ACCT 420: Textual analysis

Session 7

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
http://rmc.link/
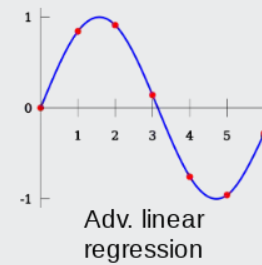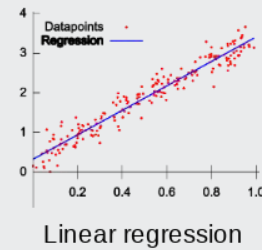
Front matter

Learning objectives
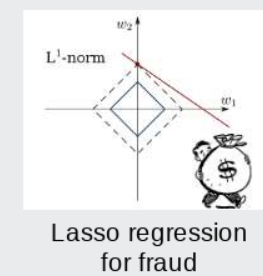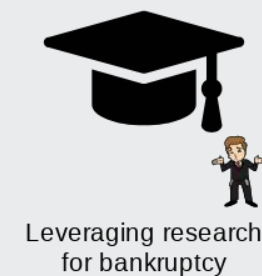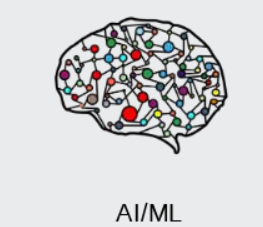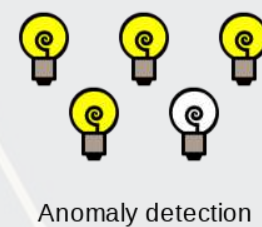

- **Theory:**
  - Natural Language Processing
- **Application:**
  - Analyzing a Citigroup annual report
- **Methodology:**
  - Text analysis
  - Machine learning

Datacamp

- Sentiment analysis in R the Tidy way
  - Just the first chapter is required
  - You are welcome to do more, of course
- I will generally follow the same "tidy text" principles as the Datacamp course does – the structure keeps things easy to manage
  - We will sometimes deviate to make use of certain libraries, which, while less tidy, make our work easier than the corresponding tidy-oriented packages (if they even exist!)

Textual data and textual analysis
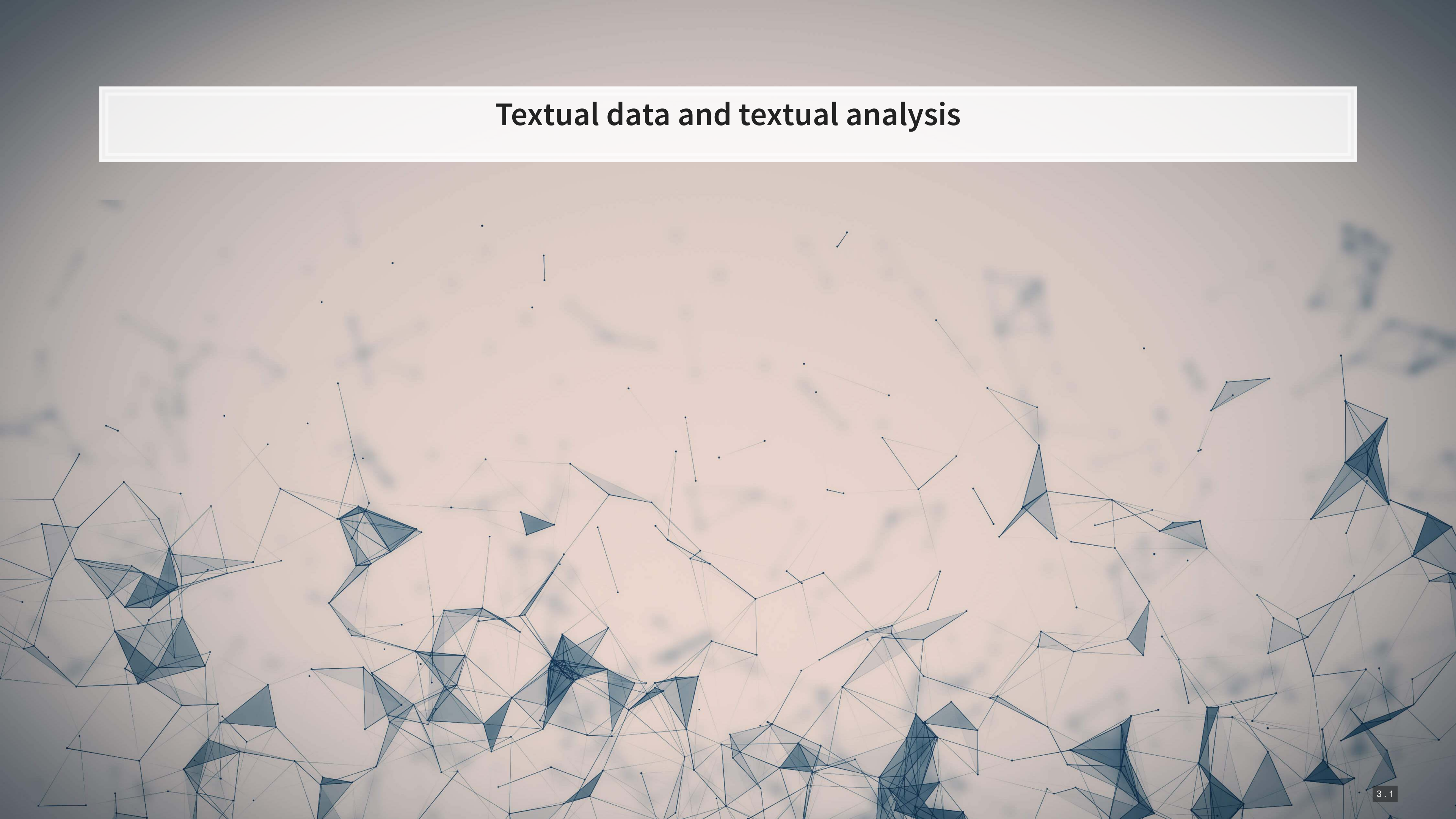
Review of Session 6

- Last session we saw that textual measures can help improve our fraud detection algorithm
- We actually looked at a bunch of textual measures:
  - Sentiment
  - Readability
  - Topic/content
- We didn't see how to make these though…
  - Instead, we had a nice premade dataset with everything already done

We'll get started on these today – sentiment and readability

We'll cover topic modeling next session

Why is textual analysis harder?

- Thus far, everything we've worked with is what is known as *structured data*
  - Structured data is numeric, nicely indexed, and easy to use
- Text data is *unstructured*
  - If we get an annual report with 200 pages of text…
    - Where is the information we want?
    - What do we want?
    - How do we crunch 200 pages into something that is…
      1. Manageable? (Structured)
      2. Meaningful?

This is what we will work on today, and we will revist some of this in the remaining class sessions

Structured data

- Our long or wide format data

Wide format

```
## # A tibble: 3 x 3
##   quarter level_3          value
##   <chr>   <chr>            <chr>
## 1 1995-Q1 Wholesale Trade 17
## 2 1995-Q1 Retail Trade    -18
## 3 1995-Q1 Accommodation   16
```

Long format

```
## # A tibble: 3 x 4
##   RegionID `1996-04` `1996-05` `1996-06`
##      <dbl>     <dbl>     <dbl>     <dbl>
## 1    84654    334200    335400    336500
## 2    90668    235700    236900    236700
## 3    91982    210400    212200    212200
```

The structure is given by the IDs, dates, and variables

Unstructured data

- Text
  - Open responses to question, reports, etc.
  - What it isn't:
    - `"JANUARY"`, `"ONE"`, `"FEMALE"`
    - Months, numbers
    - Anything with clear and concise categories
- Images
  - Satellite imagery
- Audio
  - Phone call recordings
- Video
  - Security camera footage

All of these require us to determine and *impose* structure

Some ideas of what we can do

1. Text extraction
   - Find all references to the CEO
   - Find if the company talked about global warming
   - Pull all telephone numbers or emails from a document
2. Text characteristics
   - How varied is the vocabulary?
   - Is it positive or negative (sentiment)
   - Is it written in a strong manner?
3. Text summarization or meaning
   - What is the content of the document?
   - What is the most important content of the document?
   - What other documents discuss similar issues?

Where might we encounter text data in business

1. Business contracts
2. Legal documents
3. Any paperwork
4. News
5. Customer reviews or feedback
   - Including transcription (call centers)
6. Consumer social media posts
7. Chatbots and AI assistants

Natural Language Processing (NLP)

- NLP is the subfield of computer science focused on analyzing large amounts of unstructured textual information
  - Much of the work builds from computer science, linguistics, and statistics
- Unstructured text actually has some structure derived from language itself
  - Word selection
  - Grammar
  - Phrases
  - Implicit orderings
- NLP utilizes this implicit structure to better understand textual data

NLP in everyday life

- Autocomplete of the next word in phone keyboards
  - Demo below from Google's blog
- Voice assistants like Google Assistant, Siri, Cortana, and Alexa
- Article suggestions on websites
- Search engine queries
- Email features like missing attachment detection
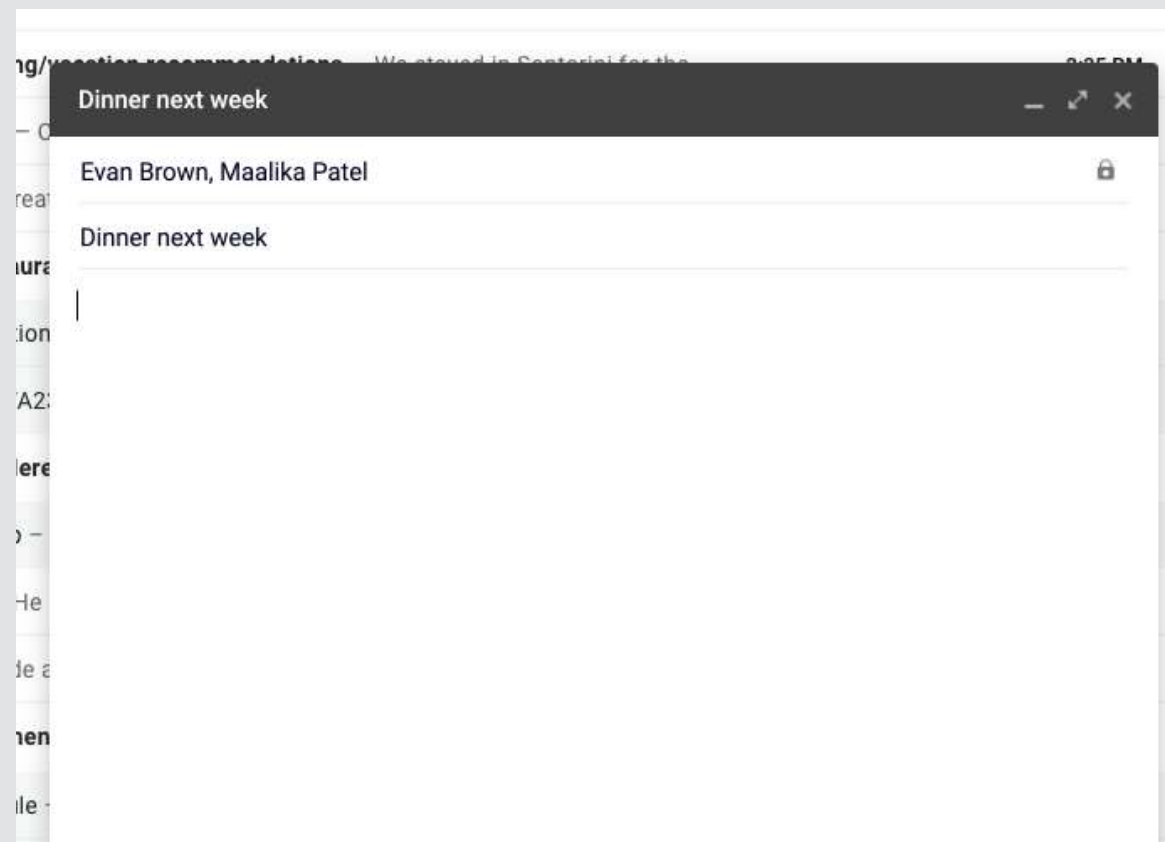
Hype Cycle for Artificial Intelligence, 2020

Case: How leveraging NLP helps call centers

- Natural Language Processing in Call Centres
- Short link: rmc.link/420class7

What are call centers using NLP for?

How does NLP help call centers with their business?

Consider

Where an we make use of NLP in business?

- We can use it for call centers
- We can make products out of it (like Google Duplex and other tech firms)
- Where else?

Working with 1 text file

Before we begin: Special characters

- Some characters in R have special meanings for string functions
    - `\ | ( ) [ { } ^ $ * + ? . !`
- To type a special character, we need to precede it with a `\`
    - Since `\` is a special character, we'll need to put `\` before `\`…
        - To type `$`, we would use `\\$`
- Also, some spacing characters have special symbols:
    - `\t` is tab
    - `\r` is newline (files from Macs)
    - `\r\n` is newline (files from Windows)
    - `\n` is newline (files from Unix, Linux, etc.)
    - You'll need to write `\\` to get the backslashes though

Loading in text data from files

- Use `read_file()` from `tidyverse`'s `readr` package to read in text data
- We'll use Citigroup's annual report from 2014
  - Note that there is a full text link at the bottom which is a .txt file
  - I will instead use a cleaner version derived from the linked file
    - The cleaner version can be made using the same techniques we will discuss today

```r
# Read text from a .txt file using read_file()
doc <- read_file("../../Data/0001104659-14-015152.txt")
# str_wrap is from stringr from tidyverse
cat(str_wrap(substring(doc,1,500), 80))
```

```
## UNITED STATES SECURITIES AND EXCHANGE COMMISSION WASHINGTON, D.C. 20549 FORM
## 10-K ANNUAL REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE
## ACT OF 1934 For the fiscal year ended December 31, 2013 Commission file number
## 1-9924 Citigroup Inc. (Exact name of registrant as specified in its charter)
## Securities registered pursuant to Section 12(b) of the Act: See Exhibit 99.01
## Securities registered pursuant to Section 12(g) of the Act: none Indicate by
## check mark if the registrant is a
```

Loading from other file types

- Ideally you have a .txt file already – such files are generally just the text of the documents
- Other common file types:
  - HTML files (particularly common from web data)
    - You can load it as a text file – just note that there are html tags embedded in it
      - Things like `<a>`, `<table>`, `<img>`, etc.
    - You can load from a URL using `httr` or `RCurl`
    - In R, you can use `XML` or `rvest` to parse out specific pieces of html files
    - If you use python, use lxml or BeautifulSoup 4 (bs4) to quickly turn these into structured documents
    - In R, you can process JSON data using `jsonlite`

Loading from other file types

- Ideally you have a .txt file already – such files are generally just the text of the documents
- Other common file types:
  - PDF files
    - Use `pdftools` to extract text into a vector of pages of text
    - Use `tabulizer` to extract tables straight from PDF files!
      - This is very painful to code by hand without this package
      - The package itself is a bit difficult to install, requiring Java and `rJava`, though

Example using html

```r
library(httr)
library(XML)

httpResponse <- GET('https://coinmarketcap.com/currencies/ethereum/')
html = httr::content(httpResponse, "text")
paste0('...', str_wrap(substring(html, 135418, 135480), 80), '...')
```

```
## [1] "...ll{margin-right:8px;}/*!sc*/ .jskEGI .namePillPrimary{backgroun..."
```

```r
xpath <- '//*[@id="__next"]/div/div[1]/div[2]/div/div[1]/div[2]/div/div[2]/div[1]/div/text()'
hdoc = htmlParse(html, asText=TRUE)  # from XML
price <- xpathSApply(hdoc, xpath, xmlValue)
print(paste0("Ethereum was priced at $", price,
             " when these slides were compiled"))
```

```
## [1] "Ethereum was priced at $$3,065.38 when these slides were compiled"
```

Automating crypto pricing in a document

```r
# The actual version I use (with caching to avoid repeated lookups) is in the appendix
cryptoMC <- function(name) {
  httpResponse <- GET(paste('https://coinmarketcap.com/currencies/',name,'/',sep=''))
  html = httr::content(httpResponse, "text")
  xpath <- '//*[@id="__next"]/div/div[1]/div[2]/div/div[1]/div[2]/div/div[2]/div[1]/div/text()'
  hdoc = htmlParse(html, asText=TRUE)
  plain.text <- xpathSApply(hdoc, xpath, xmlValue)
  plain.text
}
```

```r
paste("Ethereum was priced at", cryptoMC("ethereum"))
```

```
## [1] "Ethereum was priced at $3,065.38"
```

```r
paste("Litecoin was priced at", cryptoMC("litecoin"))
```

```
## [1] "Litecoin was priced at $151.13"
```

Basic text functions in R

- Subsetting text
- Transformation
  - Changing case
  - Adding or combining text
  - Replacing text
  - Breaking text apart
- Finding text

We will cover these using `stringr` as opposed to base R – `stringr`'s commands are much more consistent

- Every function in `stringr` can take a vector of strings for the first argument, which is *tidy*

Subsetting text

- Base R: Use `substr()` or `substring()`
- `stringr`: use `str_sub()`
  - First argument is a vector of strings
  - Second argument is the starting position (inclusive)
  - Third argument is that ending position (inclusive)

```r
cat(str_wrap(str_sub(doc, 9896, 9929), 80))
```

```
## Citis net income was $13.5 billion
```

```r
cat(str_wrap(str_sub(doc, 28900,29052), 80))
```

```
## Net income decreased 14%, mainly driven by lower revenues and lower loan loss
## reserve releases, partially offset by lower net credit losses and expenses.
```

Transforming text

- Commonly used functions:
  - `tolower()` or `str_to_lower()`: make the text lowercase
  - `toupper()` or `str_to_upper()`: MAKE THE TEXT UPPERCASE
  - `str_to_title()`: Make the Text Titlecase
- `paste()` to combine text
  - It puts spaces between by default
    - You can change this with the `sep=` option
  - If everything to combine is in 1 vector, use `collapse=` with the desired separator
  - `paste0()` is paste with `sep=""`

Examples: Case

```r
sentence <- str_sub(doc, 9896, 9929)
str_to_lower(sentence)
```

```
## [1] "citis net income was $13.5 billion"
```

```r
str_to_upper(sentence)
```

```
## [1] "CITIS NET INCOME WAS $13.5 BILLION"
```

```r
str_to_title(sentence)
```

```
## [1] "Citis Net Income Was $13.5 Billion"
```

- The `str_` prefixed functions support non-English languages as well

```r
# You can run this in an R terminal! (It doesn't work in Rmarkdown though)
str_to_upper("Citis net income was $13.5 billion", locale='tr')  # Turkish
```

Examples: `paste`

```r
# board is a list of director names
# titles is a list of the director's titles
paste(board, titles, sep=", ")
```

```
##  [1] "Michael L. Corbat, CEO"
##  [2] "Michael E. O'Neill, Chairman"
##  [3] "Anthony M. Santomero, Former president, Fed (Philidelphia)"
##  [4] "William S. Thompson, Jr., CEO, Retired, PIMCO"
##  [5] "Duncan P. Hennes, Co-Founder/Partner, Atrevida Partners"
##  [6] "Gary M. Reiner, Operating Partner, General Atlantic"
##  [7] "Joan E. Spero, Senior Research Scholar, Columbia University"
##  [8] "James S. Turley, Former Chairman & CEO, E&Y"
##  [9] "Franz B. Humer, Chairman, Roche"
## [10] "Judith Rodin, President, Rockefeller Foundation"
## [11] "Robert L. Ryan, CFO, Retired, Medtronic"
## [12] "Diana L. Taylor, MD, Wolfensohn Fund Management"
## [13] "Ernesto Zedillo Ponce de Leon, Professor, Yale University"
## [14] "Robert L. Joss, Professor/Dean Emeritus, Stanford GSB"
```

```r
cat(str_wrap(paste0("Citi's board consists of: ",
                    paste(board[1:length(board)-1], collapse=", "),
                    ", and ", board[length(board)], "."), 80))
```

```
## Citi's board consists of: Michael L. Corbat, Michael E. O'Neill, Anthony M.
## Santomero, William S. Thompson, Jr., Duncan P. Hennes, Gary M. Reiner, Joan E.
## Spero, James S. Turley, Franz B. Humer, Judith Rodin, Robert L. Ryan, Diana L.
## Taylor, Ernesto Zedillo Ponce de Leon, and Robert L. Joss.
```

Transforming text

- Replace text with `str_replace_all()`
  - First argument is text data
  - Second argument is what you want to remove
  - Third argument is the replacement
- If you only want to replace the first occurrence, use `str_replace()` instead

```r
sentence
```

```
## [1] "Citis net income was $13.5 billion"
```

```r
str_replace_all(sentence, "\\$13.5", "over $10")
```

```
## [1] "Citis net income was over $10 billion"
```

Transforming text

- Split text using `str_split()`
  - This function returns a list of vectors!
    - This is because it will turn every string passed to it into a vector, and R can't have a vector of vectors
  - `[[1]]` can extract the first vector
- You can also limit the number of splits using `n=`
  - A bit more elegant solution is using `str_split_fixed()` with `n=`
    - Returns a character matrix (nicer than a list)

Example: Splitting text

```r
paragraphs <- str_split(doc, '\n')[[1]]

# number of paragraphs
length(paragraphs)
```

```
## [1] 206
```

```r
# Last paragraph
cat(str_wrap(paragraphs[206], 80))
```

```
## The total amount of securities authorized pursuant to any instrument defining
## rights of holders of long-term debt of the Company does not exceed 10% of the
## total assets of the Company and its consolidated subsidiaries. The Company
## will furnish copies of any such instrument to the SEC upon request. Copies of
## any of the exhibits referred to above will be furnished at a cost of $0.25 per
## page (although no charge will be made for the 2013 Annual Report on Form 10-
## K) to security holders who make written request to Citigroup Inc., Corporate
## Governance, 153 East 53 rd Street, 19 th Floor, New York, New York 10022. *
## Denotes a management contract or compensatory plan or arrangement. + Filed
## herewith.
```

Finding phrases in text

- How did I find the previous examples?

```r
str_locate_all(str_to_lower(doc), "net income")
```

```
## [[1]]
##       start    end
##  [1,]   8508   8517
##  [2,]   9902   9911
##  [3,]  16549  16558
##  [4,]  17562  17571
##  [5,]  28900  28909
##  [6,]  32197  32206
##  [7,]  35077  35086
##  [8,]  37252  37261
##  [9,]  40187  40196
## [10,]  43257  43266
## [11,]  45345  45354
## [12,]  47618  47627
## [13,]  51865  51874
## [14,]  51953  51962
## [15,]  52663  52672
## [16,]  52748  52757
## [17,]  54970  54979
## [18,]  58817  58826
## [19,]  96022  96031
```

Finding phrases in text

- 4 primary functions:
  1. `str_detect()`: Reports `TRUE` or `FALSE` for the presence of a string in the text
  2. `str_count()`: Reports the number of times a string is in the text
  3. `str_locate()`: Reports the first location of a string in the text
     - `str_locate_all()`: Reports every location as a list of matrices
  4. `str_extract()`: Reports the matched phrases
- All take a character vector as the first argument, and something to match for the second argument

Example: Finding phrases

- How many paragraphs mention net income in any case?

```
x <- str_detect(str_to_lower(paragraphs), "net income")
x[1:10]
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
```

```
sum(x)
```

```
## [1] 13
```

- What is the most net income is mentioned in any paragraph

```
x <- str_count(str_to_lower(paragraphs), "net income")
x[1:10]
```

```
##  [1] 0 0 0 0 0 4 0 0 2 2
```

```
max(x)
```

```
## [1] 4
```

Example: Finding phrases

- Where is net income first mentioned in the document?

```r
str_locate(str_to_lower(doc), "net income")
```

```
##      start  end
## [1,]  8508 8517
```

- First mention of net income
  - This function may look useless now, but it'll be on of the most useful later

```r
str_extract(str_to_lower(doc), "net income")
```

```
## [1] "net income"
```

R Practice

- Text data is already loaded, as if it was loaded using `read_file()`
- Try:
  - Subsetting the text data
  - Transforming the text data
    - To all upper case
    - Replacing a phrase
  - Finding specific text in the document
- Do exercises 1 through 3 in today's practice file
  - R Practice
  - Shortlink: rmc.link/420r7

Pattern matching

Finding *patterns* in the text (regex)

- Regular expressions, aka regex or regexp, are ways of finding patterns in text
- This means that instead of looking for a specific phrase, we can match a set of phrases
- Most of the functions we discussed accept regexes for matching
  - `str_replace()`, `str_split()`, `str_detect()`, `str_count()`, `str_locate()`, and `str_extract()`, plus their variants
- This is why `str_extract()` is so great – we can extract anything from a document with it!

Regex example

- Breaking down an email
  1. A local name
  2. An @ sign
  3. A domain, which will have a . in it
- Local names can have many different characters in them
  - Match it with `[:graph:]+`
- The domain is pretty restrictive, generally just alphanumeric and .
  - There can be multiple . though
  - Match it with `[:alnum:]+\\.[.[:alnum:]]+`

```r
# Extract all emails from the annual report
str_extract_all(doc,'[:graph:]+@[:alnum:]+\\.[.[:alnum:]]+')
```

```
## [[1]]
## [1] "shareholder@computershare.com" "shareholder@computershare.com"
## [3] "docserve@citi.com"             "shareholderrelations@citi.com"
```

Breaking down the example

- @ was itself – it isn't a special character in strings in R
- \\. is just a period – we need to escape . because it is special
- Anything in brackets with colons, [: :], is a set of characters
  - [:graph:] means any letter, number, or punctuation
  - [:alnum:] means any letter or number
- + is used to indicate that we want 1 or more of the preceding element – as many as it can match
  - [:graph:]+ meant "Give us every letter, number, and punctuation you can, but make sure there is at least 1."
- Brackets with no colons, [ ], ask for anything inside
  - [.[:alnum:]]+ meant "Give us every letter, number, and . you can, but make sure there is at least 1."

Breaking down the example

- Let's examine the output `shareholder@computershare.com`
- Our regex was `[:graph:]+@[:alnum:]+\\.[.[:alnum:]]+`
- Matching regex components to output:
  - `[:graph:]+` ⟹ `shareholder`
  - `@` ⟹ `@`
  - `[:alnum:]+` ⟹ `computershare`
  - `\\.` ⟹ `.`
  - `[.[:alnum:]]+` ⟹ `com`

Useful regex components: Content

- There's a nice cheat sheet here
  - More detailed documentation here
- Matching collections of characters
  - `.` matches everything
  - `[:alpha:]` matches all letters
  - `[:lower:]` matches all lowercase letters
  - `[:upper:]` matches all UPPERCASE letters
  - `[:digit:]` matches all numbers 0 through 9
  - `[:alnum:]` matches all letters and numbers
  - `[:punct:]` matches all punctuation
  - `[:graph:]` matches all letters, numbers, and punctuation
  - `[:space:]` or `\s` match ANY whitespace
    - `\S` is the exact opposite
  - `[:blank:]` matches whitespace except newlines

Example: Regex content

```r
text <- c("abcde", 'ABCDE', '12345', '!?!?.', 'ABC123?', "With space", "New\nline")
html_df(data.frame(
  text=text,
  alpha=str_detect(text,'[:alpha:]'),
  lower=str_detect(text,'[:lower:]'),
  upper=str_detect(text,'[:upper:]'),
  digit=str_detect(text,'[:digit:]'),
  alnum=str_detect(text,'[:alnum:]')
))
```

text	alpha	lower	upper	digit	alnum
abcde	TRUE	TRUE	FALSE	FALSE	TRUE
ABCDE	TRUE	FALSE	TRUE	FALSE	TRUE
12345	FALSE	FALSE	FALSE	TRUE	TRUE
!?!?.	FALSE	FALSE	FALSE	FALSE	FALSE
ABC123?	TRUE	FALSE	TRUE	TRUE	TRUE
With space	TRUE	TRUE	TRUE	FALSE	TRUE
New line	TRUE	TRUE	TRUE	FALSE	TRUE

Example: Regex content

```
text <- c("abcde", 'ABCDE', '12345', '!?!?.', 'ABC123?', "With space", "New\nline")
html_df(data.frame(
  text=text,
  punct=str_detect(text,'[:punct:]'),
  graph=str_detect(text,'[:graph:]'),
  space=str_detect(text,'[:space:]'),
  blank=str_detect(text,'[:blank:]'),
  period=str_detect(text,'.')
))
```

text	punct	graph	space	blank	period
abcde	FALSE	TRUE	FALSE	FALSE	TRUE
ABCDE	FALSE	TRUE	FALSE	FALSE	TRUE
12345	FALSE	TRUE	FALSE	FALSE	TRUE
!?!?.	TRUE	TRUE	FALSE	FALSE	TRUE
ABC123?	TRUE	TRUE	FALSE	FALSE	TRUE
With space	FALSE	TRUE	TRUE	TRUE	TRUE
New line	FALSE	TRUE	TRUE	FALSE	TRUE

Useful regex components: Form

- `[  ]` can be used to create a class of characters to look for
  - `[abc]` matches anything that is `a`, `b`, `c`
- `[^  ]` can be used to create a class of everything else
  - `[^abc]` matches anything that isn't `a`, `b`, or `c`
- Quantity, where `x` is some element
  - `x?` looks for 0 or 1 of `x`
  - `x*` looks for 0 or more of `x`
  - `x+` looks for 1 or more of `x`
  - `x{n}` looks for `n` (a number) of `x`
  - `x{n,  }` looks for at least `n` of `x`
  - `x{n,m}` looks for at least `n` and at most `m` of `x`
- Lazy operators
  - Regexes always prefer the longest match by default
  - Append `?` to any quantity operator to make it prefer the shortest match possible

Useful regex components: Form

- Position
  - `^` indicates the start of the string
  - `$` indicates the end of the string
- Grouping
  - `(  )` can be used to group components
  - `|` can be used within groups as a logical *or*
  - Groups can be referenced later using the position of the group within the regex
    - `\\1` refers to the first group
    - `\\2` refers to the second group
    - …

Example: Regexes on real estate firm names

```
# Real estate firm names with 3 vowels in a row
str_subset(RE_names, '[AEIOU]{3}')
```

```
## [1] "STADLAUER MALZFABRIK"      "JOAO FORTES ENGENHARIA SA"
```

```
# Real estate firm names with no vowels
str_subset(RE_names, '^[^AEIOU]+$')
```

```
## [1] "FGP LTD"      "MBK PCL"      "MYP LTD"      "MCT BHD"      "R T C L LTD"
```

```
# Real estate firm names with at least 12 vowels
str_subset(RE_names, '([^AEIOU]*[AEIOU]){11,}')
```

```
## [1] "INTERNATIONAL ENTERTAINMENT"   "PREMIERE HORIZON ALLIANCE"
## [3] "JOAO FORTES ENGENHARIA SA"     "OVERSEAS CHINESE TOWN (ASIA)"
## [5] "COOPERATIVE CONSTRUCTION CO"   "FRANCE TOURISME IMMOBILIER"
## [7] "BONEI HATICHON CIVIL ENGINE"
```

```
# Real estate firm names with a repeated 4 letter pattern
str_subset(RE_names, '([[:upper:]]{4}).*\\1')
```

```
## [1] "INTERNATIONAL ENTERTAINMENT"   "CHONG HONG CONSTRUCTION CO"
## [3] "ZHONGHONG HOLDING CO LTD"      "DEUTSCHE GEOTHERMISCHE IMMOB"
```

Why is regex so important?

- Regex can be used to match anything in text
    - Simple things like phone numbers
    - More complex things like addresses
- It can be used to parse through large markup documents
    - HTML, XML, LaTeX, etc.
- Very good for validating the format of text
    - For birthday in the format YYYYMMDD, you could validate with:
        - YYYY: `[12][90][:digit:][:digit:]`
        - MM: `[01][:digit:]`
        - DD: `[0123][:digit:]`

Cavaet: Regexes are generally slow. If you can code something to avoid them, that is often better. But often that may be infeasible.

Some extras

- While the `str_*()` functions use regex by default, they actually have four modes
  1. You can specify a regex normally
     - Or you can use `regex()` to construct more customized ones, such as regexes that operate by line in a string
  2. You can specify an exact string to match using `fixed()` – fast but fragile
  3. You can specify an exact string to match using `coll()` – slow but robust; recognizes characters that are equivalent
     - Important when dealing with non-English words, since certain characters can be encoded in multiple ways
  4. You can ask for boundaries with `boundary()` such as words, using `boundary("word")`

Expanding usage

- Anything covered so far can be used for text in data
  - Ex.: Firm names or addresses in Compustat

```r
# Compustat firm names example
df_RE_names <- df_RE %>%
  group_by(isin) %>%
  slice(1) %>%
  mutate(SG_in_name = str_detect(conm, "(SG|SINGAPORE)"),
         name_length = str_length(conm),
         SG_firm = ifelse(fic=="SGP",1,0)) %>%
  ungroup()

df_RE_names %>%
  group_by(SG_firm) %>%
  mutate(pct_SG = mean(SG_in_name) * 100) %>%
  slice(1) %>%
  ungroup() %>%
  select(SG_firm, pct_SG)
```

```
## # A tibble: 2 x 2
##   SG_firm pct_SG
##     <dbl>  <dbl>
## 1       0  0.369
## 2       1  4.76
```

Expanding usage

```r
library(DT)
df_RE_names %>%
  group_by(fic) %>%
  mutate(avg_name_length = mean(name_length)) %>%
  slice(1) %>%
  ungroup() %>%
  select(fic, avg_name_length) %>%
  arrange(desc(avg_name_length), fic) %>%
  datatable(options = list(pageLength = 5))
```

Show ▾ entries   Search:

	fic	avg_name_length
1	TUR	27
2	VNM	25.5
3	EGY	25
4	CHN	24.5714285714286
5	ISR	24.3333333333333

Showing 1 to 5 of 41 entries   Previous 1 2 3 4 5 … 9 Next

R Practice 2

- This practice explores the previously used practice data using regular expressions for various purposes
- Do exercises 4 and 5 in today's practice file
  - R Practice
  - Shortlink: rmc.link/420r7

Readability and Sentiment

Readability

- Thanks to the `quanteda` package, readability is very easy to calculate in R
  - Use the `textstat_readability()` function
- There are many readability measures, however
  - Flesch Kinkaid grade level: A measure of readability developed for the U.S. Navy to ensure manuals were written at a level any 15 year old should be able to understand
  - Fog: A grade level index that was commonly used in business and publishing
  - Coleman-Liau: An index with a unique calculation method, relying only on character counts

Readability: Flesch Kincaid

$$0.39 \left(\frac{\#\ words}{\#\ sentences} \right) + 11.8 \left(\frac{\#\ syllables}{\#\ words} \right) - 15.59$$

- An approximate grade level required for reading a document
  - *Lower is more readable*
  - A JC or poly graduate should read at a level of 12
    - New York Times articles are usually around 13
  - A Bachelor's degree could be necessary for anything 16 or above

```r
library(quanteda)
```

```
## Warning: package 'quanteda' was built under R version 4.1.1
```

```r
library(quanteda.textstats)
textstat_readability(doc, "Flesch.Kincaid")
```

```
##   document Flesch.Kincaid
## 1    text1       17.73412
```

Readability: Fog

$$[Mean(Words\ per\ sentence)+$$
$$(\%\ of\ words\ > 3\ syllables)] \times 0.4$$

- An approximate grade level required for reading a document
  - *Lower is more readable*

```r
textstat_readability(doc, "FOG")
```

```
##   document      FOG
## 1    text1 21.89503
```

Readability: Coleman-Liau

$$5.88 \left(\frac{\# \; letters}{\# \; words} \right) - 29.6 \left(\frac{\# \; sentences}{\# \; words} \right) - 15.8$$

- An approximate grade level required for reading a document
  - *Lower is more readable*

```
textstat_readability(doc, "Coleman.Liau.short")
```

```
##   document Coleman.Liau.short
## 1    text1          14.96773
```

Converting text to words

- Tidy text is when you have one *token* per document per row, in a data frame
- *Token* is the unit of text you are interested in
  - Words: "New"
  - Phrases: "New York Times"
  - Sentences: "The New York Times is a publication."
  - etc.
- The `tidytext` package can handle this conversion for us!
  - Use the `unnest_tokens()` function
  - Note: it also converts to lowercase. Use the option `to_lower=FALSE` to avoid this if needed

```r
# Example of "tokenizing"
library(tidytext)
df_doc <- data.frame(ID=c("0001104659-14-015152"), text=c(doc)) %>%
  unnest_tokens(word, text)
# word is the name for the new column
# text is the name of the string column in the input data
```

The details

```
html_df(head(df_doc))
```

ID	word
0001104659-14-015152	united
0001104659-14-015152	states
0001104659-14-015152	securities
0001104659-14-015152	and
0001104659-14-015152	exchange
0001104659-14-015152	commission

The details

- `tidytext` uses the `tokenizers` package in the backend to do the conversion
  - You can call that package directly instead if you want to
- Available tokenizers include: (specify with `token=`)
  - "word": The default, individual words
  - "ngram": Collections of words (default of 2, specify with `n=`)
  - A few other less commonly used tokenizers

Word case

- Why convert to lowercase?
- How much of a difference is there between "The" and "the"?
  - "Singapore" and "singapore" – still not much difference
  - Only words like "new" versus "New" matter
    - "New York" versus "new yorkshire terrier"
- Benefit: We get rid of a bunch of distinct words!
  - Helps with *the curse of dimensionality*

The Curse of dimensionality

- There are a lot of words
- A LOT OF WORDS
- At least 171,476 according to Oxford Dictionary
- What happens if we make a matrix of words per document?

Stopwords

- Stopwords – words we remove because they have little content
  - the, a, an, and, …
- Also helps with our curse a bit – removes the words entirely
- We'll use the `stopword` package to remove stopwords

```r
# get a list of stopwords
stop_en <- stopwords::stopwords("english")   # Snowball English
paste0(length(stop_en), " words: ", paste(stop_en[1:5], collapse=", "))
```

```
## [1] "175 words: i, me, my, myself, we"
```

```r
stop_SMART <- stopwords::stopwords(source="smart")   # SMART English
paste0(length(stop_SMART), " words: ", paste(stop_SMART[1:5], collapse=", "))
```

```
## [1] "571 words: a, a's, able, about, above"
```

```r
stop_fr <- stopwords::stopwords("french")   # Snowball French
paste0(length(stop_fr), " words: ", paste(stop_fr[1:5], collapse=", "))
```

```
## [1] "164 words: au, aux, avec, ce, ces"
```

Applying stopwords to a corpus

- When we have a tidy set of text, we can just use `dplyr` for this!
  - `dplyr`'s `anti_join()` function is like a merge, but where all matches are deleted

```r
df_doc_stop <- df_doc %>%
  anti_join(data.frame(word=stop_SMART))
```

```
## Joining, by = "word"
```

```r
nrow(df_doc)
```

```
## [1] 128728
```

```r
nrow(df_doc_stop)
```

```
## [1] 74985
```

Converting to term frequency

```r
terms <- df_doc_stop %>%
  count(ID, word, sort=TRUE) %>%
  ungroup()
total_terms <- terms %>%
  group_by(ID) %>%
  summarize(total = sum(n))
tf <- left_join(terms, total_terms) %>% mutate(tf=n/total)
tf
```

```
##                    ID            word    n total          tf
## 1     0001104659-14-015152        citi 826 74985 0.011015536
## 2     0001104659-14-015152        2013 743 74985 0.009908648
## 3     0001104659-14-015152      credit 704 74985 0.009388544
## 4     0001104659-14-015152       citis 660 74985 0.008801760
## 5     0001104659-14-015152        risk 624 74985 0.008321664
## 6     0001104659-14-015152    december 523 74985 0.006974728
## 7     0001104659-14-015152   financial 513 74985 0.006841368
## 8     0001104659-14-015152          31 505 74985 0.006734680
## 9     0001104659-14-015152       loans 495 74985 0.006601320
## 10    0001104659-14-015152      assets 488 74985 0.006507968
## 11    0001104659-14-015152        fair 453 74985 0.006041208
## 12    0001104659-14-015152  securities 440 74985 0.005867840
## 13    0001104659-14-015152      billion 435 74985 0.005801160
## 14    0001104659-14-015152    citigroup 420 74985 0.005601120
## 15    0001104659-14-015152        2012 412 74985 0.005494432
## 16    0001104659-14-015152         u.s 390 74985 0.005201040
## 17    0001104659-14-015152    interest 373 74985 0.004974328
## 18    0001104659-14-015152     company 371 74985 0.004947656
## 19    0001104659-14-015152         net 324 74985 0.004320864
## 20    0001104659-14-015152     related 323 74985 0.004307528
```

Sentiment

- Sentiment works similarly to stopwords, except we are identifying words with specific, useful meanings
  - We can grab off-the-shelf sentiment measures using `get_sentiments()` from `tidytext`

```r
# Need to install the `textdata` package for the below
get_sentiments("afinn") %>%
  group_by(value) %>%
  slice(1) %>%
  ungroup()
```

```
## # A tibble: 11 x 2
##    word         value
##    <chr>        <dbl>
##  1 bastard         -5
##  2 ass             -4
##  3 abhor           -3
##  4 abandon         -2
##  5 absentee        -1
##  6 some kind        0
##  7 aboard           1
##  8 abilities        2
##  9 admire           3
## 10 amazing          4
## 11 breathtaking     5
```

```r
get_sentiments("bing") %>%
  group_by(sentiment) %>%
  slice(1) %>%
  ungroup()
```

```
## # A tibble: 2 x 2
##    word    sentiment
##    <chr>   <chr>
## 1 2-faces negative
## 2 abound  positive
```

Sentiment

```r
get_sentiments("nrc") %>%
  group_by(sentiment) %>%
  slice(1) %>%
  ungroup()
```

```
## # A tibble: 10 x 2
##    word       sentiment
##    <chr>      <chr>
##  1 abandoned  anger
##  2 abundance  anticipation
##  3 aberration disgust
##  4 abandon    fear
##  5 absolution joy
##  6 abandon    negative
##  7 abba       positive
##  8 abandon    sadness
##  9 abandonment surprise
## 10 abacus     trust
```

Loughran & McDonald dictionary – finance specific, targeted at annual reports

```r
get_sentiments("loughran") %>%
  group_by(sentiment) %>%
  slice(1) %>%
  ungroup()
```

```
## # A tibble: 6 x 2
##    word          sentiment
##    <chr>         <chr>
## 1 abide         constraining
## 2 abovementioned litigious
## 3 abandon       negative
## 4 able          positive
## 5 aegis         superfluous
## 6 abeyance      uncertainty
```

Merging in sentiment data

```r
tf_sent <- tf %>% left_join(get_sentiments("loughran"))
```

```
## Joining, by = "word"
```

```r
tf_sent[1:5,]
```

```
##                       ID   word   n total          tf   sentiment
## 1 0001104659-14-015152   citi 826 74985 0.011015536        <NA>
## 2 0001104659-14-015152   2013 743 74985 0.009908648        <NA>
## 3 0001104659-14-015152 credit 704 74985 0.009388544        <NA>
## 4 0001104659-14-015152  citis 660 74985 0.008801760        <NA>
## 5 0001104659-14-015152   risk 624 74985 0.008321664 uncertainty
```

```r
tf_sent[!is.na(tf_sent$sentiment),][1:5,]
```

```
##                        ID        word   n total          tf   sentiment
## 5  0001104659-14-015152        risk 624 74985 0.008321664 uncertainty
## 28 0001104659-14-015152        loss 267 74985 0.003560712    negative
## 29 0001104659-14-015152      losses 265 74985 0.003534040    negative
## 36 0001104659-14-015152 approximately 232 74985 0.003093952 uncertainty
## 37 0001104659-14-015152   regulatory 216 74985 0.002880576   litigious
```
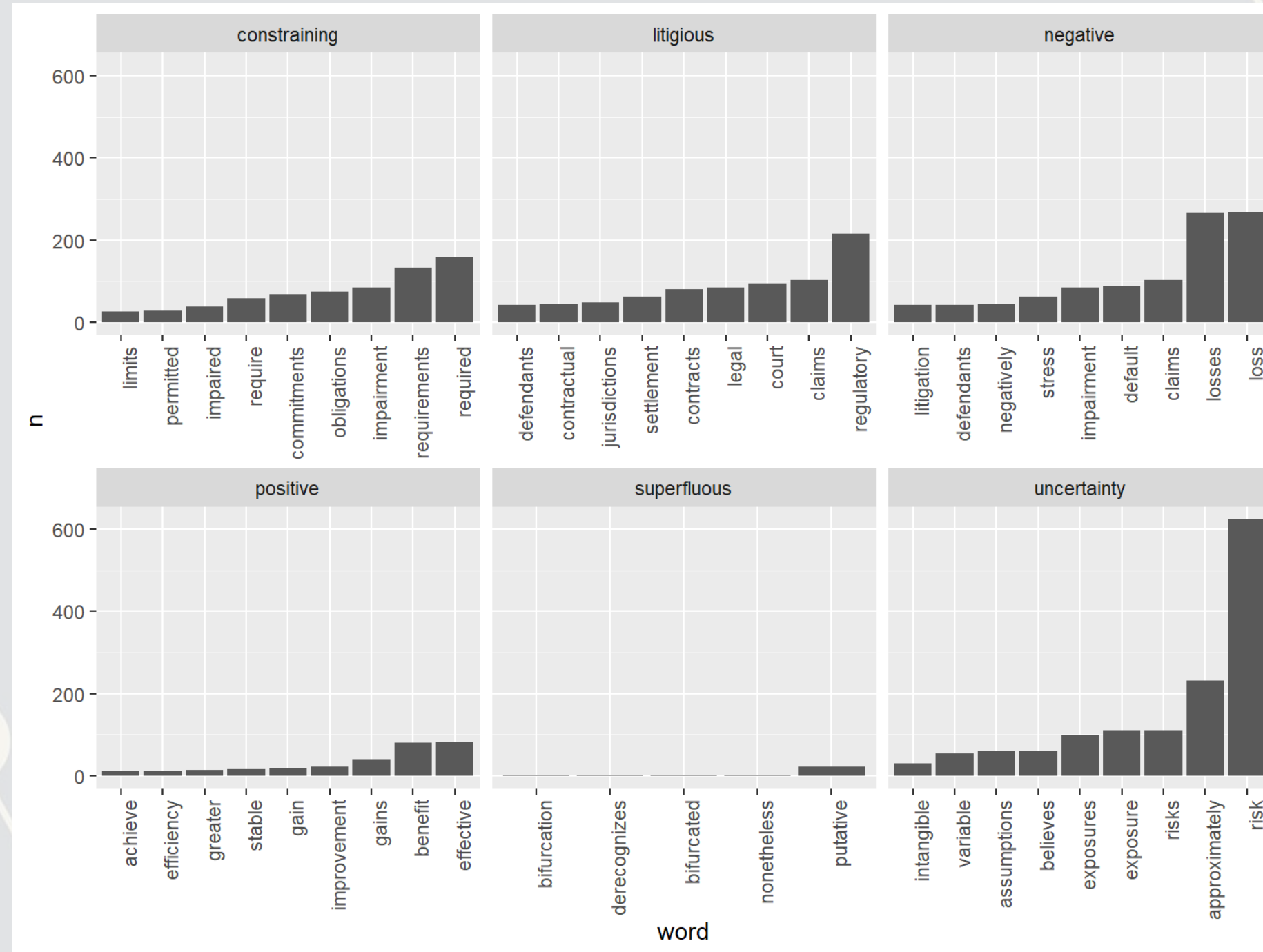
Summarizing document sentiment

```r
tf_sent %>%
  spread(sentiment, tf, fill=0) %>%
  select(constraining, litigious, negative, positive, superfluous, uncertainty) %>%
  colSums()
```

```
## constraining    litigious     negative     positive  superfluous  uncertainty
##   0.013242649  0.020750817  0.034780289  0.007054744  0.000373408  0.025325065
```

visualizing sentiment

Visualizing a document as a word cloud

- `quanteda` provides `textplot_wordcloud()`
- `cast_dfm()` converts tidy term frequencies to Quanteda
- There are also the `wordcloud` and `wordcloud2` packages for this

```
library(quanteda.textplots)
corp <- cast_dfm(tf, ID, word, n)
textplot_wordcloud(dfm(corp), color = RColorBrewer::brewer.pal(9, "Set1"))
```

Another reason to use stopwords

- Without removing stopwords, the word cloud shows almost nothing useful

```
corp_no_stop <- cast_dfm(tf_no_stop, ID, word, n)
textplot_wordcloud(dfm(corp_no_stop), color = RColorBrewer::brewer.pal(9, "Set1"))
```

R Practice 3

- Using the same data as before, we will explore
  - Readability
  - Sentiment
  - Word clouds
- Note: Due to missing packages, you will need to run the code in RStudio, not in the DataCamp light console
- Do exercises 6 through 8 in today's practice file
  - R Practice
  - Shortlink: rmc.link/420r7

End matter

For next week

- For next week:
  - Finish the third assignment
    - Submit on eLearn
  - Datacamp
    - Do the assigned chapter on text analysis
  - Start on the group project

Packages used for these slides

- `httr`
- `kableExtra`
- `knitr`
- `magrittr`
- `quanteda`
- `RColorBrewer`
- `readtext`
- `revealjs`
- `tidytext`
- `tidyverse`
  - `dplyr`, `readr`, `stringr`
- `XML`

Custom code

```r
library(knitr)
library(kableExtra)
html_df <- function(text, cols=NULL, col1=FALSE, full=F) {
  if(!length(cols)) {
    cols=colnames(text)
  }
  if(!col1) {
    kable(text,"html", col.names = cols, align = c("l",rep('c',length(cols)-1))) %>%
      kable_styling(bootstrap_options = c("striped","hover"), full_width=full)
  } else {
    kable(text,"html", col.names = cols, align = c("l",rep('c',length(cols)-1))) %>%
      kable_styling(bootstrap_options = c("striped","hover"), full_width=full) %>%
      column_spec(1,bold=T)
  }
}
```

```r
cryptoMC <- function(name) {
  if (exists(name)) {
    get(name)
  } else{
  html <- getURL(paste('https://coinmarketcap.com/currencies/',name,'/',sep=''))
  xpath <- '//*[@id="__next"]/div[1]/div[2]/div[1]/div[2]/div[1]/div/div[2]/span[1]/span[1]/text()'
  doc = htmlParse(html, asText=TRUE)
  plain.text <- xpathSApply(doc, xpath, xmlValue)
  assign(name, gsub("\n","",gsub(" ", "", paste(plain.text, collapse = ""), fixed = TRUE), fixed = TRUE),envir = .GlobalEnv)
  get(name)
  }
}
```

```r
# Create a plot of the top words by sentiment
tf_sent %>%
  filter(!is.na(sentiment)) %>%
  group_by(sentiment) %>%
  arrange(desc(n)) %>%
  mutate(row = row_number()) %>%
  filter(row < 10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(y=n, x=word)) + geom_col() + theme(axis.text.x = element_text(angle=90, hjust=1)) +
  facet_wrap(~sentiment, ncol=3, scales="free_x")
```