

Introduction to **R**: Text Mining/Analysis

Dave Armstrong

University of Wisconsin-Milwaukee

Department of Political Science

e: armstrod@uwm.edu

w: www.quantoid.net/UWMDDataDay.html

This workshop will require us to use a lot more code than the last workshop. There is no GUI that is aimed at text mining/analysis. Further, we will need to know something about not only R, but also html/CSS. We'll look at some examples as we go along. First, we'll learn how to use some useful tools.

Regular Expressions

Regular expressions are essentially search patterns used for finding and extracting specific text from within strings.

This is not a full workshop on regular expressions, but a short introduction will help clarify the uses of the tool and will prepare us for learning to scrape the web. For this short piece of instruction, we will use the website <http://regex101.com>. This is a place where you can test out your regular expressions. We will work with the first sentence from the Gettysburg Address to test out some of the main ideas.

- `[a-z]` matches lowercase letters
- `[A-Z]` matches uppercase letters
- `[0-9]` matches the digits (a shortcut for this is `\d`, `\D` matches everything but a digit character).
- `+` allows for multiple instances of the preceding character(s). For example `\d+` will match any number of digits in a row. `*` allows for multiple matches (but includes the possibility of zero)
- `.` matches nearly any character (it does match white space, letters, digits, it does not match line break characters).
- `{n,m}` finds any match of the preceding character between `n` and `m` characters long.
- `\s` is a white space character and `\S` is anything but a white space character.

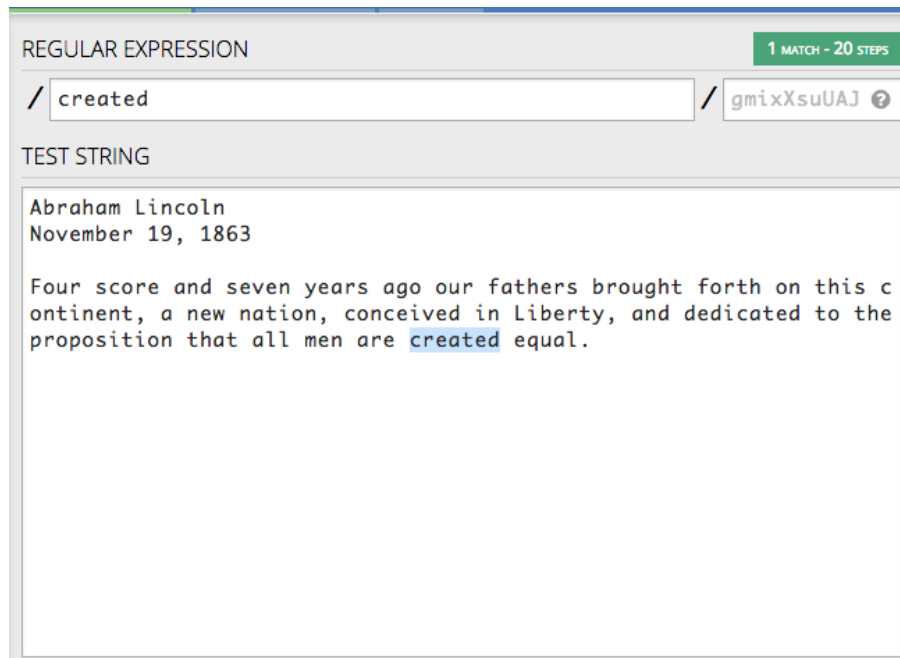
- `\w` is a word character and `\W` is anything but a word character. Similarly, `\b` is a word boundary (i.e., something that separates words).

Here is the text we're looking at:

```
Abraham Lincoln  
November 19, 1863
```

```
Four score and seven years ago our fathers brought forth on this continent, a new nation,  
conceived in Liberty, and dedicated to the proposition that all men are created equal.
```

If we were looking for the presence of a single word, it would be easy enough. For example, if we wanted to know if the word `created` was included, we could do:



Ultimately, that's not particularly interesting because we could do that lots of other easier ways, too. Regular expressions, for our purposes are good for extracting words. For instance, lets say we wanted to find all of the numbers.

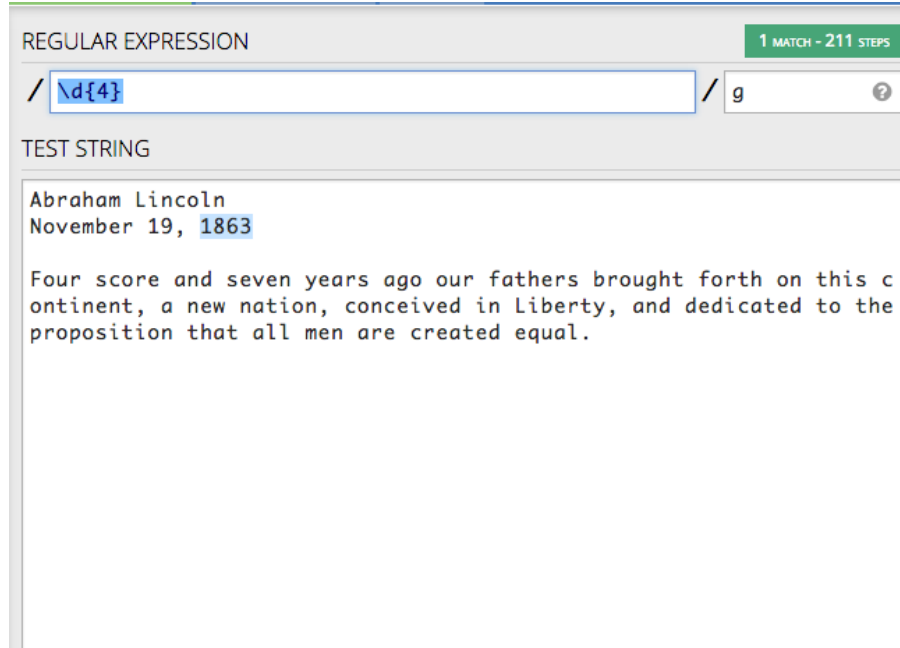
The screenshot shows a web-based regular expression tool. At the top, it says "REGULAR EXPRESSION" and "2 MATCHES - 211 STEPS". The regular expression input field contains `\d+` and the search button is labeled `g`. Below, the "TEST STRING" section contains the text: "Abraham Lincoln", "November 19, 1863", and a paragraph of text from the Gettysburg Address. The numbers "19" and "1863" are highlighted in blue, indicating they are matches for the `\d+` pattern.

We could do this in R with:

```
> library(stringr)
> # makes an object called "x" with the relevant text
> source("http://www.quantoid.net/DD/ga.R")
> str_extract_all(x, "\\d+")

[[1]]
[1] "19" "1863"
```

If we only wanted to extract the year,

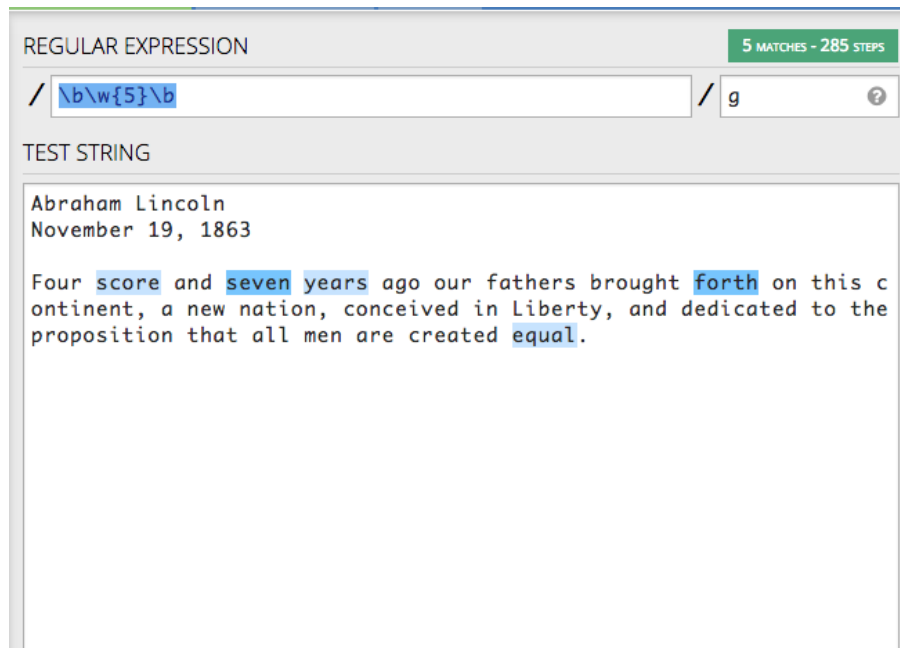


The screenshot shows a web-based regular expression tool. The 'REGULAR EXPRESSION' field contains `\d{4}` and the 'TEST STRING' field contains the text: "Abraham Lincoln November 19, 1863 Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal." A green box in the top right corner indicates "1 MATCH - 211 STEPS". The year "1863" in the test string is highlighted in blue.

```
> str_extract_all(x, "\\d{4}")
```

```
[[1]]  
[1] "1863"
```

If we wanted to extract all of the five-letter words:

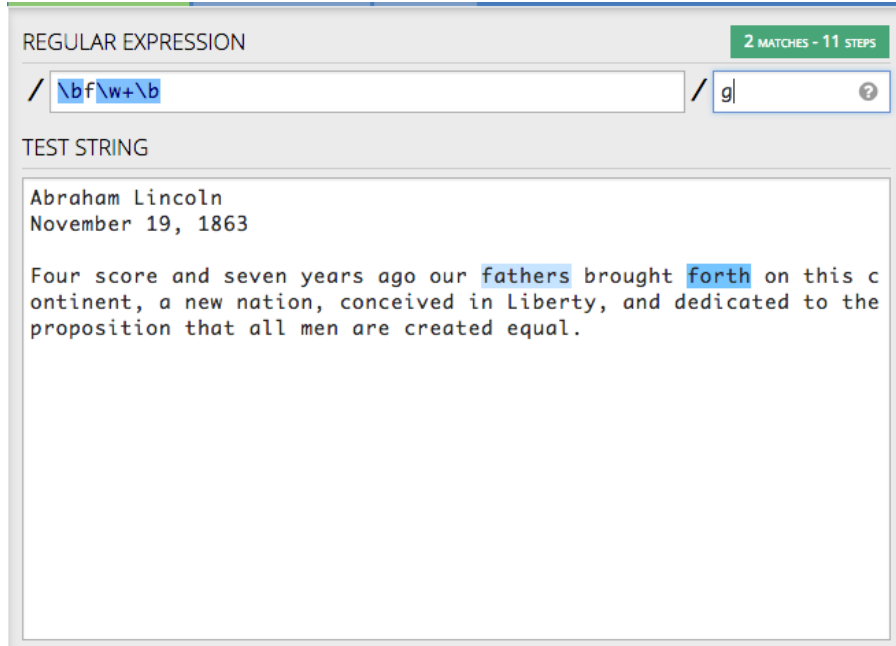


The screenshot shows the same regular expression tool. The 'REGULAR EXPRESSION' field contains `\b\w{5}\b` and the 'TEST STRING' field contains the same text as above. A green box in the top right corner indicates "5 MATCHES - 285 STEPS". The words "score", "seven", "years", "forth", and "equal" in the test string are highlighted in blue.

```
> str_extract_all(x, "\\b\\w{5}\\b")
```

```
[[1]]  
[1] "score" "seven" "years" "forth" "equal"
```

Finally, at least for now, if we wanted to find all of the words starting with `f`.



```
> str_extract_all(x, "\\bf\\w+\\b")
```

```
[[1]]
```

```
[1] "fathers" "forth"
```

Note that this didn't catch `Four` because it starts with `F` and not `f`. There are two ways to fix that:

```
> str_extract_all(x, "\\b[fF]\\w+\\b")
```

```
[[1]]
```

```
[1] "Four" "fathers" "forth"
```

```
> str_extract_all(to_lower(x), "\\bf\\w+\\b")
```

```
[[1]]
```

```
[1] "four" "fathers" "forth"
```

The string `[fF]` will match either `f` or `F`. The second option converts all letters to lowercase and then matches, so it changes `Four` to `four` and then matches the `f` at the beginning of `four`.

With these ideas in mind, let's consider a couple of different use cases.

The page is not something pretty to look at, but we don't need to look at it. We just want to extract the html tags.

```
> links <- html_nodes(page, "a")
> links <- html_attr(links, "href")
```

This does extract all of the links, but if you look at the list (truncated in the interest of space), there are links that relate to reviews and those that relate to other things:

```
> links[c(1:5, 201:205)]

[1] "http://thedissolve.com"
[2] "http://pitchfork.com"
[3] "/"
[4] "http://twitter.com#!/pitchfork"
[5] "http://www.facebook.com/Pitchfork"
[6] "/reviews/albums/20456-jeremiah-jae-lorange-the-night-took-us-in-like-family/"
[7] "/reviews/albums/20514-beauty-pill-beauty-pill-describes-things-as-they-are/"
[8] "/reviews/albums/20498-your-good-fortune-ep/"
[9] "/reviews/albums/20420-sound-color/"
[10] "/reviews/albums/20530-new/"
```

We want only the links starting with `/reviews`. We can find out which ones these are by using the `str_detect` function from `stringr`.

```
> revlinks <- which(str_detect(links, "\\s*/reviews"))
> revlinks[1:10]

[1] 17 18 31 32 33 199 200 201 202 203

> links <- links[revlinks]
```

This returns the observation numbers (with `which`) where the string is detected (with `str_detect`). This is better, but still not perfect. There are some links that are not album reviews, but links to other sections of the webpage, but still under the “reviews” sub-head. What you will notice is that the reviews links include a five-digit number.

```
> links[1:10]

[1] "/reviews/albums/"
[2] "/reviews/tracks/"
[3] "/reviews/best/albums/"
[4] "/reviews/best/tracks/"
[5] "/reviews/best/reissues/"
[6] "/reviews/albums/20529-young-thug-barter-6/"
[7] "/reviews/albums/20408-sheer-mag-ii-ep/"
[8] "/reviews/albums/20456-jeremiah-jae-lorange-the-night-took-us-in-like-family/"
[9] "/reviews/albums/20514-beauty-pill-beauty-pill-describes-things-as-they-are/"
[10] "/reviews/albums/20498-your-good-fortune-ep/"
```

Let's pull out the links with five-digit numbers in them.

```
> ind <- which(str_detect(links, "\\d{5}"))
> links <- links[ind]
> links[1:10]
```

```
[1] "/reviews/albums/20529-young-thug-barter-6/"
[2] "/reviews/albums/20408-sheer-mag-ii-ep/"
[3] "/reviews/albums/20456-jeremiah-jae-lorange-the-night-took-us-in-like-family/"
[4] "/reviews/albums/20514-beauty-pill-beauty-pill-describes-things-as-they-are/"
[5] "/reviews/albums/20498-your-good-fortune-ep/"
[6] "/reviews/albums/20420-sound-color/"
[7] "/reviews/albums/20530-new/"
[8] "/reviews/albums/20523-makes-a-king/"
[9] "/reviews/albums/20528-the-float/"
[10] "/reviews/albums/20496-transient/"
```

Now, we have the links to follow, we can set up a loop to go through the links and extract whatever information we want. The question is - how do we get at the relevant information? One easy way to do this is with <http://selectorgadget.com/>. This will show you the CSS selectors (things like divs, paragraphs, spans, etc...) that are relevant for the required text.



We see that the score, which we want, is in a span tag. Specifically, the code looks like this:

```
<span class="score score-7-9"> 7.9 </span>
```

We could find all of the `` nodes and pull them out.

```
> ses <- html_session("http://pitchfork.com/reviews/albums/")
> l <- links[str_detect(links, "foil-deer")]
> tmp.rev <- jump_to(ses, l)
> scores <- html_nodes(tmp.rev, "span") %>% html_text()
> scores[15:20]

[1] " | "
[2] "Ian Cohen finds out how the mastermind behind Chromatics and Glass Candy perfectedÃ"
[3] "April 20, 2015"
[4] " 7.9 "
[5] " Speedy Ortiz: \"Raising The Skate\" (via SoundCloud) \n"
[6] ""
```


Notice that there are lots of elements in here. There are two ways we could get the score.

- Find the entries with one or two digits and then a third digit, with a period between the first two.

```
> id <- which(str_detect(scores, "\\d{1,2}\\\\.\\d{1}"))
> scores[id]

[1] " 7.9 "
```

- Use a more targeted selector.

```
> score <- html_text(html_node(tmp.rev, xpath="//span[starts-with(@class, 'score')]"))
> str_trim(score)

[1] "7.9"
```

OK, now that we know how to grab the scores, how do we get the artist, album title and review? Let's start with the artist. Here is one place where the artist shows up.

```
<h1><a href="/artists/31267-speedy-ortiz/">Speedy Ortiz</a></h1>
```

Just like above, if we grab everything inside the `<h1>` tag, we will get many results. So we need a more targeted search. We are looking for links inside an `<h1>` tag and those where the link itself starts with `/artists/`. With a bit of trial and error, we recognize that the review and relevant details are all in the `<div id='main'>` tag. The only link inside a `h1` tag is the artist. We can get this with:

```
> artist <- html_nodes(tmp.rev, xpath="//div[starts-with(@id, 'main')] //h1 //a")
> artist <- html_text(artist)
> artist

[1] "Speedy Ortiz"
```

We can get the album with a similarly targeted search, though here the `h2` tag holds the album title.

```
<h2>Foil Deer</h2>
```

```
> album <- html_nodes(tmp.rev, xpath="//div[starts-with(@id, 'main')] //h2")
> album <- html_text(album)
> album

[1] "Foil Deer"
```

Now, we just have to capture the text of the review. Again, we can do this with a targeted search for a `div` of class `editorial`

```
> review <- html_nodes(tmp.rev,
+   xpath="//div[starts-with(@id, 'main')] //div[starts-with(@class, 'editorial')]")
> review <- html_text(review)
> substr(review, 1, 50)

[1] " OnÃSpeedy OrtizÃ's 2013 debutÃMajor Arcana,ÃSadie "
```

Now we need to put it all together. We need to consider a couple of other tools here. We need to know how to store the data and how to loop over values of a vector. Let's take the second one first.

Looping

The `for` loop is a very useful tool when dealing with aggregating over units or performing operations multiple times either on the same set of data or on different sets of data. To show the basic structure of a loop, consider the following example:

```
> n <- c("one", "two", "three", "four")
> for(i in 1:4){
+   print(n[i])
+ }

[1] "one"
[1] "two"
[1] "three"
[1] "four"
```

Here, the solitary character `i` holds the place for the numbers 1, 2, 3, and 4 in turn. It would be equivalent to do the following:

```
> n[1]

[1] "one"

> n[2]

[1] "two"

> n[3]

[1] "three"

> n[4]

[1] "four"
```

Though you often see `i` used as an index, you could perform the same task with:

```
> for(fred in 1:4){
+   print(n[fred])
+ }

[1] "one"
[1] "two"
[1] "three"
[1] "four"
```

Often times, when doing many tasks, we loop over a sequence of integer values from 1 to the number of times we want to do something. You could also loop over any set of values:

```
> for(i in c("one", "two", "three")){
+   print(i)
+ }

[1] "one"
[1] "two"
[1] "three"
```

Storing Data

As I mentioned on the first day, a rectangular dataset is only one of the ways R can store data. While it has many nice properties, a rectangular array is not necessarily what we want here. An entry in a variable is probably not the best way to store the information from the review. Here, we will use a list, which is simply a collection of elements of any kind. We will put the album, artist and score in a vector and store that information. We will also put the review as an element of the list. The whole function would look as follows:

```
> # initialize the result
> result <- list()
> # loop over links
> for(i in 1:length(links)){
+   # jump to the link from the main page
+   tmp.rev <- jump_to(ses, links[i])
+   # find the score
+   score <- html_text(html_node(tmp.rev, xpath="//span[starts-with(@class, 'score')]"))
+   score <- str_trim(score)
+   # find and capture the artist
+   artist <- html_node(tmp.rev, xpath="//div[starts-with(@id, 'main')] //h1 //a")
+   artist <- html_text(artist)
+   # find and capture the album
+   album <- html_node(tmp.rev, xpath="//div[starts-with(@id, 'main')] //h2")
+   album <- html_text(album)
+   # find and capture the review
+   review <- html_nodes(tmp.rev,
+     xpath="//div[starts-with(@id, 'main')] //div[starts-with(@class, 'editorial')]")
+   review <- html_text(review)
+   # organize artist, album, score info into a vector
+   s <- c(artist=artist, album=album, score=score)
+   # save all scraped data into an element of the result list
+   result[[i]] <- list(info = s, review=review)
+ }
```

We could look at all of the score data with.

```
> res <- t(sapply(result, function(x)x$info))
> head(res)
```

	artist	album	score
[1,]	"Young Thug"	"Barter 6"	"8.4"
[2,]	"Sheer Mag"	"II EP"	"8.1"
[3,]	"Jeremiah Jae"	"The Night Took Us in Like Family"	"7.1"
[4,]	"Beauty Pill"	"Beauty Pill Describes Things As They Are"	"7.5"
[5,]	"Mavis Staples"	"Your Good Fortune EP"	"6.8"
[6,]	"Alabama Shakes"	"Sound & Color"	"8.1"

Analyzing the Text

If we wanted to do some quantitative analysis on the text, we would first need to clean the text and organize it in a way that R can use. For this we need the `tm` and `plyr` packages. We are also going to grab some more reviews, too. Rather than do this in class, I've

downloaded them in the same fashion and compiled an object of the same format that you can download from the website (<http://www.quantoid.net/DD/pitchfork2k.rda>). Then, you can just load the file in R. The object will be called `p2k`.

```
> setwd("~/Dropbox/IntroR/UWM/Day3/")
> load("pitchfork2k.rda")
> revs <- sapply(p2k, function(x)x$review)
```

Next we need to let R know that all of these reviews belong to a corpus of material that we will want to analyze. The `tm` package is designed to deal with these sorts of sources.

```
> library(tm)
> library(plyr)
> corp <- Corpus(VectorSource(revs))
> corp <- tm_map(corp, tolower)
> corp <- tm_map(corp, removePunctuation)
> corp <- tm_map(corp, removeNumbers)
> corp <- tm_map(corp, removeWords, stopwords('english'))
> corp <- tm_map(corp, stripWhitespace)
> corp <- tm_map(corp, PlainTextDocument)
> corp <- tm_map(corp, stemDocument, language="english")
> tdm <- TermDocumentMatrix(corp)
```

Now, the object `tdm` is a term document matrix which has 51325 rows, each one corresponding to a different stemmed term in the document and 2000 columns, each one corresponding to an album review. The next thing we could do is take out some of the sparse terms, in an effort to find a core set of words that might help categorize the albums.

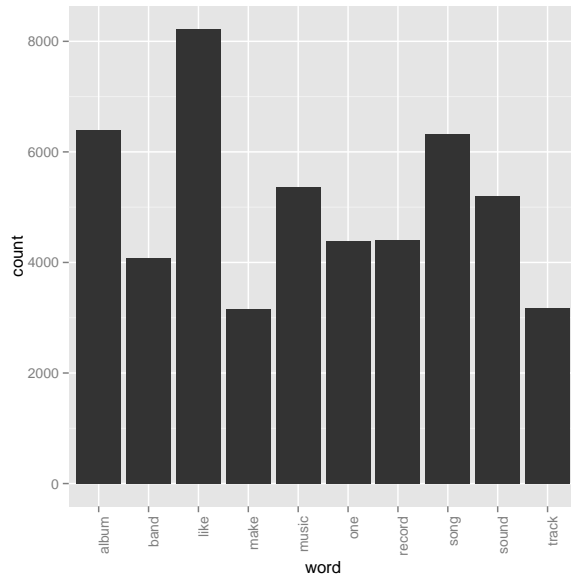
```
> tmp <- removeSparseTerms(tdm, .75)
> tmp <- t(tmp)
> score <- as.numeric(sapply(p2k, function(x)x$info)[3,])
> len <- sapply(corp, nchar)[1, ]
> tmp <- cbind(length=len, score=score, as.matrix(tmp))
> tmpdf <- as.data.frame(tmp)
```

At this point, you could save the `tmpdf` object and load it into Deducer to investigate graphically. Ultimately, however, what we'll want for many of the things we might do is a total count of how often different words are used across documents.

```
> ags <- colSums(tmp[, -c(1,2)])
> agdf <- data.frame(count=ags, word=names(ags))
```

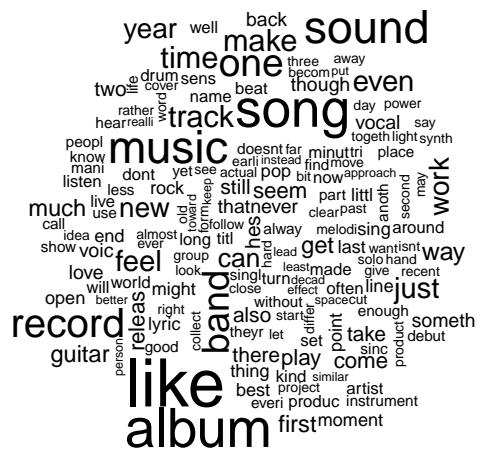
Now, we could make a bar plot of the most frequent words.

```
> library(ggplot2)
> agdf <- agdf[order(agdf[,1], decreasing=T), ]
> agdf$wnum <- 1:nrow(agdf)
> agdf$wnum <- factor(agdf$wnum, labels=rownames(agdf))
> ggplot() + geom_bar(aes(x = word, y = count), data=agdf[1:10, ], stat = 'identity') +
+   theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Next, we could make a Word Cloud:

```
> library(wordcloud)
> wordcloud(agdf$word, agdf$count)
```

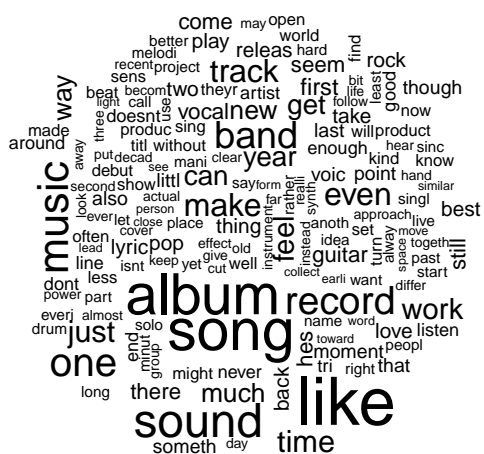


What if we wanted a word cloud based on score.

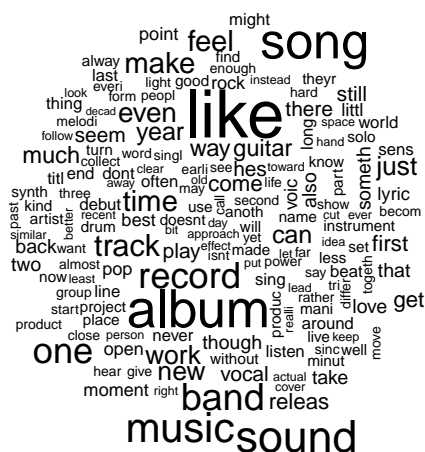
```
> q <- quantile(tmpdf$score, c(0, .25, .5, .75, 1))
> score_cat <- cut(tmpdf$score, breaks=q)
> ag2 <- by(tmp[, -c(1, 2)], list(score_cat), colSums)
> ag2 <- do.call(cbind, ag2)

> wordcloud(rownames(ag2), ag2[, 1])
> wordcloud(rownames(ag2), ag2[, 2])
> wordcloud(rownames(ag2), ag2[, 3])
> wordcloud(rownames(ag2), ag2[, 4])
```

Figure 1: wordclouds by score



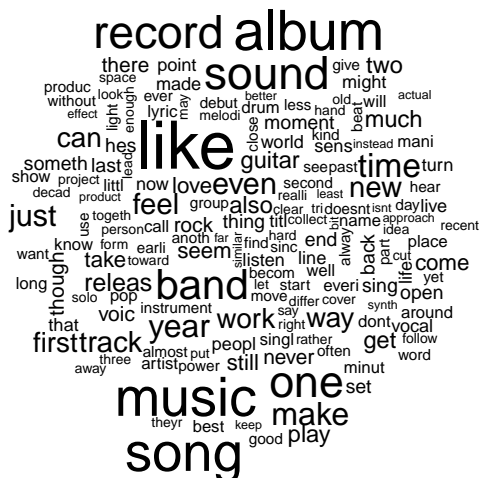
(a) 1st quartile



(b) 2nd quartile



(c) 3rd quartile



(d) 4th quartile

We could also do some quantitative analysis. First, we can find associations among words:

```
> findAssocs(tdm, "good", .14)
```

```

good
waypoint 0.16
get      0.15
just     0.15
hard     0.14
make     0.14

```

```
namea    0.14
shoebox  0.14
```

We could also look at a more complicated tree-based regression model for score:

```
> library(randomForest)
> N <- nrow(tmp)
> train <- sample( 1:nrow(tmp), floor(N/2), replace=F)
> test <- (1:nrow(tmp))[-train]
> y <- tmpdf$score
> ranf <- randomForest(y=y[train], x=as.matrix(tmpdf[train, -2]))
> imp <- ranf$importance
> imp <- imp[order(imp, decreasing=T), , drop=F]
> imp[1:15, , drop=F]
```

	IncNodePurity
length	87.03082
music	15.72706
record	14.42008
album	12.97612
space	12.54727
pop	12.38725
new	12.28719
group	12.15389
one	11.83239
least	11.24670
song	11.04606
less	10.66965
even	10.45839
peopl	10.19111
much	10.15339

Twitter

There is a packaged called `twitterR` that allows you easy access to tweets:

```
> consumer_key <- 'ahcyHdH0myoDXowxWSgDUW0B1'
> consumer_secret <- 'JEyZHcZhJY9n32jV7x15gq8NWV0v34GLF8z63eby0J6V27znmz'
> access_token <- '212334901-piyPsDlFP5sJqD1vdb4Bh0opIJyfytRDzX8dw3rS'
> access_secret <- '06SQJ0z1UPN6WgZtqsAQKEwd3RFyDZh2cmZh6emzU41NX'
> #necessary file for Windows
> # download.file(url="http://curl.haxx.se/ca/cacert.pem", destfile="cacert.pem")
>
> library(twitterR)
> setup_twitter_oauth(consumer_key,
+                     consumer_secret,
+                     access_token,
+                     access_secret)

[1] "Using direct authentication"

> #the cainfo parameter is necessary only on Windows
> tweets <- searchTwitter("#UWMvital", n=1500)
> # on Windows
> # tweets <- searchTwitter("#hillary2016", n=1500, cainfo="cacert.pem")
> tweet_text <- sapply(tweets, function(x)x$getText())
```

We could find the most re-tweeted tweet for this hash:

```
> retweets <- tweet_text[which(str_detect(tweet_text, "~RT"))]
> retweets <- str_replace_all(retweets, "\\@\\w+\\b", "")
> retweets <- str_replace_all(retweets, "http.+\\b", "")
> tab <- table(retweets)
> tab <- tab[order(tab, decreasing=T)]
> t(t(tab))
```

retweets

```
RT : #UWM has a $1.5+ billion impact on WI's economy. Hear from 5 top #Milwaukee voices on what makes #UWMvital:
RT : We provide a pathway to success for our great student-athletes #UWMvital #PantherProud
RT : #UWM has a $1.5 billion-plus impact on WI. Hear from 5 Milwaukee leaders on what makes #UWMvital.
RT : Meet #UWM, a living laboratory of innovation, engaged research & diverse students:
RT : Did you know 118,000 live and work in Wisconsin?
RT : "Milwaukee would fall on the other side of the tracks without #UWM." - President Tim Sheehy
RT : #uwmvital La Table Française c'est super! The French Table nourishes our minds & bodies!
RT : "When we're trying to solve a problem, #UWM is the first place we look." - Daniel Bader of
RT : #UWM Nursing Centers empower, educate, and care for over 8,000 Milwaukeeans yearly! #UWMvital
RT : 118,000 #UWM graduates live and work in Wisconsin. We're vital to Wisconsin. How is #UWMvital to you?
RT : Neighbors, nurses, partners shape #UWM's prescription for improving health care:
RT : #UWM: vital to Wisconsin for the literary culture it fosters & the writers editors & books it produces! #UWMvital
RT : #uwmvital UWM lets Wisconsin communicate with the world: we teach 21 languages & have MANY globally-focused majors!
RT : If you haven't already checked out the #UWMvital video, take a few minutes now -
RT : MMAC President Tim Sheehy shares why is vital to the city driving Wisconsin's economy #uwmvital
RT : Watch how 's talent and research drives Milwaukee's economy and community:
RT : Who volunteers 160 million hours per year? Wisconsinites. #Nonprofit sector snapshot via (PDF):
RT : Why is #UWMvital? Hear from 5 influential #Milwaukee leaders on #UWM's $1.5+ billion economic impact in Wisconsin:
RT : #uwmvital to our community, leading meaningful research & my own personal growth proud
RT : students in the support #uwmvital
RT : #UWM's advocacy for young people in urban centers makes #UWMvital to Superintendent :
RT : #uwmvital to our community, leading meaningful research, and my own personal growth. proud #gradstudent
RT : . lit a flame. keeps it burning. MAY 1:
RT : . is one of 300+ community partners who call #UWM a "true collaborative partner."
RT : A few Lubar #EMBA students & faculty featured in new #uwmvital video.
RT : Did you know that 118,000 #UWM graduates live and work in Wisconsin? Spread the word! #UWMVital
RT : Last day of the #UWMvital contest! How is #UWM vital to you?
RT : My latest project with my co-lifer, ! Proud to share so many reasons that make #uwmvital.
RT : RT : #UWM has a $1.5+ billion impact on WI's economy. Hear from 5 top #Milwaukee voices on what makes #UWMvital:
RT : We are proud to be a longstanding community partner of UW-Milwaukee! Thank you, , for the work you do:
RT : Where mentors can engage w/ their students & foster future UWM leaders. A place where you feel welcome! #UWMvital
```

From this point, you can read the tweets into a Corpus and do whatever we already did with them.

Try it yourself!

- How would you scrape all of the Obama Speeches from:
<http://www.americanrhetoric.com/barackobamaspeeches.htm>
- What if you wanted to learn something about the Chancellor's statements on the Budget:
<http://uwm.edu/budget/uwm-sources/chancellors-messages/>