0.33
##         smirking         students           survey           weekly
##             0.33             0.33             0.33             0.33
##              wer
##             0.33
```

```
findAssocs(TermDocMatrix, "time", .25)
```

```
## $time
##          first      everybody      finishing       great2nd         itmain
##           0.34           0.27           0.27           0.27           0.27
##       lunch1st      okaythere   platedessert        slivers slowappetizers
##           0.27           0.27           0.27           0.27           0.27
## strongambiance       thousand        despite       gazpacho            pan
##           0.27           0.27           0.27           0.26           0.26
```

```
findAssocs(TermDocMatrix, "atmosphere", .20)
```

```
## $atmosphere
##        loaded      cilantro      decently           155     355shrimp
##          0.25          0.23          0.22          0.22          0.22
##      allvalue          bleh         chile     chipsthis  decorservice
##          0.22          0.22          0.22          0.22          0.22
##     gossiping       hanging        hoisin hoisinsiracha  juiceserrano
##          0.22          0.22          0.22          0.22          0.22
##    refreshing    restosfood       robotic         sunny   tastysticky
##          0.22          0.22          0.22          0.22          0.22
## tortillasthis         wayyy        wonton         baked
##          0.22          0.22          0.22          0.20
```

```
findAssocs(TermDocMatrix, "friendly", .25)
```

```
## $friendly
##             staff               3oz              56oz           baggage
##              0.28              0.26              0.26              0.26
##            chinsy       complaining             crack          downhill
##              0.26              0.26              0.26              0.26
##            excess             gotta          grateful        impersonal
##              0.26              0.26              0.26              0.26
##             knows          outgoing portionnegativeit     reciprocation
##              0.26              0.26              0.26              0.26
##         slightest   straightforward            subpar               tag
##              0.26              0.26              0.26              0.26
##         welcomeno     whatsoevershe             youno             alice
##              0.26              0.26              0.26              0.26
##             andor         attatched             being             billy
##              0.26              0.26              0.26              0.26
##              bomb           complex          fazoolis          foodwhen
##              0.26              0.26              0.26              0.26
##              foot             shops      springsummer            stands
##              0.26              0.26              0.26              0.26
```