Web Scraping, APIs, Bag O' Words

Department of Biostatistics and Bioinformatics

Steve Pittard wsp@emory.edu

April 25, 2018

Motivations - Everybody Puts Stuff on the Internet

Motivations

The web hosts lots of interesting data that you can "scrape" for later use:

- **tables:** Fetch tables like from Wikipedia
- **forms:** You can submit forms and fetch the result
- **css:** You can access parts of a web site using style or css selectors
- **Tweets: Process tweets including emojis**
- **Web Sites: User forums have lots of content**
- **Instagram: Yes you can "scrape" photos also**

Motivations

Look at the Wikipedia page for world population by country
`https://en.wikipedia.org/wiki/World_population`

- We can get any table we want using rvest
- We might have to experiment to figure out which one
- Get the one that lists the ten most populous countries
- I think this might be the 4th table on the page
- How do we get this ?

Motivations

The web has its own languages: HTML, CSS, Javascript

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

My First Heading

My first paragraph.

First	Last
Steve	Jones
Mary	Smith
Frank	Collins

Motivations

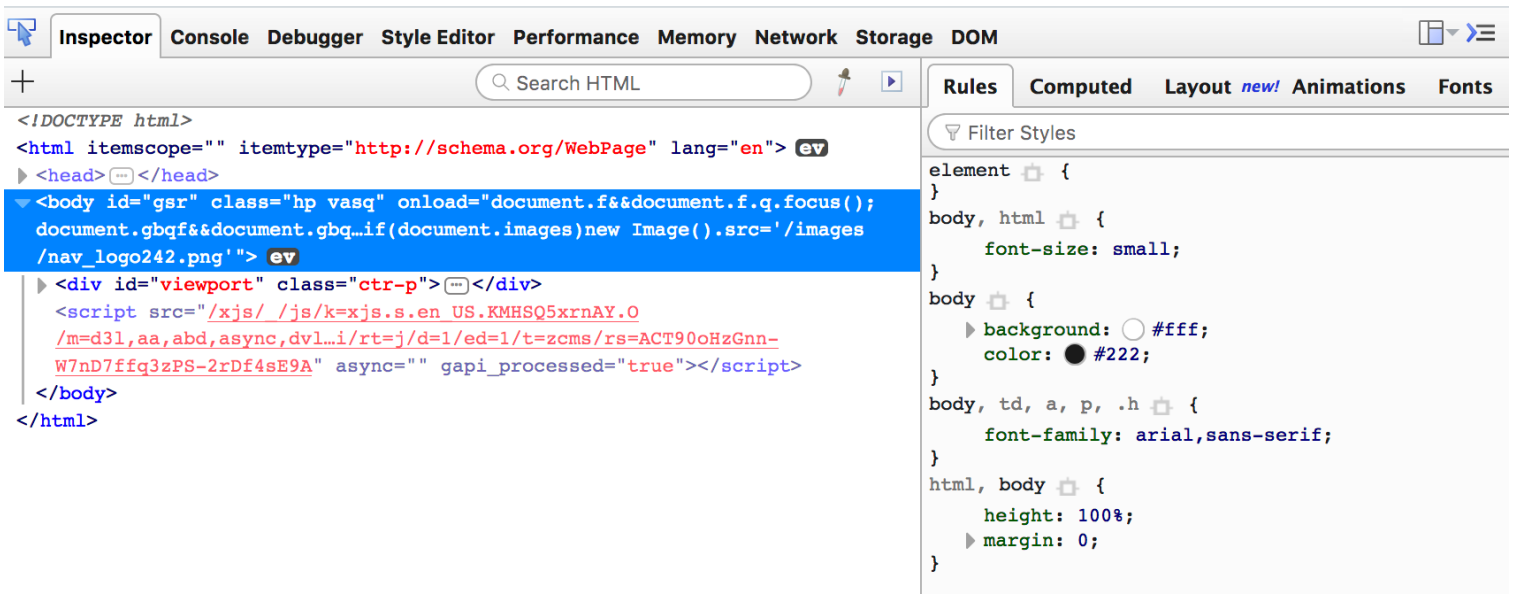
```
h1 {
    color: red;
}

h2 {
    color: red;
}

p {
    color: blue;
}
table, th, td {
   border: 1px solid green;
}
```

My First Heading

My first paragraph.

First	Last
Steve	Jones
Mary	Smith
Frank	Collins

Motivations



Looking Under The Hood

There are some browser plugins that identify sections of web pages

- Selector Gadget `http://selectorgadget.com/`
- Firebug `https://getfirebug.com/`
- Google Chrome - Right click to inspect a page element
- Google Chrome - View Developer - Developer Tools

Example

Grab the data associated with this table - Google Devtools

10 most populous countries

Rank	Country / Territory	Population	Date	Approx. % of world population	Source
1	China[note 4]	1,387,520,000	November 15, 2017	18.3%	[91]
2	India	1,323,930,000	November 15, 2017	17.5%	[92]
3	United States	326,111,000	November 15, 2017	4.3%	[93]
4	Indonesia	261,600,000	October 31, 2016	3.45%	[94]
5	Pakistan	209,585,000	November 15, 2017	2.76%	[95]
6	Brazil	208,264,000	November 15, 2017	2.75%	[96]
7	Nigeria	188,500,000	October 31, 2016	2.49%	[97]
8	Bangladesh	163,483,000	November 15, 2017	2.16%	[98]
9	Russia[note 5]	146,773,226	June 1, 2017	1.94%	[99]
10	Japan	126,750,000	July 1, 2017	1.67%	[100]

Example

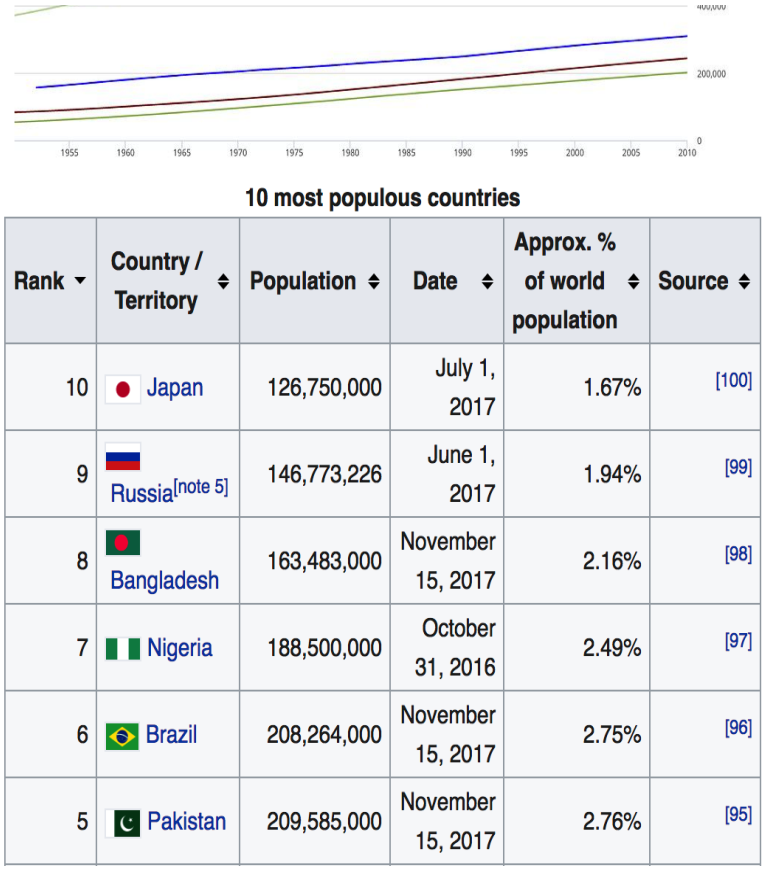
Grab the data associated with this table - Selector Gadget

10 most populous countries

Rank	Country / Territory	Population	Date	Approx. % of world population	Source
1	China[note 4]	1,387,520,000	November 15, 2017	18.3%	[91]
2	India	1,323,930,000	November 15, 2017	17.5%	[92]
3	United States	326,111,000	November 15, 2017	4.3%	[93]
4	Indonesia	261,600,000	October 31, 2016	3.45%	[94]
5	Pakistan	209,585,000	November 15, 2017	2.76%	[95]
6	Brazil	208,264,000	November 15, 2017	2.75%	[96]
7	Nigeria	188,500,000	October 31, 2016	2.49%	[97]
8	Bangladesh	163,483,000	November 15, 2017	2.16%	[98]
9	Russia[note 5]	146,773,226	June 1, 2017	1.94%	[99]
10	Japan	126,750,000	July 1, 2017	1.67%	[100]

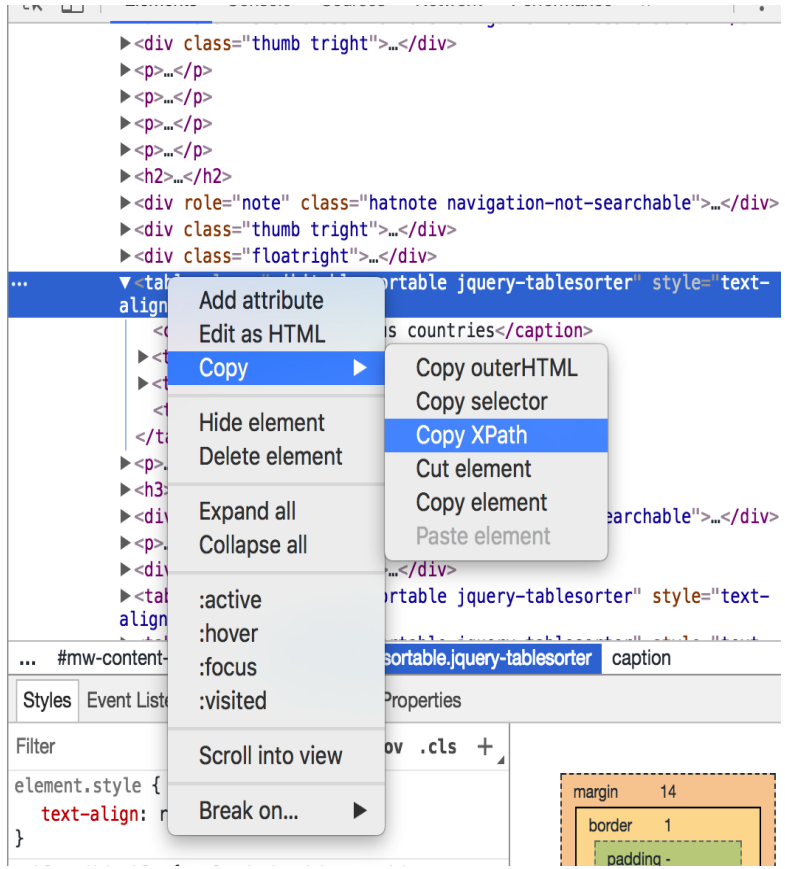
Example

Find out the right path to this element



10 most populous countries

Rank	Country / Territory	Population	Date	Approx. % of world population	Source
10	Japan	126,750,000	July 1, 2017	1.67%	[100]
9	Russia[note 5]	146,773,226	June 1, 2017	1.94%	[99]
8	Bangladesh	163,483,000	November 15, 2017	2.16%	[98]
7	Nigeria	188,500,000	October 31, 2016	2.49%	[97]
6	Brazil	208,264,000	November 15, 2017	2.75%	[96]
5	Pakistan	209,585,000	November 15, 2017	2.76%	[95]



Example

```
library(rvest)

url <- "https://en.wikipedia.org/wiki/World_population"

ten_most_populous <- read_html(url)

str(ten_most_populous)
List of 2
$ node:<externalptr>
  $ doc :<externalptr>
  - attr(*, "class")= chr [1:2] "xml_document" "xml_node"

# Get some content

ten_most_populous %>%
   html_nodes(xpath='//*[@id="mw-content-text"]/div/table[5]') %>%
   html_table() -> ten_most_df
```

Example

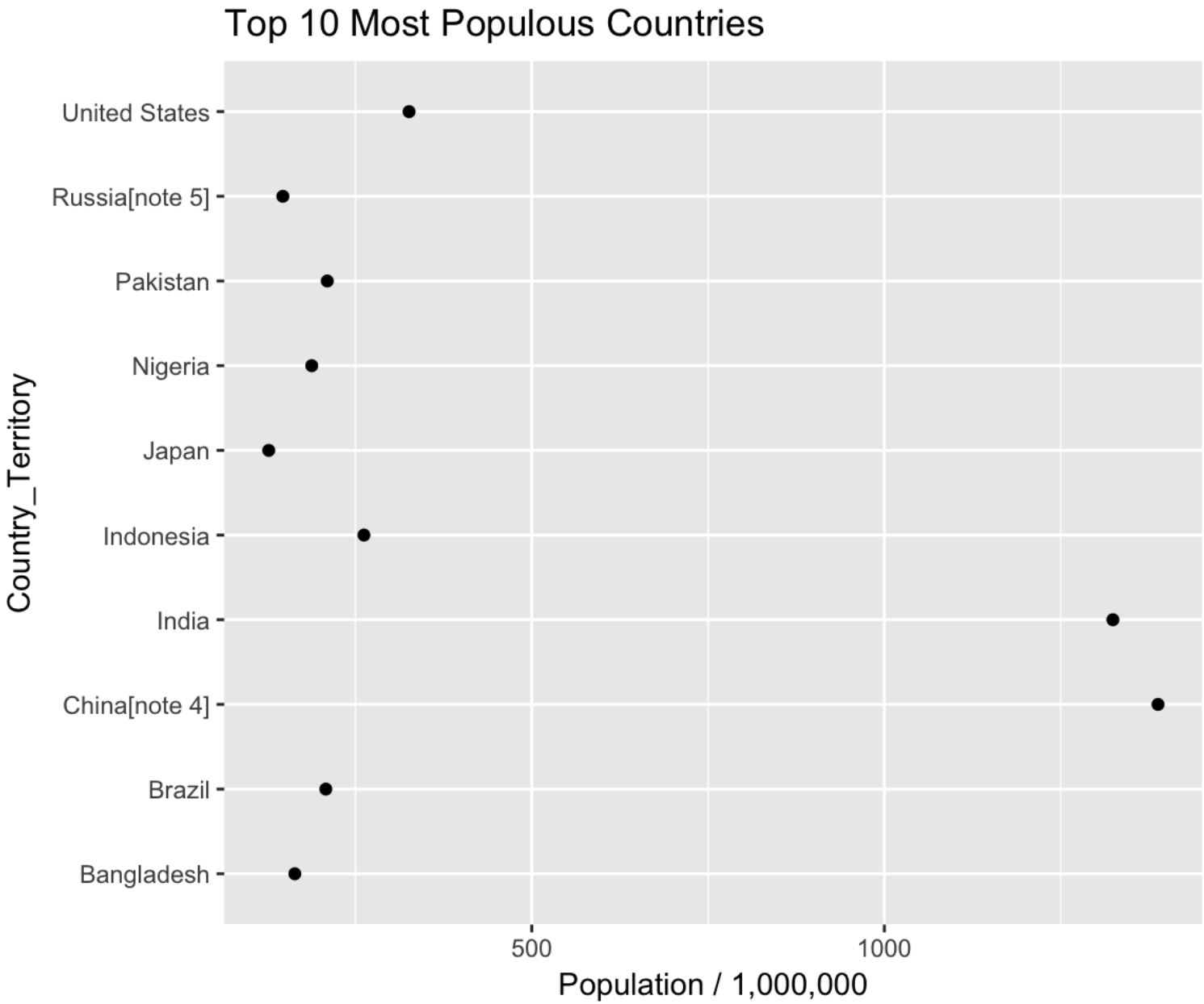
```
(ten_most_df[[1]] %>% select(2:4) -> ten_most_df)
```

```
 Country / Territory    Population                Date
1        China[note 4] 1,387,590,000 November 19, 2017
2                India 1,324,110,000 November 19, 2017
3        United States   326,134,000 November 19, 2017
4            Indonesia   261,600,000  October 31, 2016
5             Pakistan   209,629,000 November 19, 2017
6               Brazil   208,281,000 November 19, 2017
7              Nigeria   188,500,000  October 31, 2016
8           Bangladesh   163,505,000 November 19, 2017
9       Russia[note 5]   146,773,226      June 1, 2017
10              Japan   126,750,000      July 1, 2017
names(ten_most_df)
[1] "Country / Territory" "Population"          "Date"
```

Example

```
names(ten_most_df) <- c("Country_Territory","Population",
                        "Date")
ten_most_df[1:3,]
Country_Territory    Population                Date
1     China[note 4] 1,387,590,000 November 19, 2017
2             India 1,324,110,000 November 19, 2017
3     United States   326,134,000 November 19, 2017

ten_most_df %>% mutate(Population=gsub(",","",Population)) %>%
   mutate(Population=round(as.numeric(Population)/1e+06))  %>%
   ggplot(aes(x=Country_Territory,y=Population)) + geom_point() +
   labs(y = "Population / 1,000,000") + coord_flip() +
   ggtitle("Top 10 Most Populous Countries")
```

Example


Top 10 Most Populous Countries
United States
Russia[note 5]
Pakistan
Nigeria
Japan
Indonesia
India
China[note 4]
Brazil
Bangladesh
Country_Territory
500
1000
Population / 1,000,000

Summary

Let's Summarize

- Need some basic HTML and CSS knowledge to find correct elements
- How to extract text from common elements
- How to extract text from specific elements
- Always have to do some text cleanup of data
- It usually takes multiple times to get it right

`http://bradleyboehmke.github.io/2015/12/scraping-html-text.html`

Summary

HTML Elements look like:

```
<h1>, <h2>, ..., <h6> Heading 1 and so on
<p> Paragraph elements
<ul> Unordered List
<ol> Ordered List
<li> List Element
<div> Division / Section
<table> Tables
<form> Web forms
```

http://bradleyboehmke.github.io/2015/12/scraping-html-text.html

Summary

We can use **rvest** to pick out these elements as a group or individually:

```
library(rvest)

scraping_wiki <- read_html("https://en.wikipedia.org/wiki/Web_scraping")

scraping_wiki %>%   html_nodes("h1")

{xml_nodeset (1)}
[1] <h1 id="firstHeading" class="firstHeading" lang="en">Web scraping</h1>

scraping_wiki %>% html_nodes("h1") %>% html_text()
[1] "Web scraping"
```

http://bradleyboehmke.github.io/2015/12/scraping-html-text.html

Summary

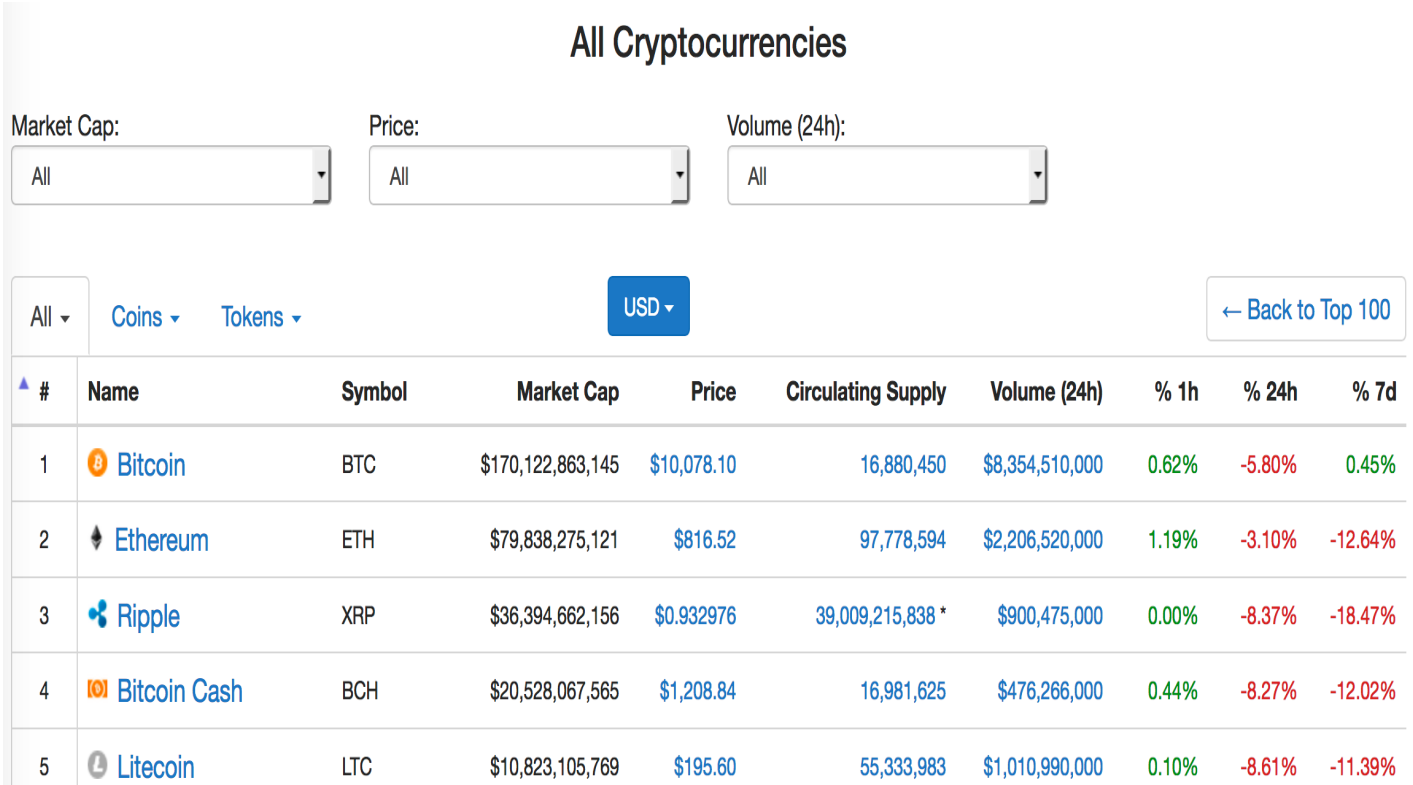
We can grab a set of tags like all <h2> nodes:

```
library(rvest)

scraping_wiki %>%
         html_nodes("h2") %>%
         html_text()

[1] "Contents"
[2] "Techniques[edit]"
[3] "Software[edit]"
[4] "Legal issues[edit]"
[5] "Methods to prevent web scraping[edit]"
[6] "See also[edit]"
[7] "References[edit]"
[8] "Navigation menu"
```

http://bradleyboehmke.github.io/2015/12/scraping-html-text.html

Bitcoin Example

`https://coinmarketcap.com/all/views/all/`



All Cryptocurrencies

Market Cap: All | Price: All | Volume (24h): All

All ▾ | Coins ▾ | Tokens ▾ | USD ▾ | ← Back to Top 100

#	Name	Symbol	Market Cap	Price	Circulating Supply	Volume (24h)	% 1h	% 24h	% 7d
1	Bitcoin	BTC	$170,122,863,145	$10,078.10	16,880,450	$8,354,510,000	0.62%	-5.80%	0.45%
2	Ethereum	ETH	$79,838,275,121	$816.52	97,778,594	$2,206,520,000	1.19%	-3.10%	-12.64%
3	Ripple	XRP	$36,394,662,156	$0.932976	39,009,215,838 *	$900,475,000	0.00%	-8.37%	-18.47%
4	Bitcoin Cash	BCH	$20,528,067,565	$1,208.84	16,981,625	$476,266,000	0.44%	-8.27%	-12.02%
5	Litecoin	LTC	$10,823,105,769	$195.60	55,333,983	$1,010,990,000	0.10%	-8.61%	-11.39%

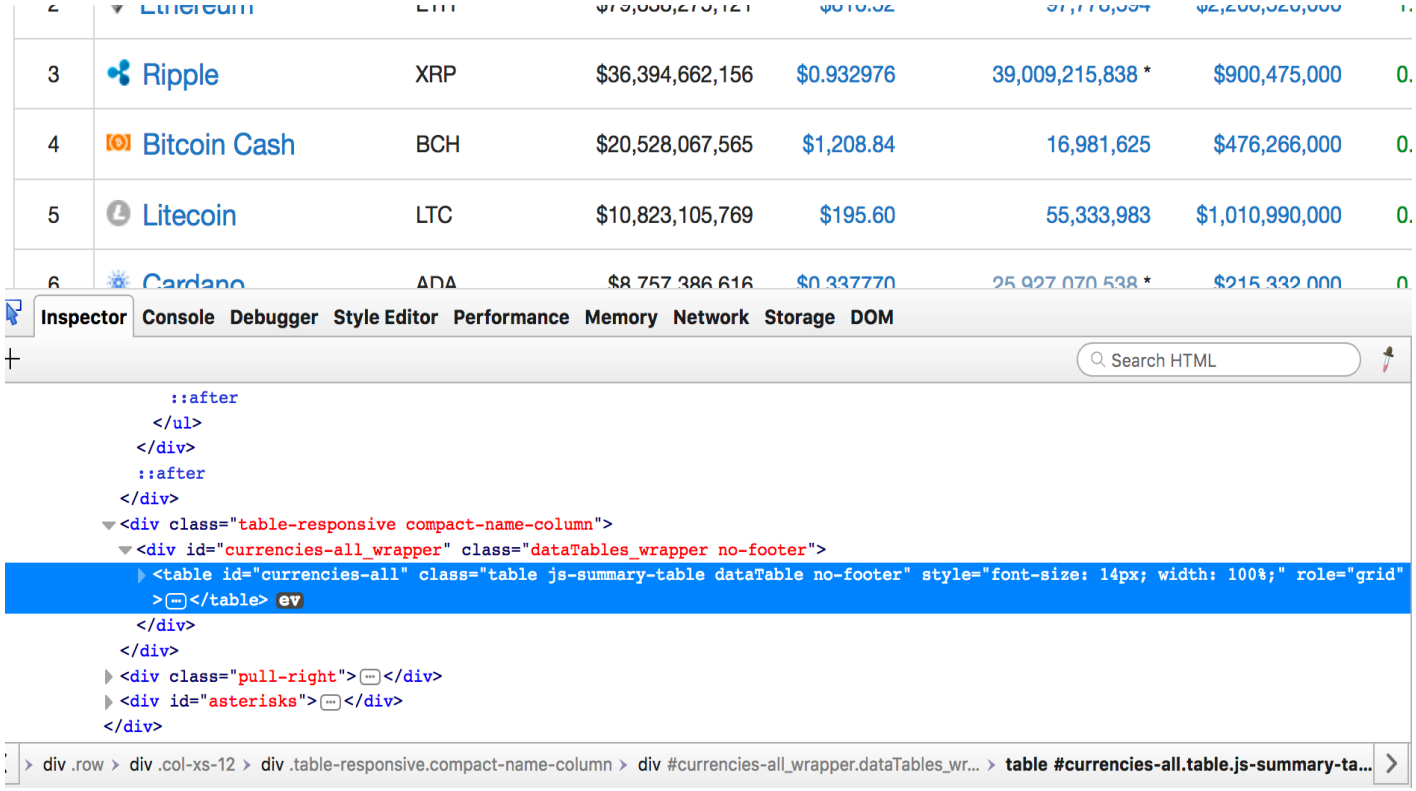
Bitcoin Example

How to Get This Info ?

- Use Selector Gadget
- Use "Inspect Element" in Chrome or Firefox
- In either approach then get the XPATH

Bitcoin Example

https://coinmarketcap.com/all/views/all/



Bitcoin Example

```
library(rvest)
library(dplyr)

url <- "https://coinmarketcap.com/all/views/all/"

bc <- url %>% read_html()

path <- '//*[@id="currencies-all"]'

bc_table <- bc %>% html_nodes(xpath=path) %>% html_table()

# We get back a one element list that is a data frame
str(bc_table,0)

bc_table <- bc_table[[1]]

head(bc_table[,3:5])
  Symbol       Market Cap     Price
1    BTC $170,361,129,795 $10092.20
2    ETH  $79,437,018,614   $812.41
3    XRP  $36,505,409,319 $0.935815
4    BCH  $20,502,161,136  $1207.31
5    LTC  $10,807,757,102   $195.32
6    ADA   $8,788,058,340 $0.338953
```

Bitcoin Example

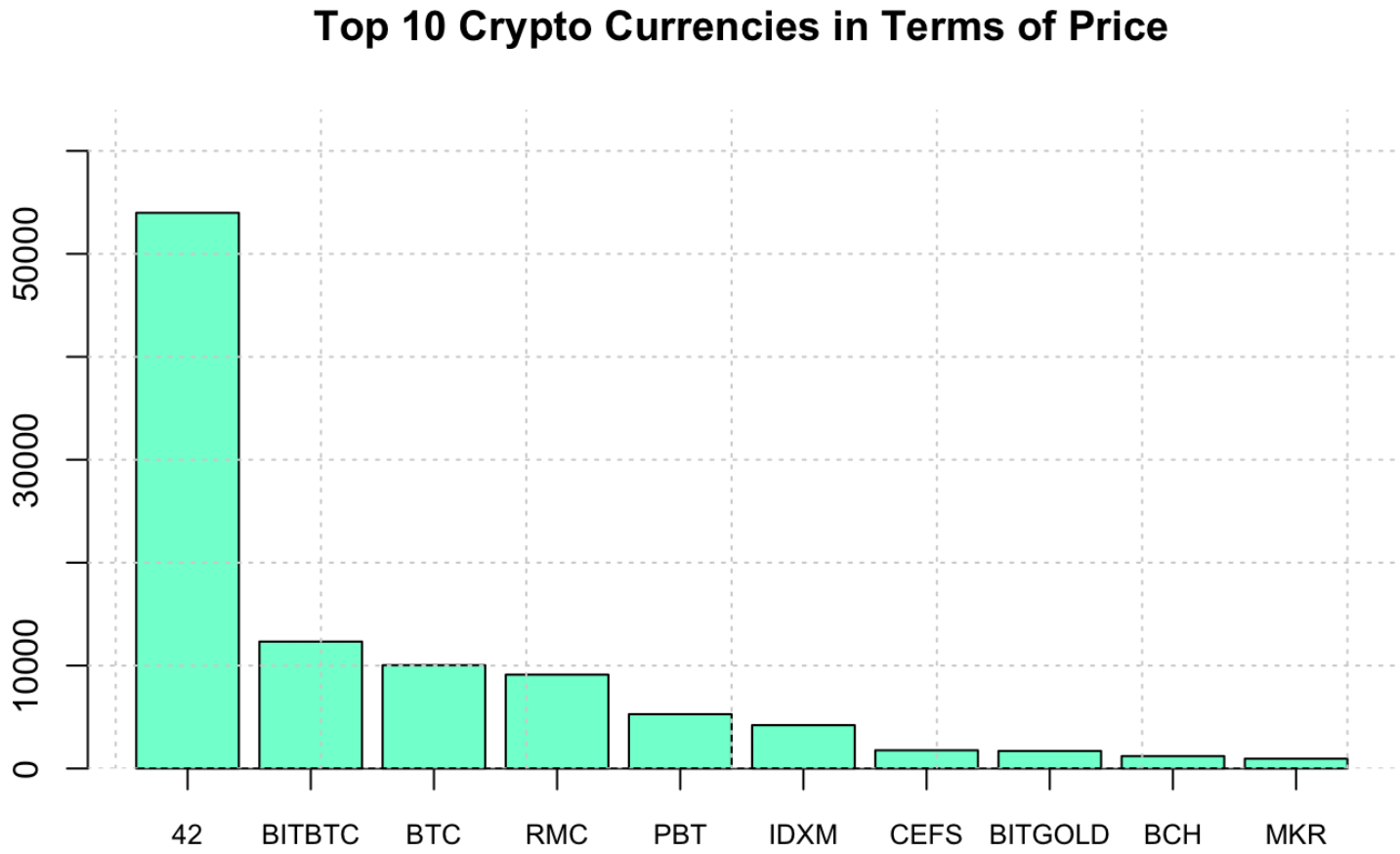
```
# The data is "dirty" and has characers in it that need cleaning

bc_table <- bc_table %>% select(Name,Symbol,Price)
bc_table <- bc_table %>% mutate(Name=gsub("\n"," ",Name))
bc_table <- bc_table %>% mutate(Name=gsub("\\.+","",Name))
bc_table <- bc_table %>% mutate(Price=gsub("\\$","",Price))
bc_table <- bc_table %>% mutate(Price=round(as.numeric(Price),2))

# There are four rows wherein the Price is missing NA

bc_table <- bc_table %>% filter(complete.cases(bc_table))

# Let's get the Crypto currencies with the Top 10 highest prices

top_10 <- bc_table %>% arrange(desc(Price)) %>% head(10)
```

Bitcoin Example

```
# Next we want to make a barplot of the Top 10

ylim=c(0,max(top_10$Price)+10000)
main="Top 10 Crypto Currencies in Terms of Price"
bp <- barplot(top_10$Price,col="aquamarine",
              ylim=ylim,main=main)
axis(1, at=bp, labels=top_10$Symbol,  cex.axis = 0.8)
grid()

# Let's take the log of the price

ylim=c(0,max(log(top_10$Price))+5)
main="Top 10 Crypto Currencies in Terms of log(Price)"
bp <- barplot(log(top_10$Price),col="aquamarine",
              ylim=ylim,main=main)
axis(1, at=bp, labels=top_10$Symbol,  cex.axis = 0.8)
grid()
```

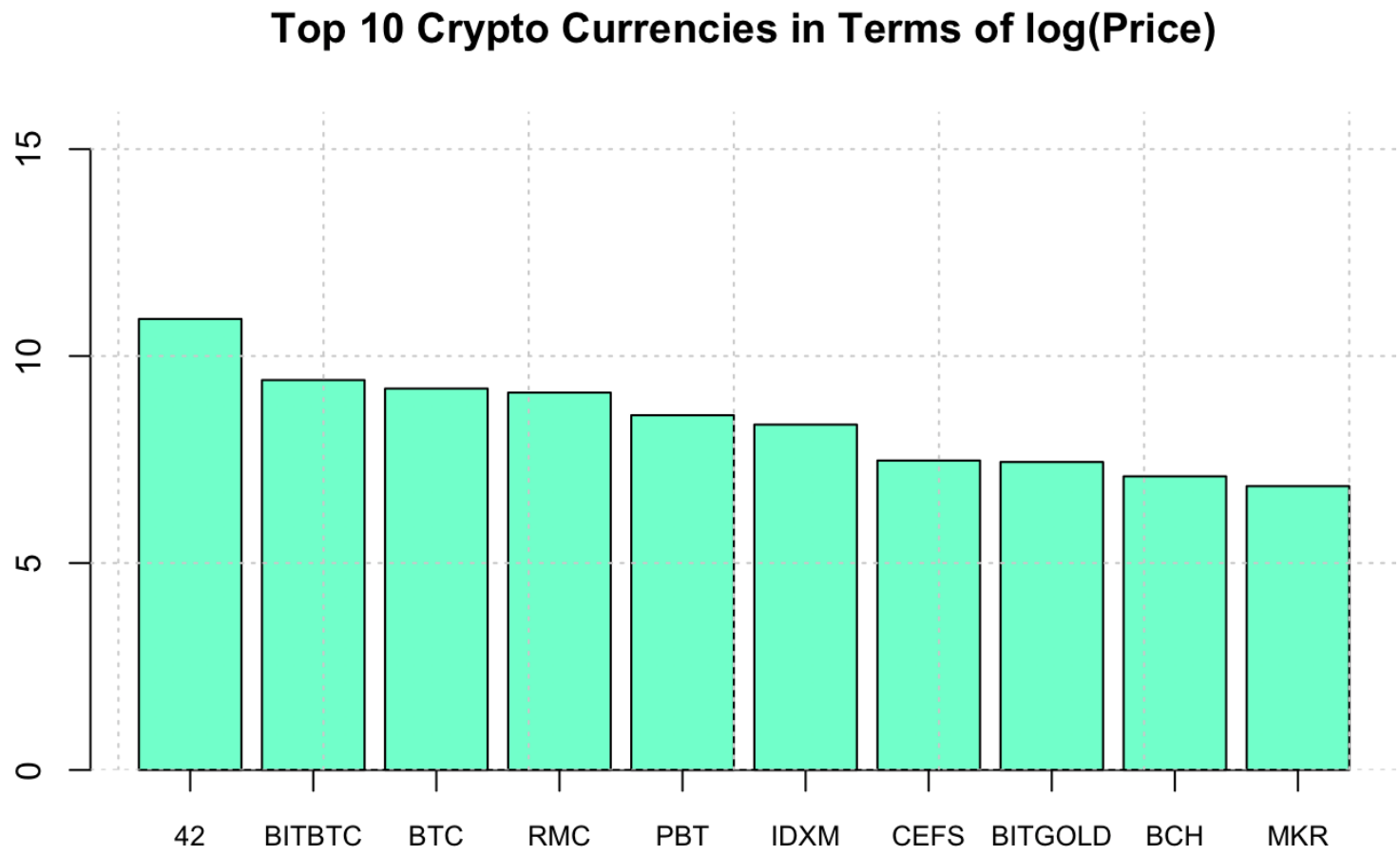
Bitcoin Example

`https://coinmarketcap.com/all/views/all/`



Bitcoin Example

`https://coinmarketcap.com/all/views/all/`



APIs - You Can Do It All Yourself

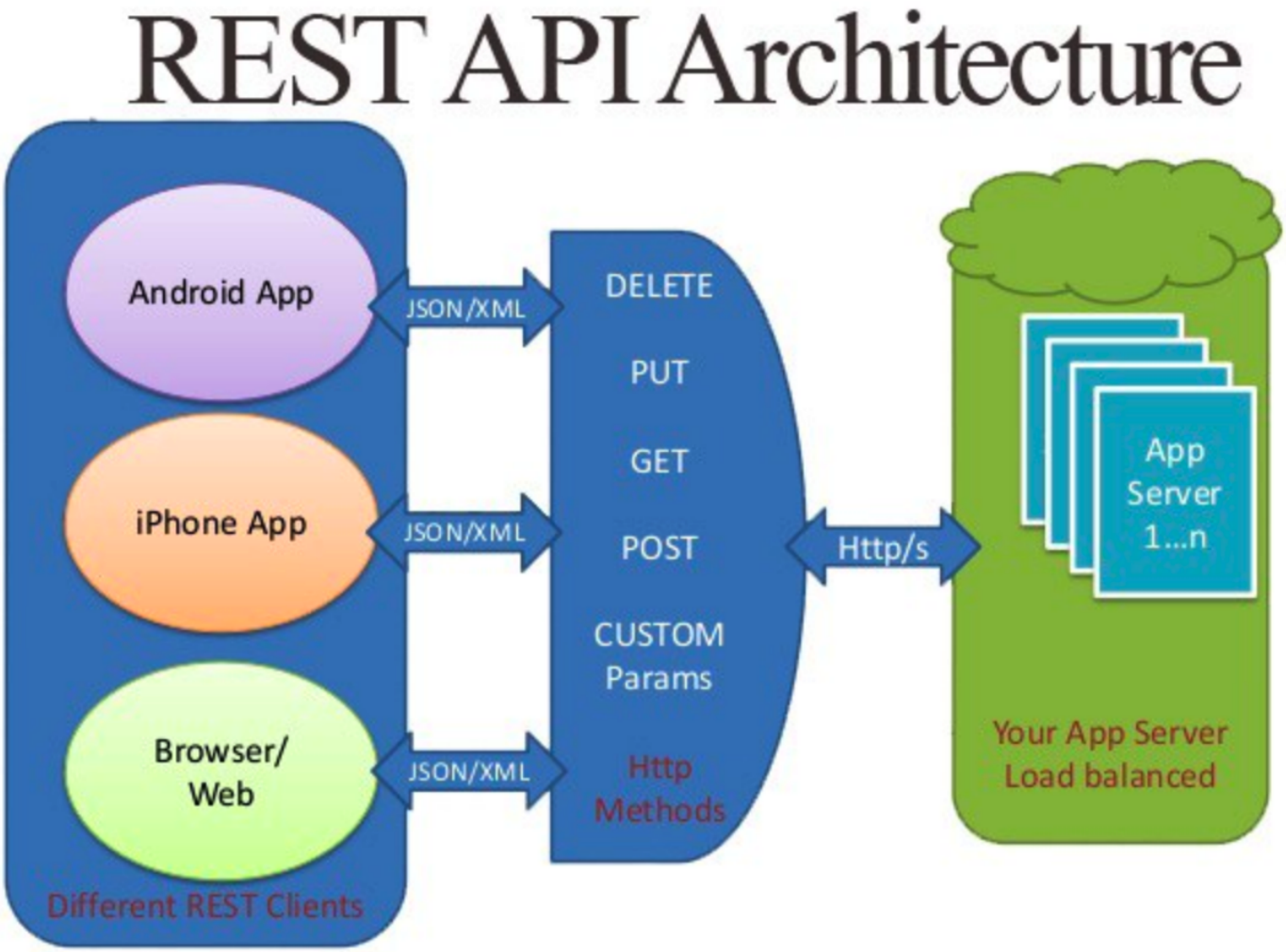
APIs - Motivations

This is all cool but it is tedious. An alternative exists called an API (Application Programming Interface)

- A set of clearly defined methods of communication between software components
- Here were focus on Web APIs to faciliate the access of data on web sites
- Web APIs involved Hypertext Transfer Protocol (HTTP) request messages
- Response is usually Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format
- Many, if not all, of the service you might access support the REST API
- Some R packages do a good job of wrapping in the detrails of the API

`https://en.wikipedia.org/wiki/Representational_state_transfer`

APIs - Motivations



https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php

APIs - Motivations

Identify a Web page or service you would like to access

- Go to CRAN search and type in the name
- If there is an R API then your life will be much easier
- There can be multiple packages for the same service (e.g. twitteR and rtweet)
- Some are good some are just stripped down functions that do the basics
- You don't need a service specific R package to interact with a REST API
- Using rvest, httr, RCurl, XML, and RJSON will simplify access

APIs - Motivations

Check out the OMDb API

OMDb API

The Open Movie Database

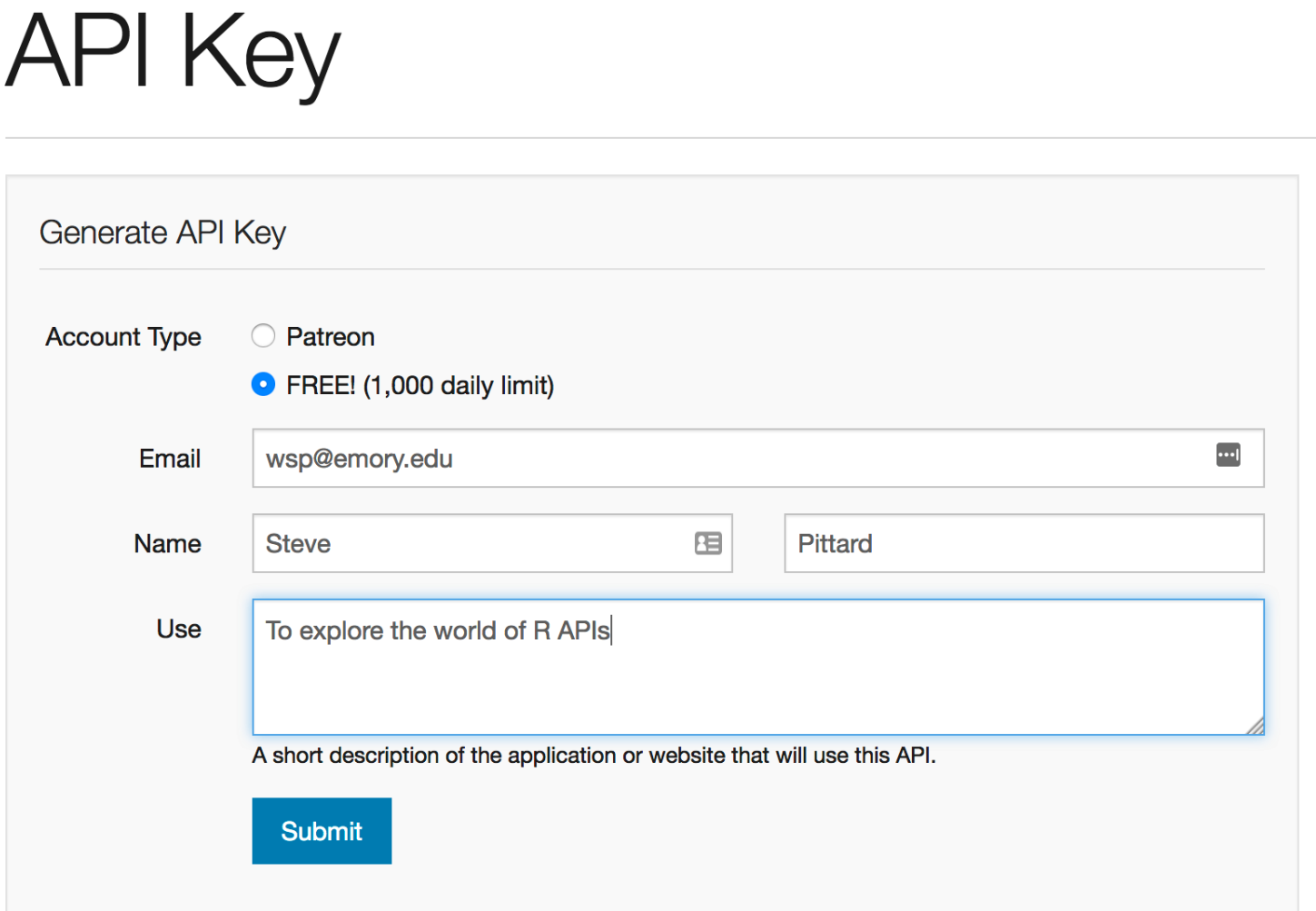
The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

If you find this service useful, please consider making a one-time donation or become a patron.

`https://www.omdbapi.com/`

APIs - Motivations

Sign up for an API key which is free



`https://www.omdbapi.com/`

APIs - Motivations

They tell you how to use they key:

By ID or Title	
i	A valid IMDb ID (e.g. tt1285016)
t	Movie title to search for
type	Type of result to return (movie, series, episode)
y	Year of release
plot	Return short or full plot
r	The data type to return (XML or JSON)
callback	JSONP callback name.
v	API version (reserved for future use)

`http://www.omdbapi.com/?apikey=[yourkey]&t=The+Godfather`

APIs - Motivations

Paste the URL into any web browser. You must supply your key for this to work. What you get back is a JSON formatted entry corresponding to "The GodFather" movie.

`http://www.omdbapi.com/?apikey=[yourkey]&t=The+Godfather`


```
Title:      "The Godfather"
Year:       "1972"
Rated:      "R"
Released:   "24 Mar 1972"
Runtime:    "175 min"
Genre:      "Crime, Drama"
Director:   "Francis Ford Coppola"
Writer:     "Mario Puzo (screenplay), Francis Ford Coppola (screenplay), Mario Puzo (novel)"
Actors:     "Marlon Brando, Al Pacino, James Caan, Richard S. Castellano"
Plot:       "The aging patriarch of an organized crime dynasty transfers control of his clandestine empire
Language:   "English, Italian, Latin"
Country:    "USA"
Awards:     "Won 3 Oscars. Another 24 wins & 27 nominations."
Poster:     "https://images-na.ssl-images-amazon.com/images/M/MV5BY2Q2NzQ3ZDUtNWU5OC00Yjc0LThlYmEtNWM3NTFr
Ratings:
  0:
    Source:   "Internet Movie Database"
    Value:    "9.2/10"
```

APIs - Motivations

You can use the RJSONIO package to parse the returned JSON document.

```
movie <- fromJSON("http://www.omdbapi.com/?apikey=f7c004c&t=The+Godfather")

str(movie)
List of 25
 $ Title     : chr "The Godfather"
 $ Year      : chr "1972"
 $ Rated     : chr "R"
 $ Released  : chr "24 Mar 1972"
 $ Runtime   : chr "175 min"
 $ Genre     : chr "Crime, Drama"
 $ Director  : chr "Francis Ford Coppola"
 $ Writer    : chr "Mario Puzo (screenplay), Francis Ford Coppola (screenplay), Mario Puzo (novel)"
 $ Actors    : chr "Marlon Brando, Al Pacino, James Caan, Richard S. Castellano"
 $ Plot      : chr "The aging patriarch of an organized crime dynasty transfers control of his clandestine empi
 $ Language  : chr "English, Italian, Latin"
 $ Country   : chr "USA"
 $ Awards    : chr "Won 3 Oscars. Another 24 wins & 27 nominations."
 $ Poster    : chr "https://images-na.ssl-images-amazon.com/images/M/MV5BY2Q2NzQ3ZDUtNWU5OC00YjcOLThlYmEtNWM3NT
 $ Ratings   :List of 3
  ..$ : Named chr [1:2] "Internet Movie Database" "9.2/10"
  .. ..- attr(*, "names")= chr [1:2] "Source" "Value"
  ..$ : Named chr [1:2] "Rotten Tomatoes" "99%"
$ Metascore : chr "100"
 $ imdbRating: chr "9.2"
 $ imdbVotes : chr "1,292,187"
 $ imdbID    : chr "tt0068646"
 $ Type      : chr "movie"
 $ DVD       : chr "09 Oct 2001"
 $ BoxOffice : chr "N/A"
 $ Production: chr "Paramount Pictures"
 $ Website   : chr "http://www.thegodfather.com"
 $ Response  : chr "True"
```

APIs - Motivations

```
movie$Plot
[1] "The aging patriarch of an organized crime dynasty transfers control of his cla
     empire to his reluctant son."

# Get the ratings from the three services

sapply(movie$Ratings,unlist)
       [,1]                       [,2]              [,3]
Source "Internet Movie Database" "Rotten Tomatoes" "Metacritic"
Value  "9.2/10"                  "99%"             "100/100"
```

APIs - Motivations

Let's Get all the Episodes for Season 1 of Game of Thrones

```
url <- "http://www.omdbapi.com/?apikey=f7c004c&t=Game%20of%20Thrones&Season=1"
movie <- fromJSON(url)
str(movie,1)

episodes <- data.frame(do.call(rbind,movie$Episodes),stringsAsFactors = FALSE)
episodes
```

	Title	Released	Episode	imdbRating	imdbID
1	Winter Is Coming	2011-04-17	1	9.0	tt1480055
2	The Kingsroad	2011-04-24	2	8.8	tt1668746
3	Lord Snow	2011-05-01	3	8.7	tt1829962
4	Cripples, Bastards, and Broken Things	2011-05-08	4	8.8	tt1829963
5	The Wolf and the Lion	2011-05-15	5	9.1	tt1829964
6	A Golden Crown	2011-05-22	6	9.2	tt1837862
7	You Win or You Die	2011-05-29	7	9.3	tt1837863
8	The Pointy End	2011-06-05	8	9.1	tt1837864
9	Baelor	2011-06-12	9	9.6	tt1851398
10	Fire and Blood	2011-06-19	10	9.5	tt1851397

APIs - Motivations

Wait a minute. Looks like someone created an R package that wraps all this for us. It is called **omdbapi**

omdbapi is an R package wrapper for the Open Movie Database API

NOTE: THE OMDB API NOW REQUIRES AN API KEY https://www.patreon.com/posts/api-is-going-10743518

The following functions are implemented:

- `find_by_id` : Retrieve OMDB info by IMDB ID search
- `find_by_title` : Retrieve OMDB info by title search
- `get_actors` : Get actors from an omdb object as a vector
- `get_countries` : Get countries from an omdb object as a vector
- `get_directors` : Get directors from an omdb object as a vector
- `get_genres` : Get genres from an omdb object as a vector
- `get_writers` : Get writers from an omdb object as a vector
- `search_by_title` : Lightweight OMDB title search

```
# Use devtools to install

devtools::install_github("hrbrmstr/omdbapi")
```

APIs - Motivations

Let's use it:

```
library(omdbapi)

# The first time you use this you will be prompted to enter your
# API key

search_by_title("Star Wars", page = 2)
Couldn't find env var OMDB_API_KEY See ?omdb_api_key for more details.
Please enter your API key and press enter:
: <enter your key here>

Updating OMDB_API_KEY env var to PAT
                                    Title      Year    imdbID   Type
1                Star Wars: The Clone Wars 2008-2015 tt0458290 series
2                    Star Wars: Clone Wars 2003-2005 tt0361243 series
3                        Star Wars: Rebels      2014 tt2930604 series
4            The Star Wars Holiday Special      1978 tt0193524  movie
5                  Robot Chicken: Star Wars      2007 tt1020990  movie
6  Star Wars: Knights of the Old Republic      2003 tt0356070   game
7           Star Wars: The Force Unleashed      2008 tt1024923   game
8                Star Wars: Battlefront II      2005 tt0483168   game
9      Robot Chicken: Star Wars Episode II      2008 tt1334272  movie
10   Robot Chicken: Star Wars Episode III      2010 tt1691338  movie
```

APIs - Motivations

Let's use it:

```
library(omdbapi)

(gf <- find_by_title("The GodFather"))
# A tibble: 3 x 25
          Title  Year Rated   Released Runtime        Genre              Director
          <chr> <chr> <chr>     <date>   <chr>        <chr>                 <chr>
1 The Godfather  1972     R 1972-03-24 175 min Crime, Drama Francis Ford Coppola
2 The Godfather  1972     R 1972-03-24 175 min Crime, Drama Francis Ford Coppola
3 The Godfather  1972     R 1972-03-24 175 min Crime, Drama Francis Ford Coppola
# ... with 18 more variables: Writer <chr>, Actors <chr>, Plot <chr>, Language <chr
#   Poster <chr>, Ratings <list>, Metascore <chr>, imdbRating <dbl>, imdbVotes <dbl
#   DVD <date>, BoxOffice <chr>, Production <chr>, Website <chr>, Response <chr>

get_actors((gf))
[1] "Marlon Brando"   "Al Pacino"   "James Caan"    "Richard S. Castellano"
```

APIs - Summary

- Many websites offer an API
- See `https://www.programmableweb.com/` for a big list of them
- Some cost money whereas some are free
- Most work on the idea of key-based access using a REST approach
- They return either XML/JSON (usually as an option)
- R has the XML, xml2, and RJSONIO packages to parse the returned documents
- Sometimes there are R packagew written on top of the APIs
- These insulate the user from parsing JSON/XML and turning things into data frames

Bag of Words - Motivations

What is it ?

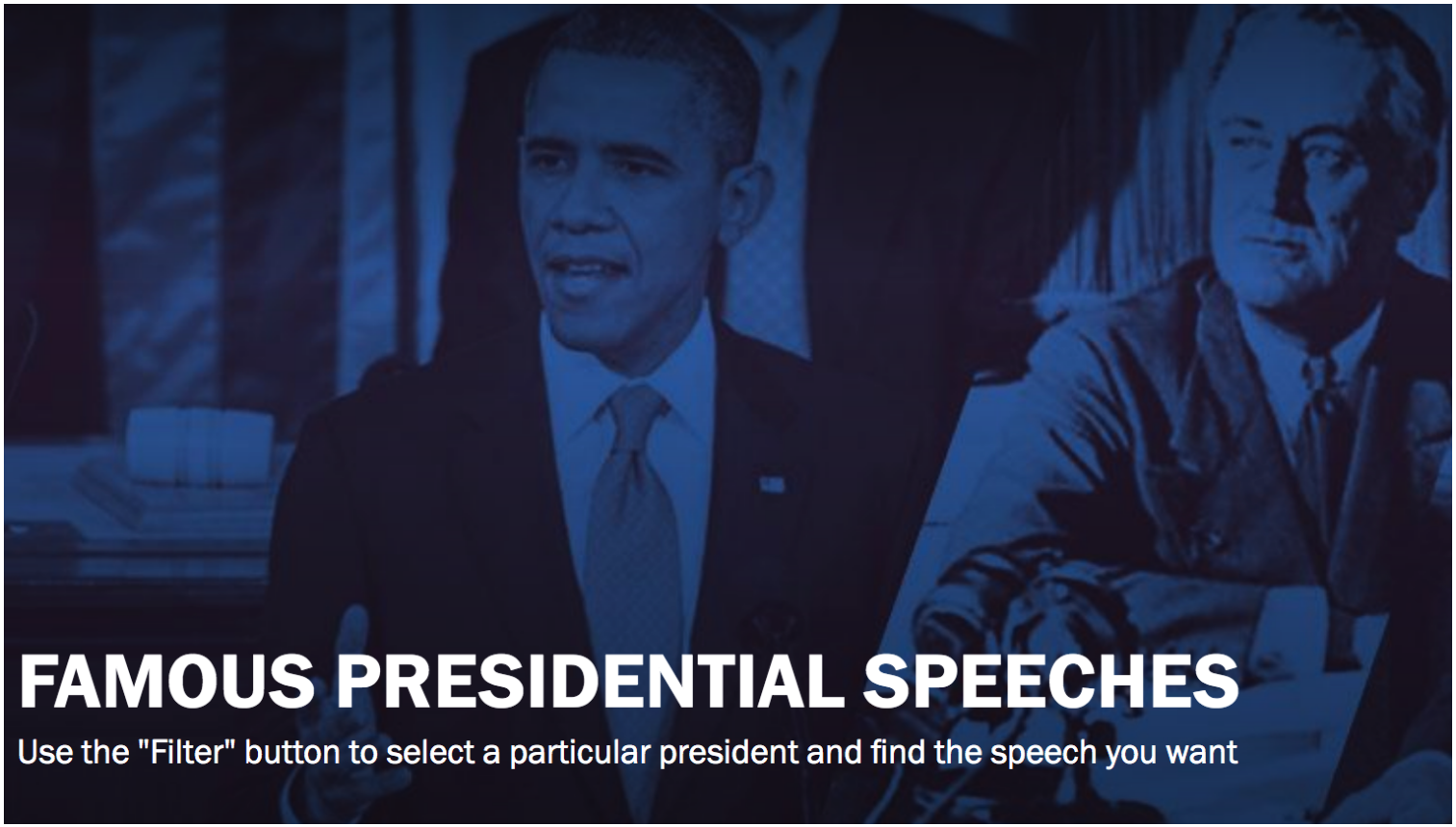
- Extracting meaning from collections of text (without reading !)
- Detection and analysis of patterns in unstructured textual collections
- Use Natural Language Processing techniques to reach conclusions
- Discover what ideas occur in text and how they might be linked
- Can the discovered patterns be used to infer or predict behavior ?
- Identify interesting ideas that might otherwise be ignored

Motivations

- Bag of Words (BoW) is a way of getting info from unstructured text
- Describes the occurrence of words in a document
- Concerned only with if certain words occur in a document but not where in the document
- Considers that common words, (e.g. "the","a","an") don't add significant meaning
- For analyzing book/article text, tweets, SMS text messages, etc
- Treats words as elements in a vector
- Word frequencies and counts help
- Can assess the "sentiment" of a body of text
- R has libraries to help

Un Autre Exemple

Presidential Speeches



`https://millercenter.org/the-presidency/presidential-speeches`

Un Autre Exemple

Find the URL for Lincoln's March 4, 1865 Speech:

```
url <- "https://millercenter.org/the-presidency/presidential-speeches/march-4-1865-second-inaugural-address"

library(rvest)
lincoln_doc <- read_html(url) %>%
                    html_nodes(".view-transcript") %>%
                    html_text()

lincoln_doc

[1] "TranscriptFellow-Countrymen:  At this second appearing to
take the oath of the Presidential office there is less occasion
for an extended address than there was at the first. Then a statement
somewhat in detail of a course to be pursued seemed fitting and
proper. Now, at the expiration of four years, during which public
decla..
```

Un Autre Exemple

Find the URL for Lincoln's March 4, 1865 Speech:

```
lincoln_doc <- read_html(url) %>%
                    html_nodes(".view-transcript") %>%
                    html_text()

word_vec <- unlist(strsplit(lincoln_doc," "))
word_vec[1:20]
 [1] "TranscriptFellow-Countrymen:" ""
 [3] "At"                           "this"
 [5] "second"                       "appearing"
 [7] "to"                           "take"
 [9] "the"                          "oath"
[11] "of"                           "the"
[13] "Presidential"                 "office"
[15] "there"                        "is"
[17] "less"                         "occasion"
[19] "for"                          "an"
```

Un Autre Exemple

Find the URL for Lincoln's March 4, 1865 Speech:

```
sort(table(word_vec),decreasing = TRUE)[1:10]
the    to   and    of that   for    be    in    it     a
 54    26    24    22   11     9     8     8     8     7
```

This is okay but many of these words don't add much to understanding the document. These are called "stop words" and we could remove them.

Un Autre Exemple

```
# Remove all punctuation marks
word_vec <- gsub("[[:punct:]]","",word_vec)
stop_words <- c("the","to","and","of","the","for","in","it",
                "a","this","which","by","is","an","hqs","from",
                "that","with","as")

for (ii in 1:length(stop_words)) {
    for (jj in 1:length(word_vec)) {
      if (stop_words[ii] == word_vec[jj]) {
          word_vec[jj] <- ""
      }
  }
}

word_vec <- word_vec[word_vec != ""]
sort(table(word_vec),decreasing = TRUE)[1:10]
word_vec
  war   all    be    we   but   God shall   was    do   let
   11     8     8     6     5     5     5     5     4     4
```

Un Autre Exemple

Find the URL for Lincoln's March 4, 1865 Speech:

```
url <- "https://millercenter.org/the-presidency/presidential-speeches/march-4-1865-second-inaugural-address"
```

```
library(rvest)
lincoln_doc <- read_html(url) %>%
                    html_nodes(".view-transcript") %>%
                    html_text()
```

```
(frequent_terms <- freq_terms(lincoln_doc,10))
   WORD  FREQ
1   the    58
2   to     27
3   and    24
4   of     22
5   it     13
6   that   12
7   war    11
8   all    10
9   for     9
10  in      9
11  which   9
```

Stop Words

Not very helpful because there are lots of "filler" words. These are also known as "stop words".

```
library(qdap)
# There is a function that can remove the sotp words for us
cleaned_up_terms <- rm_stopwords(lincoln_doc,stopwords=Top100Words)

freq_terms(cleaned_up_terms,10)
   WORD     FREQ
1  war        12
2  god         5
3  shall       5
4  both        4
5  let         4
6  union       4
7  years       4
8  interest    3
9  must        3
10 neither     3
11 offenses    3
12 those       3
13 us          3
14 woe         3
```

WorkFlow - If I Can Just Remember The Steps

Workflow

Consider the following workflow:

- Identify and Obtain text (e.g. websites, Twitter, Databases, PDFs, surveys)
- Create a text "Corpus" - a structure that contains the raw text
- Apply transformations:
  - Normalize case (convert to lower case)
  - Remove puncutation and stopwords
  - Remove domain specific stopwords
- Create a Term Document Matrix (or Document Term Matrix)
- Perform Analysis and Visualizations (word frequency, tagging, wordclouds)
- Then Do Sentiment Analysis

Workflow

Consider the following workflow (slightly altered)

- Identify and Obtain text (e.g. websites, Twitter, Databases, PDFs, surveys)
- Create a text "Corpus" - a structure that contains the raw text
- Create a Term Document Matrix (or Document Term Matrix)
- Apply transformations:
  - Normalize case (convert to lower case)
  - Remove puncutation and stopwords
  - Remove domain specific stopwords
- Perform Analysis and Visualizations (word frequency, tagging, wordclouds)
- Then Do Sentiment Analysis

Workflow

R has Packages to Help. These are just some of them:

- QDAP - Quantitative Discourse Package
- tm - text mining applications within R
- tidytext - Text Mining using ddplyr and ggplot and tidyverse tools
- SentimentAnalysis - For Sentiment Analysis

However, consider that:

- Some of these are easier to use than others
- Some can be kind of a problem to install (e.g. qdap)
- They all offer similar capabilities
- We'll look at qdap and tidytext

Packages

QDAP:

- Requires Java version that matches your R version (64 bit)
- This is usually a separate download
- `https://www.journaldev.com/476/java-windows-10-download-install`
- Then install `qdap` just as you would any other R package
- Then load the library using the typical `library(qdap)`

Packages

tm:

- The main structure is a "Corpus"
- Represents a collection of text documents
- `tm` implements a `VCorpus`
- A `VCoprus` is a "Volatile Corpus" in memory (RAM)
- For very large collections use `PCorpus` for on disk storage
- A Corpus can then be turned into a TDM or DTM
- Lots of functions are then available for analyzing terms

Cleaning Words

Language is imperfect and especially so with written stuff:

- Convert to lower case
- Remove whitespace, punctuation, numbers
- If tweets then remove URLs, hastags, user names, Retweets
- Do "stemming" (reduce "catlike" and "catty" to "cat")
- Remove "stop words" (words of little value)
- The `tm` package has a built in stop words function

```
stopwords()[1:20]
 [1] "i"          "me"         "my"         "myself"
 [5] "we"         "our"        "ours"       "ourselves"
 [9] "you"        "your"       "yours"      "yourself"
[13] "yourselves" "he"         "him"        "his"
[17] "himself"    "she"        "her"        "hers"
```

Cleaning Words

Cleanup is ususally done right after reading in a Corpus.

- But it could be done before by using common R commands (e.g. gsub) or `stringr`
- Some people use a combination of approaches and packages (can be confusing)
- The `tm` package has an all "purpose cleaner"
- You can add in your own private "stop words"
- `tm` isn't perfect but it works well enough
- Remember: If you don't remove stop words then things like WordClouds become cluttered with meaningless words.

Cleaning Words

Here is a function that contains most of the important cleaning functions including stop word removal:

```
tm_cleaner <- function(corpus, stop=stopwords("en"), rm_num=TRUE) {
  require(tm)
  corpus <- tm_map(corpus, stripWhitespace)
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removeWords, stop)
   corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, content_transformer(function(x) gsub("http\\w+", "", x)))
  return(corpus)
}

st <- "They do such a great job preserving our freedom.  It
was Malia's birthday, on the Fourth of July, and she's now 14."

clean_st <- tm_cleaner(VCorpus(VectorSource(st)))
clean_st[[1]][1]
$content
[1] "    great job preserving  freedom   malias birthday   fourth  july   now "
```

Cleaning Words

You could add in your own stop words:

```
tm_cleaner <- function(corpus, stop=stopwords("en"), rm_num=TRUE) {
 ..
 ..
  return(corpus)
}

st <- "They do such a great job preserving our freedom.  It
was Malia's birthday, on the Fourth of July, and she's now 14."

mystop <- c("now",stopwords("en"))

clean_st <- tm_cleaner(VCorpus(VectorSource(st)),stop=mystop)

clean_st[[1]][1]
$content
[1] "    great job preserving  freedom   malias birthday   fourth  july
```

Create a Corpus and Clean It

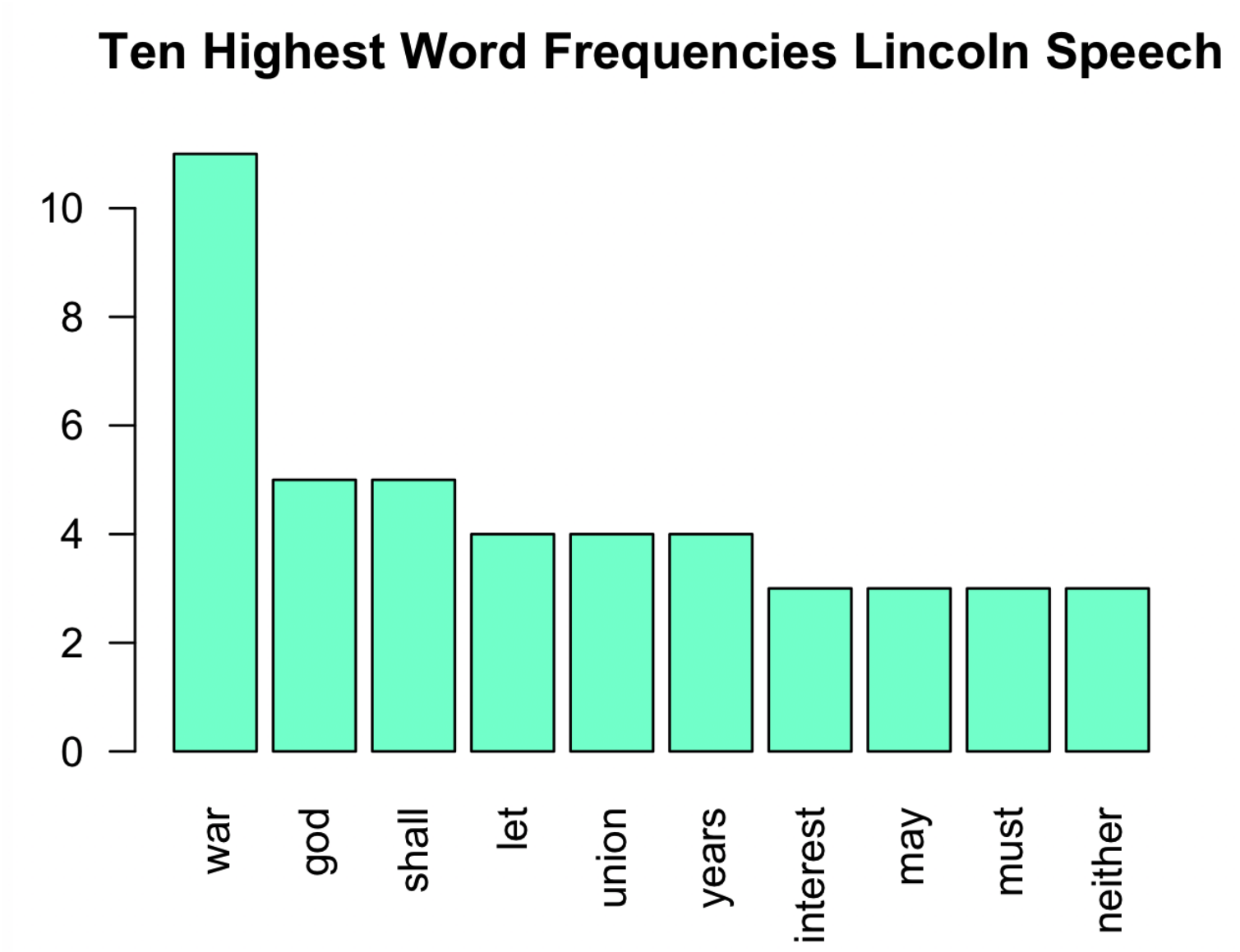
Let's go back to the `lincoln_doc`

```
library(tm)  # The tm package let's us create a Corpus

lincoln_vec_source <- VectorSource(lincoln_doc)
lincoln_vcorpus <- VCorpus(lincoln_vec_source)

str(lincoln_vcorpus)

tm_cleaner <- function(corpus, stop=stopwords("en"), rm_num=TRUE) {
  require(tm)
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, stripWhitespace)
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removeWords, stop)
  corpus <- tm_map(corpus, content_transformer(function(x) gsub("http\\w+", "", x))
  return(corpus)
}

cleaned_lincoln_corp <- tm_cleaner(lincoln_vcorpus)
```

Create a Term Document Matrix

```
# Next create a tdm
lincoln_tdm <- TermDocumentMatrix(cleaned_lincoln_corp)
lincoln_tdm_m <- as.matrix(lincoln_tdm)

# Get the term frequencies and sort them
terms_freq <- rowSums(lincoln_tdm_m)
terms_freq_sorted <- sort(terms_freq, decreasing=TRUE)
terms_freq_sorted[1:10]

barplot(terms_freq_sorted[1:10],col="aquamarine",las=2,
        main="Ten Highest Word Frequencies Lincoln Speech")
```

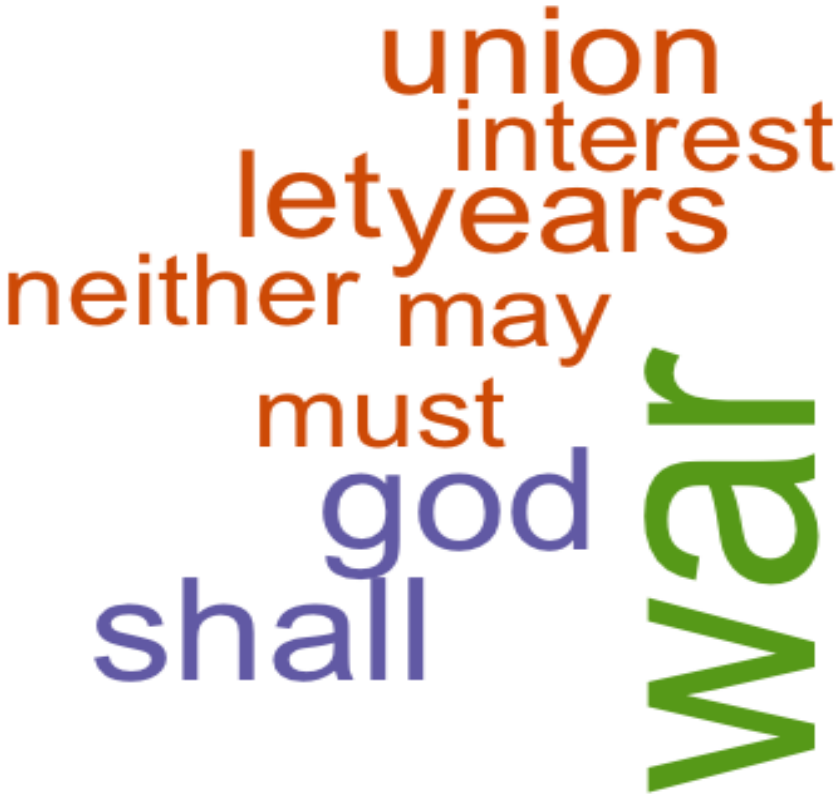
Get Frequent Terms and Make a Wordcloud



Get Frequent Terms and Make a Wordcloud

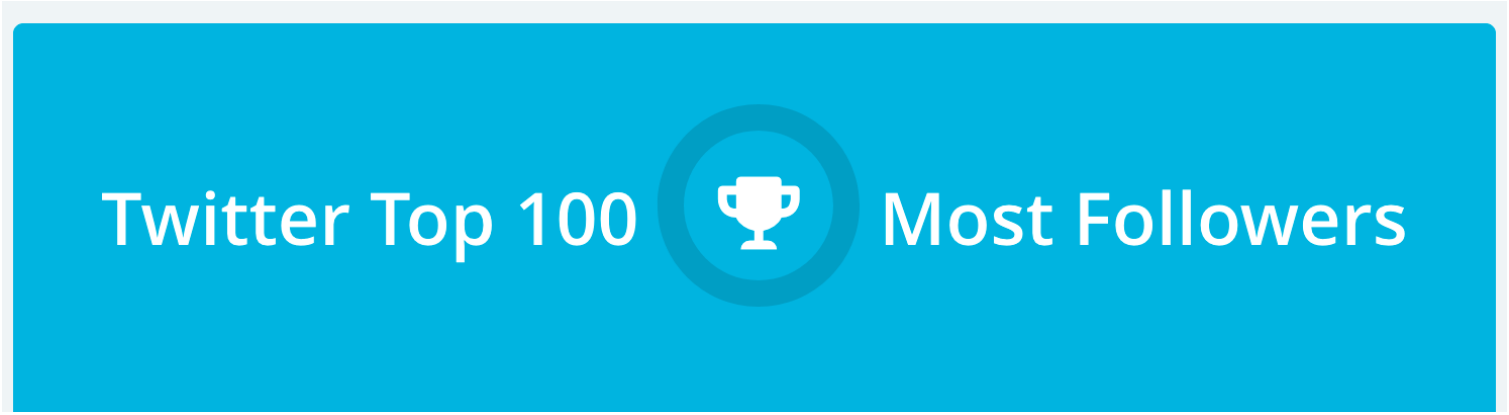
```
# Create a Wordcloud with Colors

library(wordcloud)
library(RColorBrewer)

df <- data.frame(word=names(terms_freq_sorted),
                 freq=terms_freq_sorted,
                 stringsAsFactors = FALSE)

color_palette <- brewer.pal(5,"Dark2")
wordcloud(df$word,df$freq,max.words = 10,colors = color_palette)
```

Get Frequent Terms and Make a Wordcloud



Finally - The Perfect Instagram Shot !

Social Media - Big Business



Social Media - Big Business

DIGITAL

Selena Gomez's Social Media Posts Are Evidently Worth $550,000 Apiece

Women rule Facebook, Twitter and Instagram

Twitter

- Tweets come from many sources - people, politicians, media outlets, sports teams, movie stars, etc
- Multimedia content can be sent along with a Tweet (pics, movies, emojis)
- Real time communication
- Anyone can create an account and also sign up to use the API
- Great way to see trends in a time window (1 hour, 1 day, 1 month, 3 monts,etc)

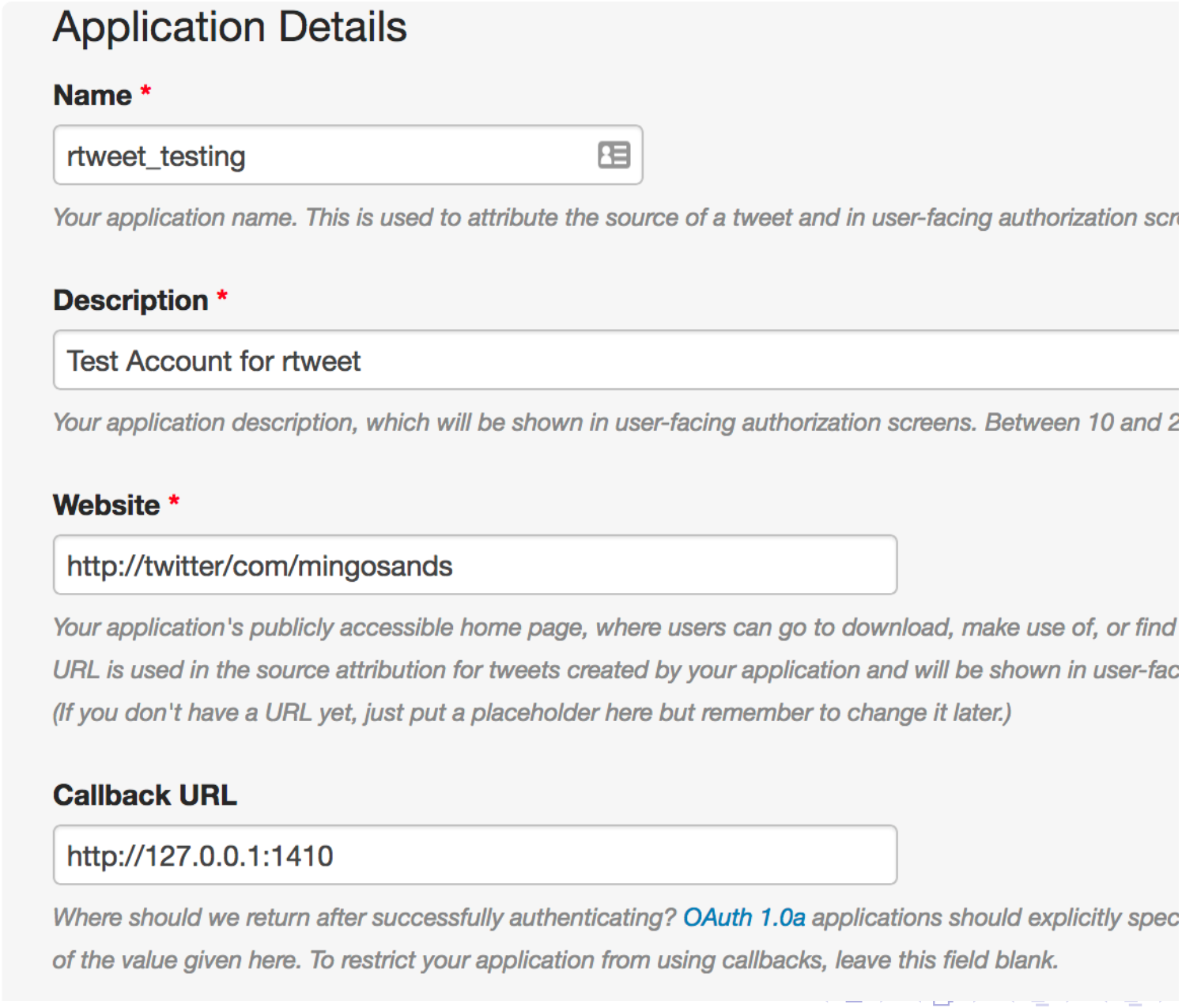
Twitter Anatomy

- User Name: A unique userid
- Time Stamp: When the tweet went out
- Text: The content of the tweet
- Hashtags: Words prefixed with a # character. Describes a specific group or topic (existing or not). Some hashtags are well known, others are not.
- Links: Hyper links to other web sourcved
- Replies: Replies to a posted tweet
- Retweets: When someone shares a third party tweet with followers
- Favorites: A history of tweets you have liked
- Latitude / Longitude: Some tweets have geo coding information

Getting Started

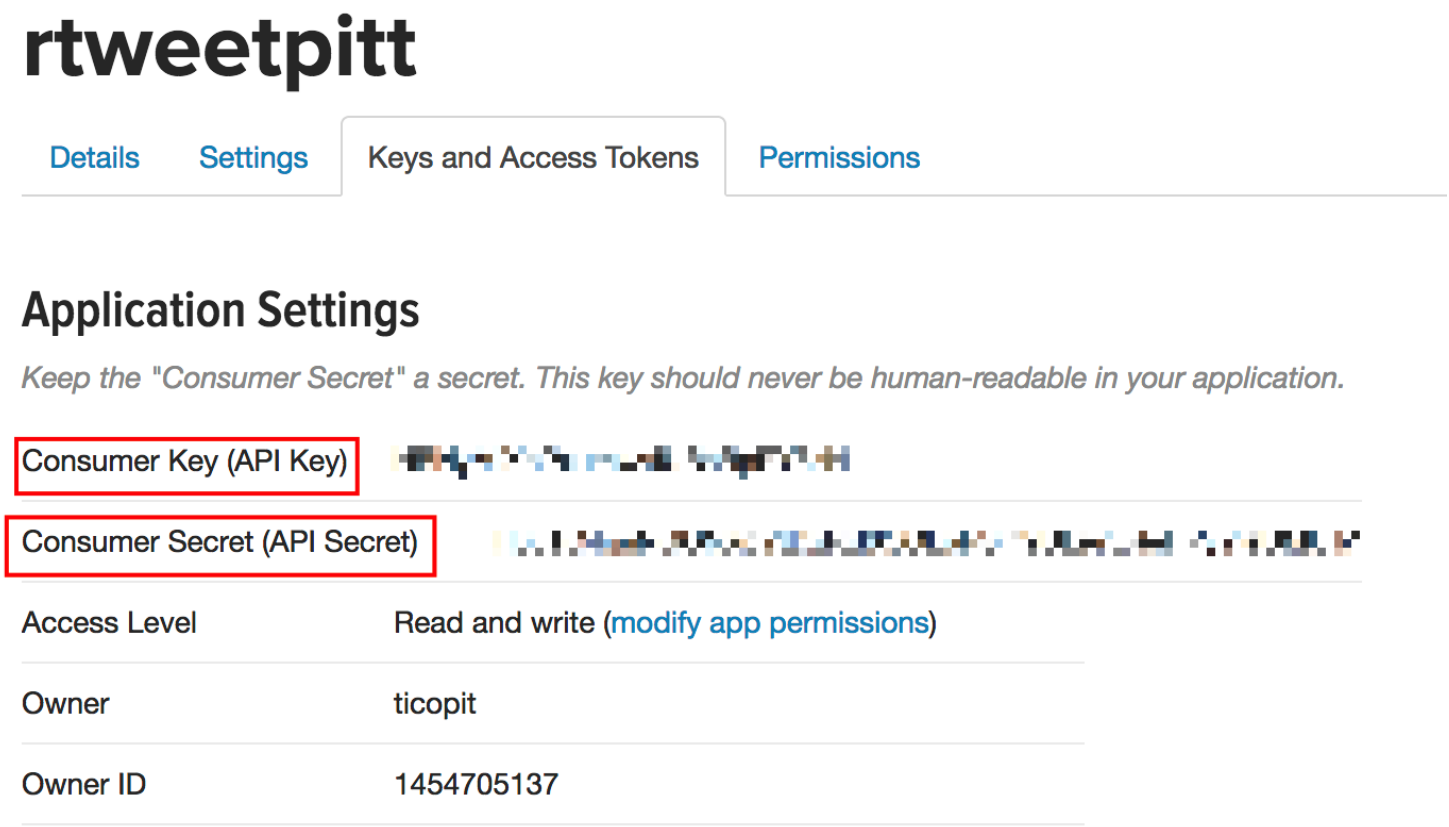
- Install the `rtweet` package
- Setup a Twitter account. It's free at `https://twitter.com/`
- Note that it requires your cell number to create an App
- (But you can delete the account after the class is over)
- After you have created your account then go `http://apps.twitter.com`
- From with your account create an "application" so you can mine other tweets
  - Name: rtweet_testing
  - Description: Testing Account for rtweet
  - Website: `http://twitter.com/userid` (replace userid with your userid)
  - Callback URL: `http://127.0.0.1:1410`
  - Create App

Getting Started



Getting Started

Then go to the App home page and click `Keys and Access Tokens`



My info is pixelated but yours won't be. Make a copy of the two keys

Getting Started

```
appname <- "rtweetpit"
key <- "your_consumer_key_here"
secret <- "your_secret_consumer_key_here"

# Get authenticated

library(rtweet)

twitter_token <- create_token(
  app = "appname"rtweetpit,
  consumer_key = key,
  consumer_secret = secret)

# Running this code will contact Twitter and load a web page.
# If authentication is successful then you will see a web page
# with the contents of

Authentication complete. Please close this page and return to R.
```

Social Media - Big Business

1	KATY PERRY	KATY PERRY @katyperry	109,350,521	212	9,136
2	Justin Bieber	Justin Bieber @justinbieber	106,300,993	314,988	30,607
3	Barack Obama	Barack Obama @BarackObama	102,003,213	623,115	15,510
4	Rihanna	Rihanna @rihanna	87,631,105	1,117	10,090
5	Taylor Swift	Taylor Swift @taylorswift13	85,595,101	0	91
6	Lady Gaga	Lady Gaga @ladygaga	78,388,370	127,610	8,729
7	Ellen DeGeneres	Ellen DeGeneres @TheEllenShow	77,772,841	35,853	16,570
8	Cristiano Ronaldo	Cristiano Ronaldo @Cristiano	72,602,242	95	3,158

Weird Results Can Be Useful

```
# Create a Function to Pull in Some Tweets

my_search_tweets <- function(string="Katy Perry",n=3000) {
  require(rtweet)
  tweet_table <- search_tweets(string,n=n,
                               retryonratelimit = TRUE,
                               include_rts = FALSE,
                               lang = "en")
  return(plain_tweets(tweet_table$text))
}

raw_tweets <- my_search_tweets("Katy Perry")
katy_raw_tweets <- raw_tweets # save for later
```

Weird Results Can Be Useful

"Safe" Tweets Collected on 4/24/18

```
[1] "@chenausky1776 @uw75 @PureMichGirl I can assure you that Katy Perry
is not a communist, you are just a little cry baby and just as bad as a
snowflake."
```

Pop Music Is Okay, but No Politics: Communist China Bans Katy Perry for Expressing her Opinions—With a Dress

Create a Corpus

Next we create a Vector Source from which we will create a Volatile Corpus:

```
library(tm)  # The tm package let's us create a Corpus

cleaned_tweets_vec  <- VectorSource(raw_tweets)
cleaned_tweets_corp <- VCorpus(cleaned_tweets_vec)

# Next we have to clean the Corpus. tm allows us to map predefined
# functions from its own library of cleanup functions. And we can
# define our own

tm_cleaner <- function(corpus, stop=stopwords("en"), rm_num=TRUE) {

 require(tm)
 corpus <- tm_map(corpus, removePunctuation)
 corpus <- tm_map(corpus, stripWhitespace)
 corpus <- tm_map(corpus, removeNumbers)
 corpus <- tm_map(corpus, content_transformer(tolower))
 corpus <- tm_map(corpus, removeWords, stop)
 corpus <- tm_map(corpus, content_transformer(function(x) gsub("http\\w+", "", x)))
 return(corpus)
}
```

Clean the Corpus

- **Next we create some stop words which are common words we wish to remove from the Corpus**

```
kp_stopwords <- c(stopwords("english"),
                  "katy","perry","official","video","hey","perrys",
                  "now","katyperry","youtube","american")

cleaned_tweets_corp <- tm_cleaner(cleaned_tweets_corp,
                                  stop=kp_stopwords)
```

- **Next we create a TermDocumentMatrix from which we create a numeric matrix**

```
cleaned_tweets_tdm <- TermDocumentMatrix(cleaned_tweets_corp)

# Save a copy of the cleaned tdm for later use
katy_cleaned_tweets_tdm <- cleaned_tweets_tdm

cleaned_tweets_tdm_m <- as.matrix(cleaned_tweets_tdm)

terms_freq <- rowSums(cleaned_tweets_tdm_m)
```

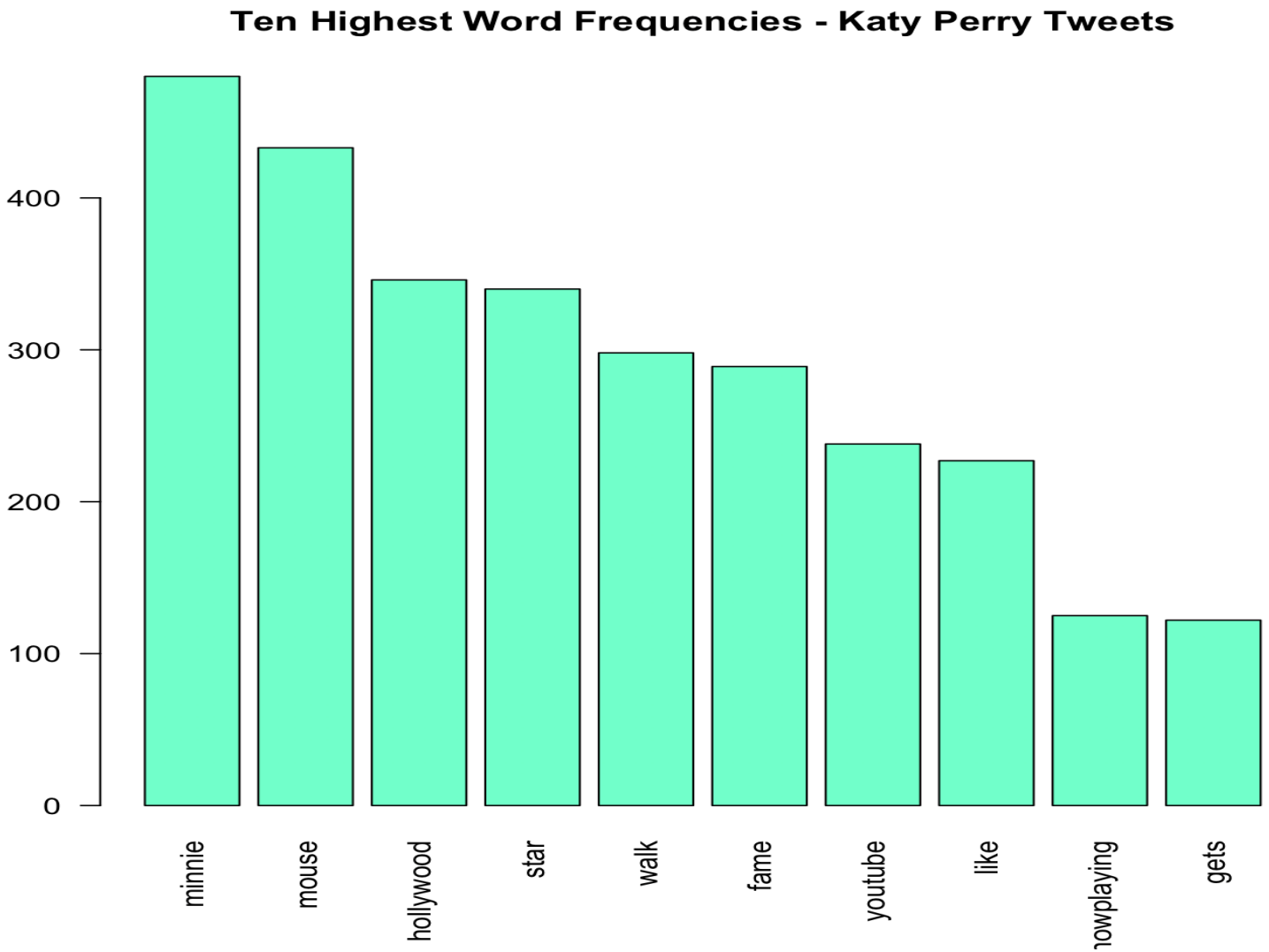
Barplot of Frequent Terms

Next we can create a barplot of the 10 most frequently appearing words

```
terms_freq_sorted <- sort(terms_freq,decreasing=T)

terms_freq_sorted[1:10]
    minnie      mouse  hollywood       star       walk       fame
       480        433        346        340        298        289
   youtube       like nowplaying      gets
       238        227        125        122

# But it would be better to look at a graph

barplot(terms_freq_sorted[1:10],
        col="aquamarine",
        las=2,
        main="Ten Highest Word Frequencies - Katy Perry Tweets")
```

Pre-Cleaning Data

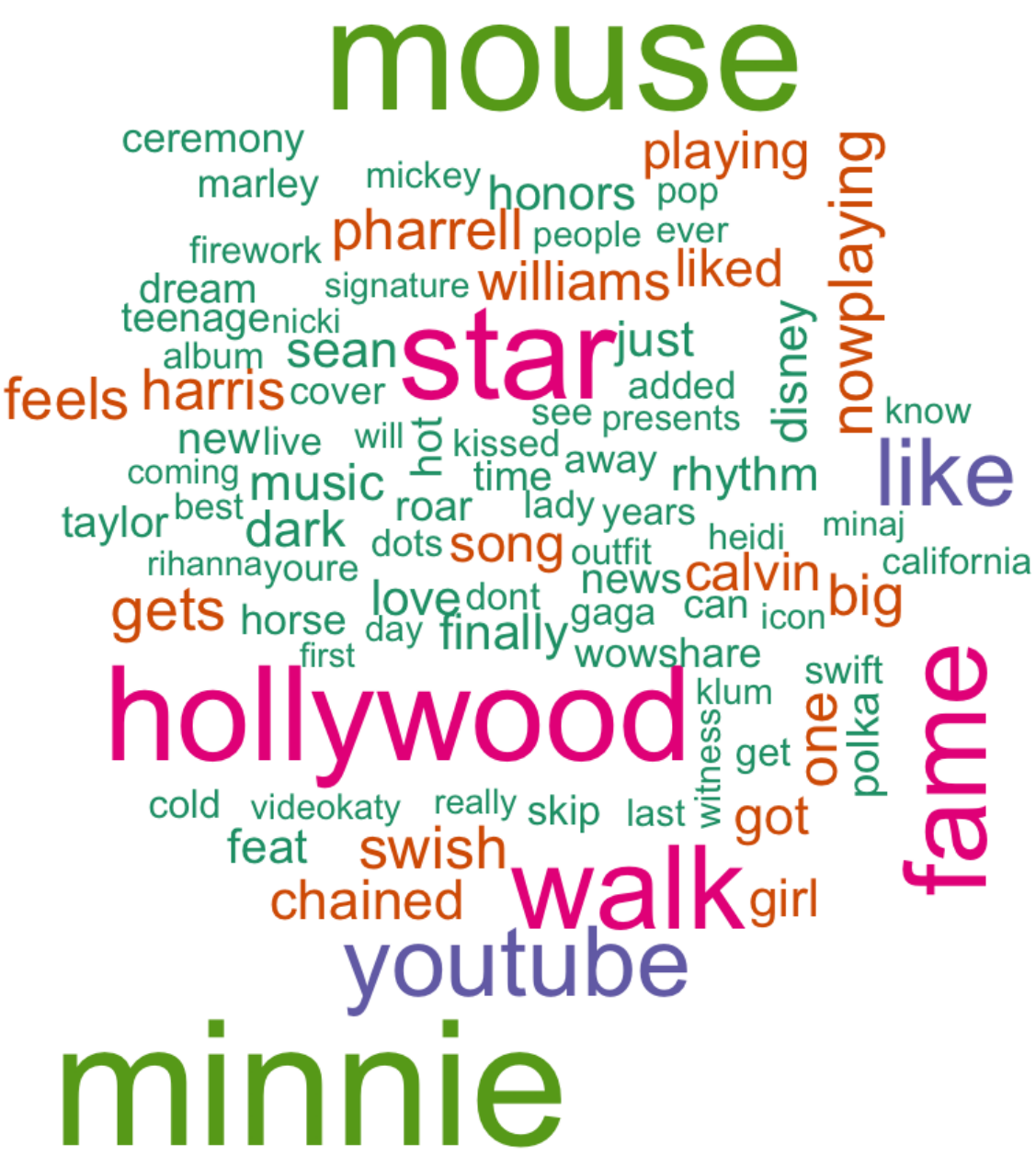
Next we can create a barplot of the 10 most frequently appearing words



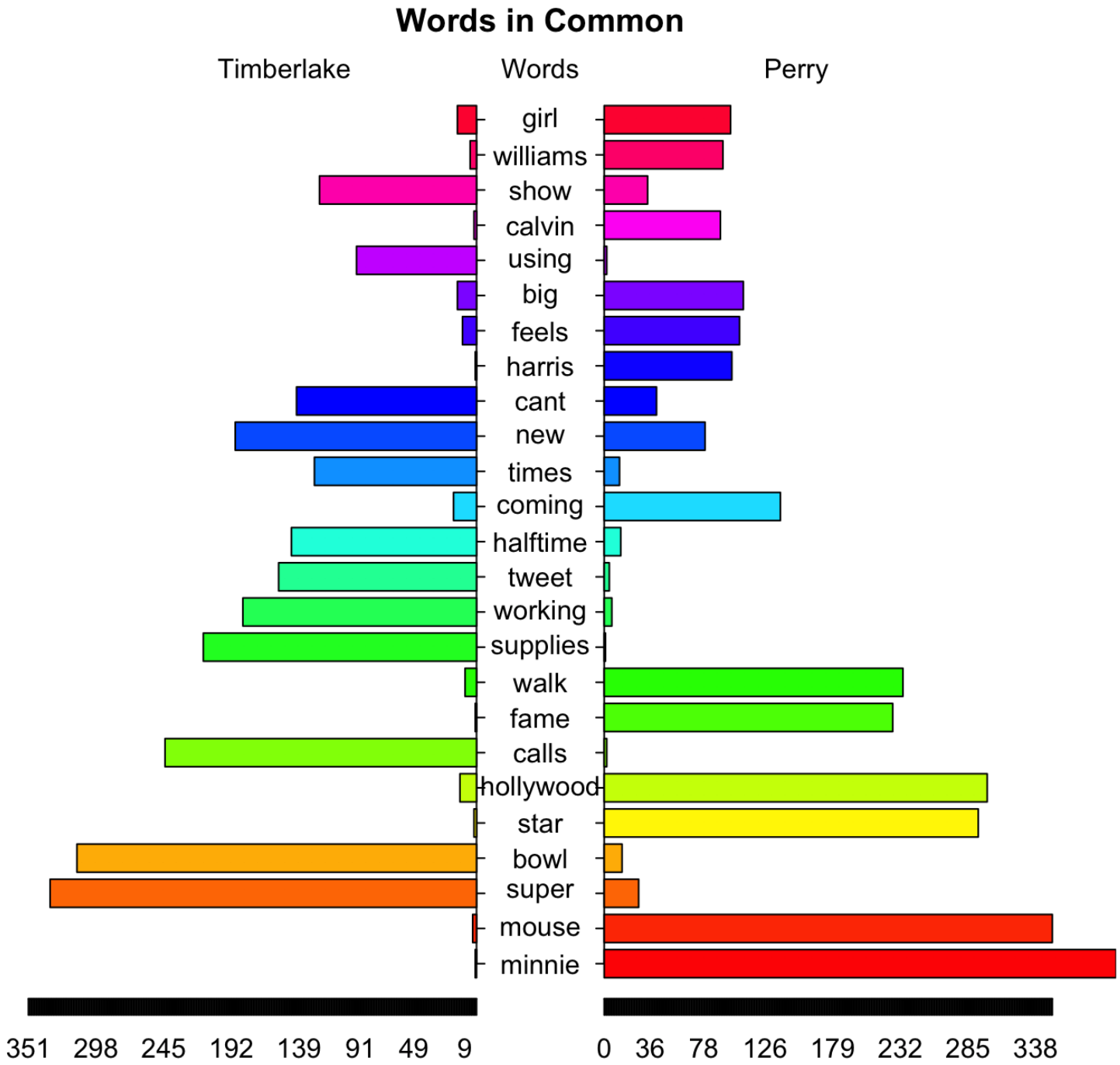
Wordcloud

But a Word Cloud would look better:

```
library(wordcloud)
library(RColorBrewer)

# We create a data frame of frequent terms and their
# respective frequencies

df <- data.frame(word=names(terms_freq_sorted),
                 freq=terms_freq_sorted,
                 stringsAsFactors = FALSE)

# We'll pick a nice color palette for a word cloud
# We need darker colors but not too dark

color_palette <- brewer.pal(5,"Dark2")

# Finally draw the word cloud
wordcloud(df$word,df$freq,max.words = 90,colors = color_palette)
```


mouse
ceremony
marley
mickey
honors
playing
pop
pharrell
people
ever
firework
williams
liked
dream
signature
teenage
nicki
star
just
sean
album
feels
harris
cover
added
see
presents
disney
nowplaying
know
new
live
will
hot
kissed
away
like
coming
music
time
rhythm
taylor
best
dark
roar
lady
years
minaj
dots
song
outfit
heidi
rihanna
youre
news
calvin
california
gets
horse
love
dont
gaga
can
icon
big
day
finally
wowshare
first
hollywood
klum
swift
fame
witness
one
polka
get
cold
videokaty
really
skip
last
got
feat
swish
walk
girl
chained
youtube
minnie

Turns out that Katy Perry paid tribute to Minnie Mouse


Words in Common
Timberlake
Words
Perry
girl
williams
show
calvin
using
big
feels
harris
cant
new
times
coming
halftime
tweet
working
supplies
walk
fame
calls
hollywood
star
bowl
super
mouse
minnie
351 298 245 192 139 91 49 9
0 36 78 126 179 232 285 338

We can generate other figures based on the raw tweets as well as the TDM

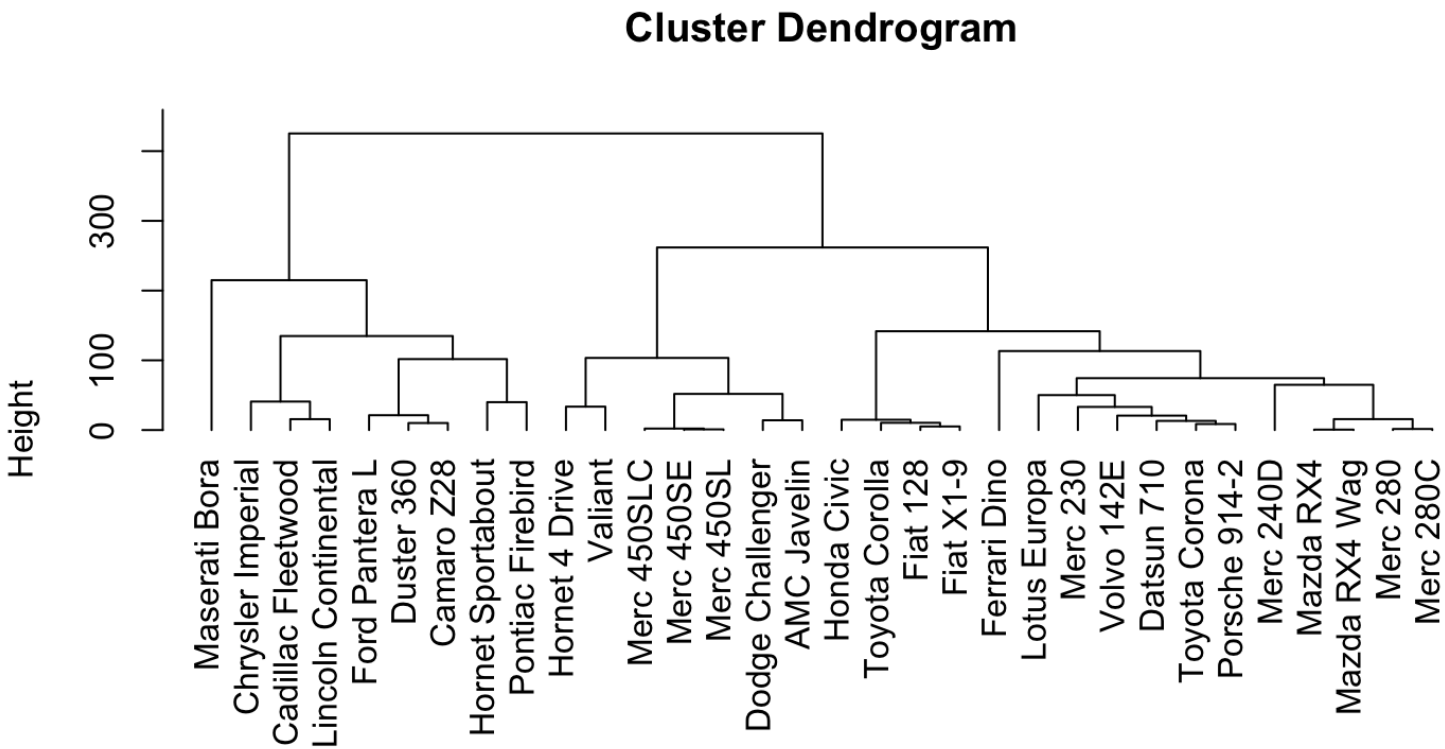
- Word Association Plots
- Distanxe Matrices / Dendrograms
- N-Grams and Bi-Grams

hclust

Let's look at a generating a word cluster using the **hclust** function:

```
# Compute the distance between the rows of mtcars

dist_m <- dist(mtcars)

# Create a clustered grouping using the matrix

clust <- hclust(dist_m)

# Plot a Dendrogram to visualize relationships

plot(clust,hang=-1)
```

Dendrogram

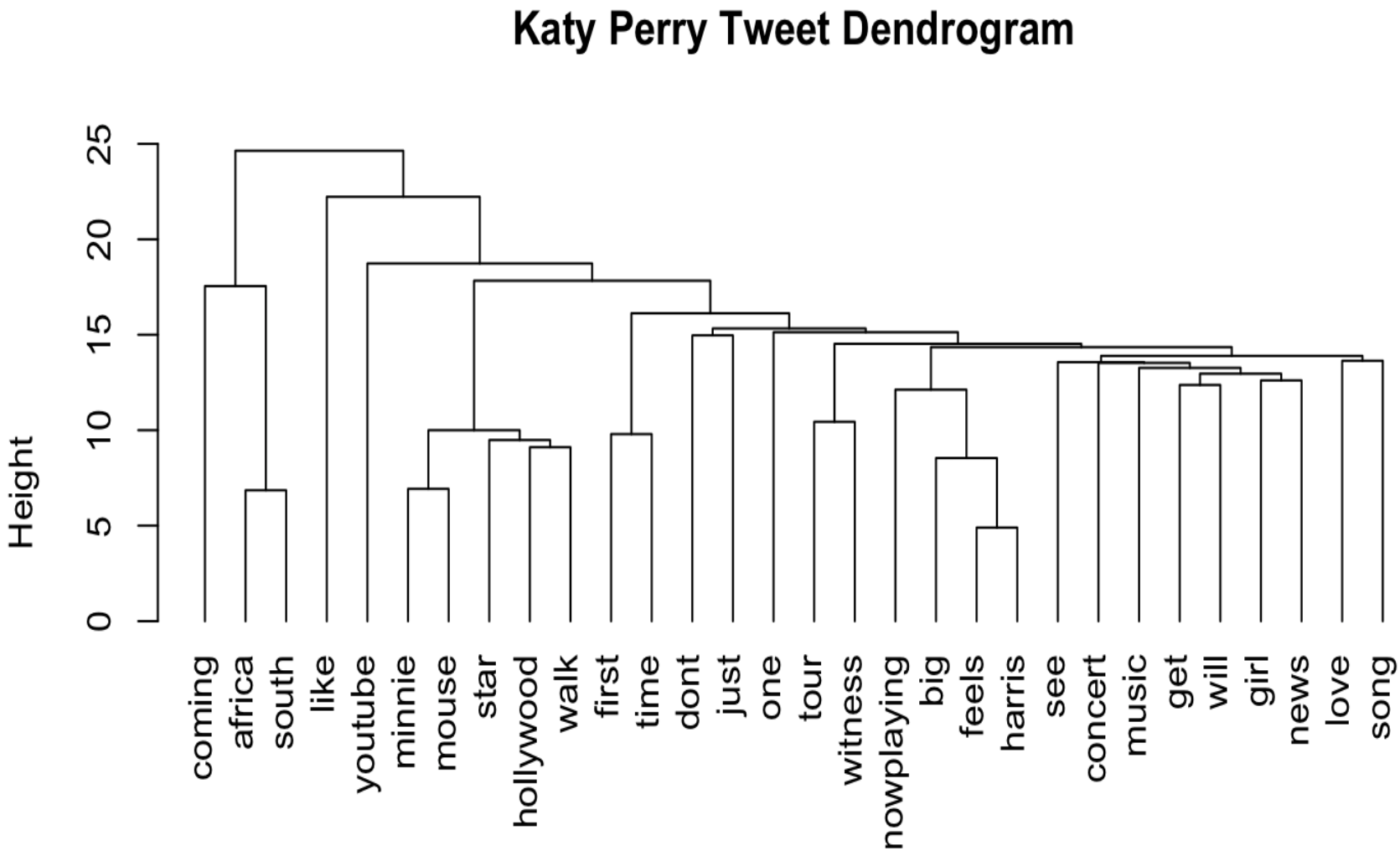

Cluster Dendrogram
Height
0
100
300
Maserati Bora
Chrysler Imperial
Cadillac Fleetwood
Lincoln Continental
Ford Pantera L
Duster 360
Camaro Z28
Hornet Sportabout
Pontiac Firebird
Hornet 4 Drive
Valiant
Merc 450SLC
Merc 450SE
Merc 450SL
Dodge Challenger
AMC Javelin
Honda Civic
Toyota Corolla
Fiat 128
Fiat X1-9
Ferrari Dino
Lotus Europa
Merc 230
Volvo 142E
Datsun 710
Toyota Corona
Porsche 914-2
Merc 240D
Mazda RX4
Mazda RX4 Wag
Merc 280
Merc 280C

hclust

We can do this with a Term Document Matrix which can have many more rows / tweets

```
cleaned_tweets_vec  <- VectorSource(katy_raw_tweets)
cleaned_tweets_corp <- VCorpus(cleaned_tweets_vec)

kp_stopwords <- c(stopwords("english"),"katy","perry","katyperry")

# Clean the Katy Perry tweets and create a TDM

cleaned_tweets_corp <- tm_cleaner(cleaned_tweets_corp, stop=kp_stopwords)
cleaned_tweets_tdm <- TermDocumentMatrix(cleaned_tweets_corp)

# Remove the sparse terms from the TDM
tweets_tdm <- removeSparseTerms(cleaned_tweets_tdm,sparse=0.975)

# Create a matrix from the TDM and then a data frame
# which is then used to create a distance object

tdm_m <- as.matrix(tweets_tdm)
tdm_df <- as.data.frame(tdm_m)
tweets_dist <- dist(tdm_df)
hc <- hclust(tweets_dist)

plot(hc,hang=-1,main="Katy Perry Tweet Dendrogram")
```

Dendrogram

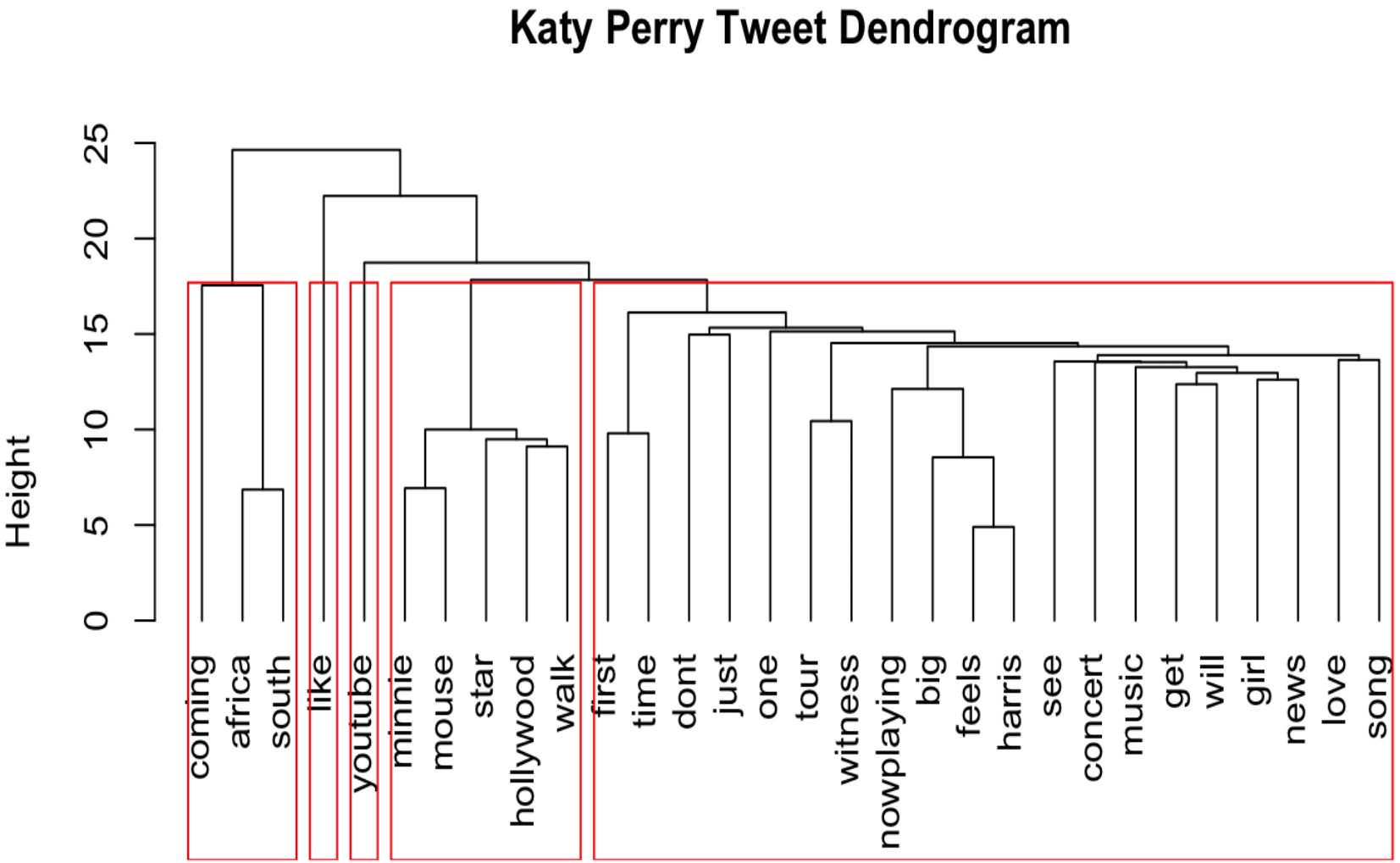

Katy Perry Tweet Dendrogram
Height
0
5
10
15
20
25
coming
africa
south
like
youtube
minnie
mouse
star
hollywood
walk
first
time
dont
just
one
tour
witness
nowplaying
big
feels
harris
see
concert
music
get
will
girl
news
love
song

hclust

We can decorate the dendrogram:

```
# Remove the sparse terms from the TDM
tweets_tdm <- removeSparseTerms(cleaned_tweets_tdm,sparse=0.975)

# Create a matrix from the TDM and then a data frame
# which is then used to create a distance object

tdm_m <- as.matrix(tweets_tdm)
tdm_df <- as.data.frame(tdm_m)
tweets_dist <- dist(tdm_df)
hc <- hclust(tweets_dist)

plot(hc,hang=-1,main="Katy Perry Tweet Dendrogram")

# Put rectangles for ech cluster
rect.hclust(hc,k=5)
```

Dendrogram


Katy Perry Tweet Dendrogram
Height
0
5
10
15
20
25
coming
africa
south
like
youtube
minnie
mouse
star
hollywood
walk
first
time
dont
just
one
tour
witness
nowplaying
big
feels
harris
see
concert
music
get
will
girl
news
love
song

dist hclust

A TDM should have about 50 to 80 terms for developing a dendrogram

- Most TDMs are sparse with a lot of zeroes
- We have been dealing with thousands of tweets
- The associated TDM can balloon easily with
- Use **removeSparseTerms** to eliminate low frequency terms
- Set the *sparse* argument to be close to 1 to reduce number of terms
- Set the *sparse* argument to be close to 0 to keep most of the terms
- Let's revisit the previous example to

removeSparseTerms

```
(cleaned_tweets_tdm <- TermDocumentMatrix(cleaned_tweets_corp))

<<TermDocumentMatrix (terms: 5623, documents: 2946)>>
Non-/sparse entries: 21743/16543615
Sparsity           : 100%
Maximal term length: 59
Weighting          : term frequency (tf)

# Remove the sparse terms from the TDM

(tweets_tdm <- removeSparseTerms(cleaned_tweets_tdm,sparse=0.975))

<<TermDocumentMatrix (terms: 30, documents: 2946)>>
Non-/sparse entries: 3521/84859
Sparsity           : 96%
Maximal term length: 10
Weighting          : term frequency (tf)
```

Sentiment

We can use the `SentimentAnalysis` package to help us

- Uses three standard "dictionaries" to judge sentiment
- Works with a Corpus, TDM, or raw text
- Good companion to the `tm` package
- Let's you compare responses across all three dictionaries
- Harvard IV - (Pyschological dictionary used in GI software)
- Henry's Finance - Financial Dictionary
- Loughran-McDonald - Journal of Finance

Sentiment

Basic Example

```
library(SentimentAnalysis)

text <- c("Gee Delta. Thanks for losing my luggage today",
          "I had a great flight and experienced few problems",
          "Delta. Never again will I fly using your services.")

sentiment <- analyzeSentiment(text)

convertToBinaryResponse(sentiment$SentimentQDAP)
#[1] positive positive positive
#Levels: negative positive

convertToBinaryResponse(sentiment$SentimentLM)
#[1] negative positive negative
#Levels: negative positive

convertToBinaryResponse(sentiment$SentimentGI)
# [1] positive positive positive
```

Sentiment

Katy Perry Tweets

```
sentiment <- analyzeSentiment(katy_cleaned_tweets_tdm)

str(sentiment)
'data.frame': 1000 obs. of  14 variables:
 $ WordCount          : num  9 6 5 6 11 4 8 5 7 5 ...
 $ SentimentGI        : num  0 0.1667 0 0 -0.0909 ...
 $ NegativityGI       : num  0 0 0 0.1667 0.0909 ...
 $ PositivityGI       : num  0 0.167 0 0.167 0 ...
 $ SentimentHE        : num  0 0 0 0 0 0 0 0 0 0 ...
 $ NegativityHE       : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PositivityHE       : num  0 0 0 0 0 0 0 0 0 0 ...
 $ SentimentLM        : num  0 0 0 0 0 0 0 0 0 0 ...
 $ NegativityLM       : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PositivityLM       : num  0 0 0 0 0 0 0 0 0 0 ...
 $ RatioUncertaintyLM: num  0 0 0 0 0 0 0 0 0 0 ...
 $ SentimentQDAP      : num  0 0.167 0 0 -0.182 ...
 $ NegativityQDAP     : num  0 0 0 0.167 0.182 ...
 $ PositivityQDAP     : num  0 0.167 0 0.167 0 ...
```

Sentiment

```
binary <- convertToBinaryResponse(sentiment$SentimentQDAP)

binary[1:10]
[1] positive positive positive positive negative positive positive
[8] positive positive positive

(binary_table <- table(binary))
negative positive
82      917

grep("negative",binary)[1:35]
[1]   5  29  30  32  34  39  40  42  44  52  58  61  68  72
[15] 87  88  93  97  98 108 115 132 137 158 171 172 183 185
[29 195 203 209 244 250 252 283
```

Sentiment

```
grep("negative",binary)[1:35]
[1]   5  29  30  32  34  39  40  42  44  52  58  61  68  72
[15] 87  88  93  97  98 108 115 132 137 158 171 172 183 185
[29 195 203 209 244 250 252 283

raw_tweets[c(29,30,32)]
[1] "Unpopular opinion but seeing a man (KATY PERRY) and
immediately interrupting him to tell him how attractive
he is (KATY PERRY) is just as problematic as if it were
a man doing it to a women."

[2] "Why does Satanist, Katy Perry want a convnt? @KatyPerry
blasts bankrupt house flipper who owes her $5m #RomanCatholic
#Christians #Christianity #Ccot @Church_Militant"

[3] "Shout out to @jeffschroeder23 for calling out double
standard on American Idol. Yes, Katy Perry ambush style kissing
male contestants on the mouth is way over the line wrong in todays
woke media. #NoDoubleStandards"
```

Bi Grams

You can look at pairs of words

```
library(RWeka)
 BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))

cleaned_tweets_tdm <- TermDocumentMatrix(cleaned_tweets_corp,
control = list(tokenize = BigramTokenizer))
```

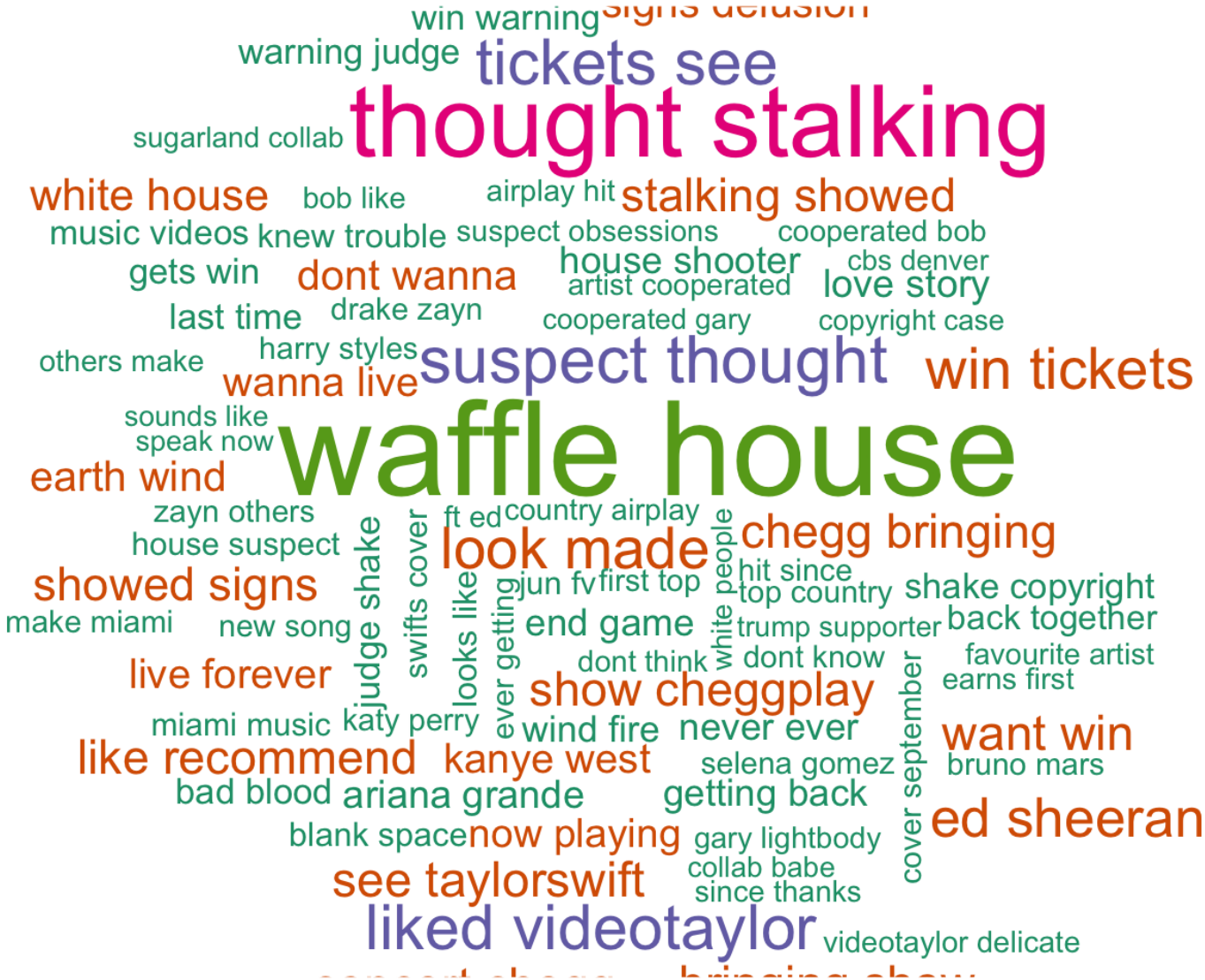
Combined Function

```
myCleaner <- function(tweets, wordcloud=TRUE, mystop=stopwords(),bigram=TRUE) {
  dyn.load(paste0(system2('/usr/libexec/java_home', stdout = TRUE), '/jre/lib/server/libjvm.dylib'))
  x <- c("tm","wordcloud","RColorBrewer","rtweet","RWeka")
  lapply(x, require, character.only = TRUE)

  plain_tweets_corp <- VCorpus(VectorSource(plain_tweets(tweets)))
  cleaned_tweets_corp <- tm_cleaner(plain_tweets_corp,stop=mystop)

  # CREATE THE TDM and TDM MATRIX

  BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))

  if (bigram) {
    cleaned_tweets_tdm <- TermDocumentMatrix(cleaned_tweets_corp, control = list(tokenize = BigramTokenizer))
  } else {
    cleaned_tweets_tdm <- TermDocumentMatrix(cleaned_tweets_corp)
  }

  cleaned_tweets_tdm_m <- as.matrix(cleaned_tweets_tdm)

  if (wordcloud) {
    terms_freq <- rowSums(cleaned_tweets_tdm_m)
    terms_freq_sorted <- sort(terms_freq,decreasing=T)

    df <- data.frame(word=names(terms_freq_sorted),freq=terms_freq_sorted,
                     stringsAsFactors = FALSE)

    color_palette <- brewer.pal(5,"Dark2")
    wordcloud(df$word,df$freq,max.words = 90,colors = color_palette)
  }

  # RETURN THE TDM OBJECT
  return(cleaned_tweets_tdm)
}
```

Combined Function

```
raw_tweets <- my_search_tweets("Taylor Swift")

stp <- c("taylor","swift","youtube",stopwords("en"))

myCleaner(raw_tweets,mystop=bigram=TRUE)
```

Combined Function



Manny The Cat - Master of Social Media