ACCT 420: Textual analysis

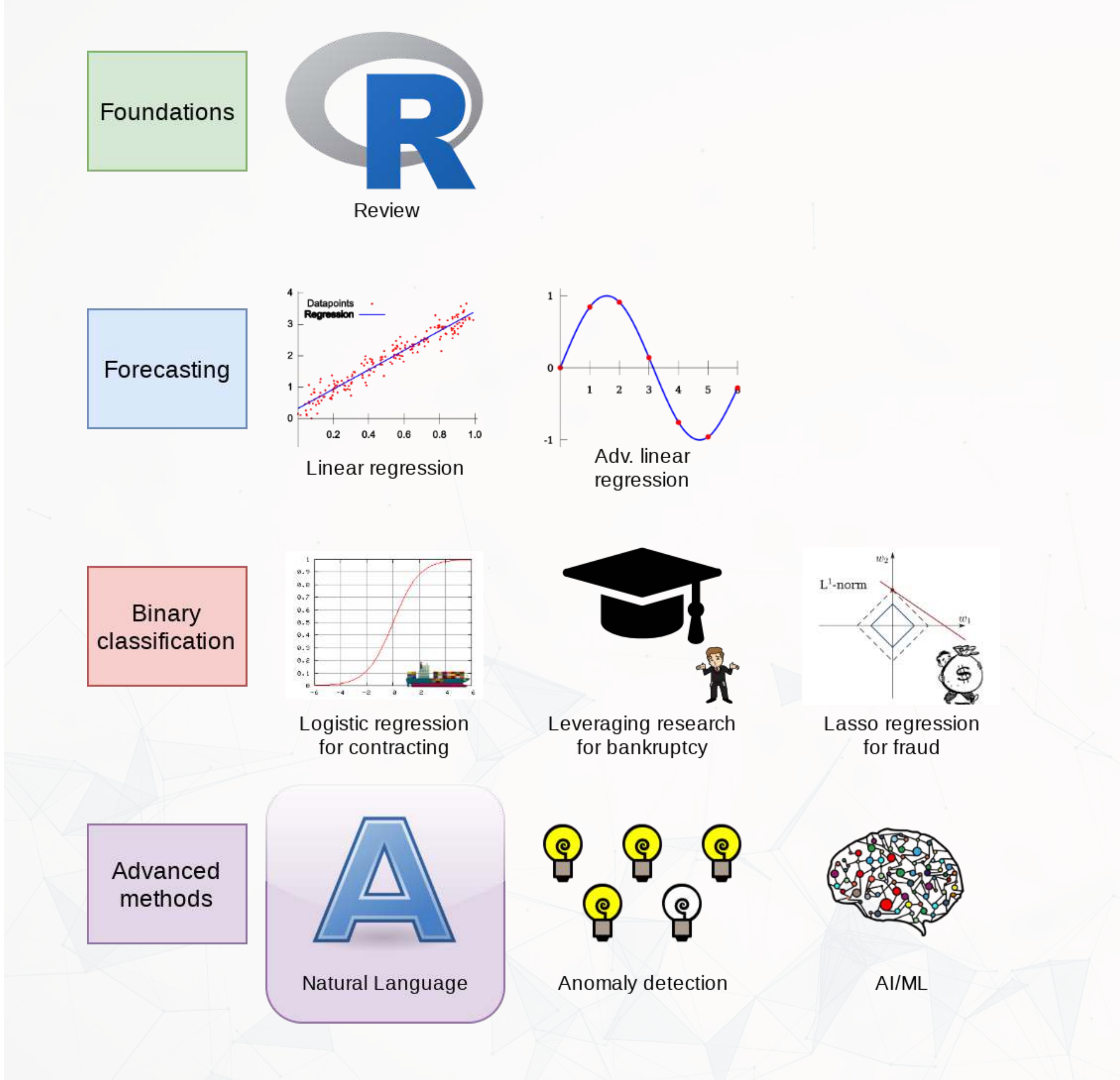
Dr. Richard M. Crowley

rcrowley@smu.edu.sg

https://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
  - The first chapter is helpful if you find the code in this lesson to be a bit too tricky
  - 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

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, such as satellite imagery
- Audio, such as phone call recordings
- Video, such as 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

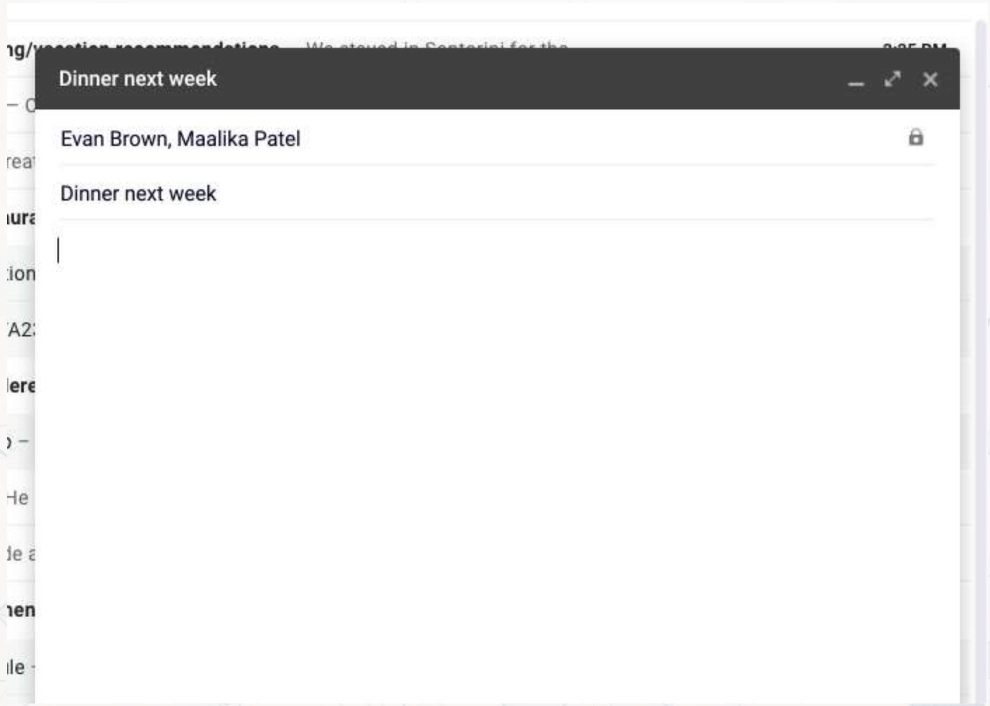
1. Business contracts
2. Legal documents
3. Any paperwork
4. News
5. Customer reviews or feedback
    - Including transcription (call centers)
6. Consumer social media posts
7. Chatbots and AI assistants

Natural Language Processing (NLP)

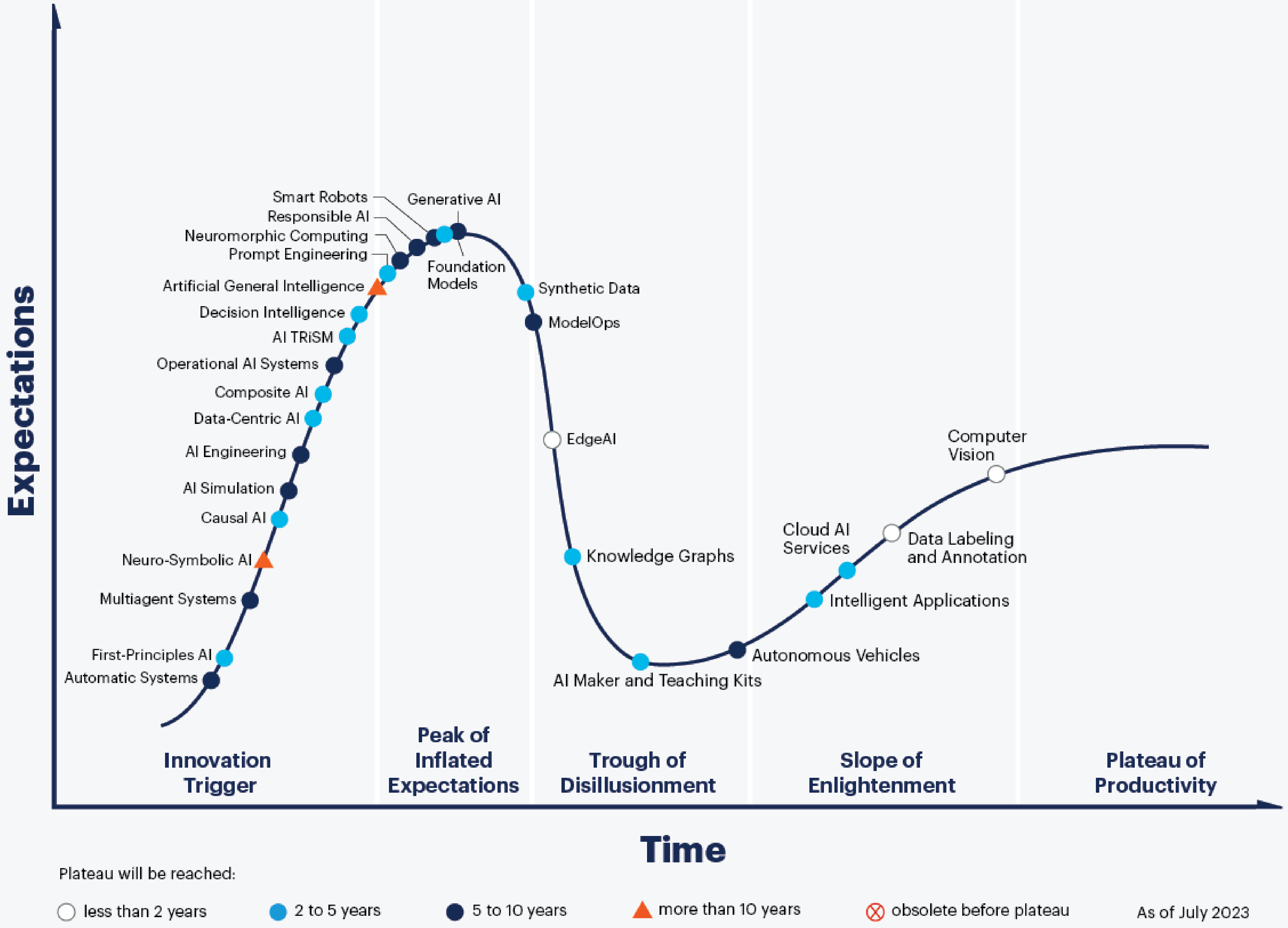
- NLP is the subfield of computer science focused on analyzing large amounts of unstructured textual information
  - Much of the work builds from computer science, linguistics, and statistics
- Unstructured text actually has some structure derived from language itself
  - Word selection
  - Grammar
  - 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, 2023



gartner.com

Source: Gartner

Gartner.

Case: How leveraging NLP helps call centers

- Natural Language Processing in Call Centers
- 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 Microsoft's annual report from 2021
  - 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/0001564590-21-039151.txt")
# str_wrap is from stringr from tidyverse
# The first 500 characters of the second paragraph
cat(str_wrap(substring(doc,1728,2228), 80))
```

```
Microsoft is a technology company whose mission is to empower every person
and every organization on the planet to achieve more. We strive to create
local opportunity, growth, and impact in every country around the world. Our
platforms and tools help drive small business productivity, large business
competitiveness, and public-sector efficiency. They also support new startups,
improve educational and health outcomes, and empower human ingenuity. We bring
technology and products together into ex
```

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, 11305, 11363), 80), '...')
```

```
[1] "...ntent=\"The live Ethereum price today is $1,590.58 USD with..."
```

```r
xpath <- '//*[@id="section-coin-overview"]/div[2]/span/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 $1,590.54 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="section-coin-overview"]/div[2]/span/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 $1,590.64"
```

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

```
[1] "Litecoin was priced at $64.70"
```

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, 138177, 138384), 80))
```

```
In fiscal year 2021, the COVID-19 pandemic continued to impact our business
operations and financial results. Cloud usage and demand benefited as customers
accelerate their digital transformation priorities.
```

```r
cat(str_wrap(str_sub(doc, 144162 , 144476), 80))
```

```
Operating expenses increased $2.0 billion or 4% driven by investments in cloud
engineering and commercial sales, offset in part by savings related to COVID-19
across each of our segments, prior year charges associated with the closing of
our Microsoft Store physical locations, and a reduction in bad debt expense.
```

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, 138287, 138384)
str_to_lower(sentence)
```

```
[1] "cloud usage and demand benefited as customers accelerate their digital transformation priorities. "
```

```r
str_to_upper(sentence)
```

```
[1] "CLOUD USAGE AND DEMAND BENEFITED AS CUSTOMERS ACCELERATE THEIR DIGITAL TRANSFORMATION PRIORITIES. "
```

```r
str_to_title(sentence)
```

```
[1] "Cloud Usage And Demand Benefited As Customers Accelerate Their Digital Transformation Priorities. "
```

- 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("Cloud usage and demand benefited...", 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] "Reid Hoffman, Partner, Greylock Partners"
 [2] "Hugh Johnston, Vice Chairman and Chief Financial Officer, PepsiCo"
 [3] "Teri List, Former Executive Vice President and Chief Financial Officer, Gap Inc."
 [4] "Satya Nadella, Chairman and Chief Executive Officer"
 [5] "Sandra E. Peterson, Lead Independent Director"
 [6] "Penny Pritzker, Founder and Chairman, PSP Partners"
 [7] "Carlos Rodriguez, Executive Chair, ADP, Inc."
 [8] "Charles W. Scharf, CEO and President, Wells Fargo & Company"
 [9] "John W. Stanton, Chairman, Trilogy Partnerships"
[10] "John W. Thompson, Partner, Lightspeed Venture Partners"
[11] "Emma Walmsley, CEO, GSK"
[12] "Padmasree Warrior, Founder, President and CEO, Fable Group Inc."
```

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

```
Microsoft's board consists of: Reid Hoffman, Hugh Johnston, Teri List, Satya
Nadella, Sandra E. Peterson, Penny Pritzker, Carlos Rodriguez, Charles W.
Scharf, John W. Stanton, John W. Thompson, Emma Walmsley, and Padmasree Warrior.
```

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] "Cloud usage and demand benefited as customers accelerate their digital transformation priorities. "
```

```r
str_replace_all(sentence, "digital transformation", "data science")
```

```
[1] "Cloud usage and demand benefited as customers accelerate their data science priorities. "
```

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] 474
```

```r
# First paragraph of the MD&A
cat(str_wrap(paragraphs[206], 80))
```

```
The following Management’s Discussion and Analysis of Financial Condition
and Results of Operations (“MD&A”) is intended to help the reader understand
the results of operations and financial condition of Microsoft Corporation.
MD&A is provided as a supplement to, and should be read in conjunction with,
our consolidated financial statements and the accompanying Notes to Financial
Statements (Part II, Item 8 of this Form 10-K). This section generally discusses
the results of our operations for the year ended June 30, 2021 compared to
the year ended June 30, 2020. For a discussion of the year ended June 30, 2020
compared to the year ended June 30, 2019, please refer to Part II, Item 7,
“Management’s Discussion and Analysis of Financial Condition and Results of
Operations” in our Annual Report on Form 10-K for the year ended June 30, 2020.
```

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,] 139992 140001
 [2,] 142476 142485
 [3,] 144664 144673
 [4,] 144834 144843
 [5,] 148712 148721
 [6,] 151464 151473
 [7,] 177859 177868
 [8,] 216135 216144
 [9,] 217104 217113
[10,] 218151 218160
[11,] 219629 219638
```

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?

```r
x <- str_detect(str_to_lower(paragraphs), "revenue")
x[51:60]
```

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

```r
sum(x)
```

```
[1] 86
```

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

```r
x <- str_count(str_to_lower(paragraphs), "revenue")
x[51:60]
```

```
 [1] 0 0 0 2 2 1 2 0 1 0
```

```r
max(x)
```

```
[1] 5
```

Example: Finding phrases

- Where is net income first mentioned in the document?

```r
str_locate(str_to_lower(doc), "revenue")
```

```
     start   end
[1,] 31243 31249
```

- 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), "revenue")
```

```
[1] "revenue"
```

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
  - Available at: 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

- Finding full sentences mentioning COVID

```r
# Extract all sentences mentioning COVID from the annual report
str_extract_all(doc, '(?<=^|\\.\\s{1,5})[^.]*?COVID[^.]*?\\.')
```

```
[[1]]
 [1] "\nIn March 2020, the World Health Organization declared the outbreak of COVID-19 to be a pandemic."
 [2] "The COVID-19 pandemic continues to have widespread and unpredictable impacts on global society,
economies, financial markets, and business practices, and continues to impact our business operations,
including our employees, customers, partners, and communities."
 [3] "Refer to Management's Discussion and Analysis of Financial Condition and Results of Operations (Part II,
Item 7 of this Form 10-K) for further discussion regarding the impact of COVID-19 on our fiscal year 2021
financial results."
 [4] "The extent to which the COVID-19 pandemic impacts our business going forward will depend on numerous
evolving factors we cannot reliably predict."
 [5] "\nWith a continued focus on digital transformation, Microsoft is helping to ensure that no one is left
behind, particularly as economies recover from the COVID-19 pandemic."
 [6] "During fiscal year 2021, our Daily Pulse surveys gave us invaluable insights into ways we could support
employees through the COVID-19 pandemic and addressing racial injustice."
 [7] "\nWe took a wide variety of measures to protect the health and well-being of our employees, suppliers,
```

Breaking down the example

> `'(?<=^|\\.\\s{1,5})[^.]*?COVID[^.]*?\\.'`

- `(?<=...)` is called a *positive look-behind assertion*
  - It succeeds whenever the '…' matches the text before what you want to find
  - `^` is the start of the string
  - `\\.\\s{1,5}` is a period followed by some whitespace characters (up to 5)
    - A quirk of look-behinds is you need to specify a maxmimum length for everything
  - `|` is an *or*
  - Taken together: the look-behind matches if there is a new paragraph or a period followed by whitespace
- `[^.]*?`
  - `[^.]` is anything except a period
  - `*` means 0 or more of the preceeding pattern
  - `?` means keep it as short as possible
- `COVID` is the literal text
- `\\.` is a period

Breaking down the example

- Let's examine the output: `In fiscal year 2021, the COVID-19 pandemic continued to impact our business operations and financial results.`
- Our regex was `(?<=^|\\.\\s{1,5})[^.]*?COVID[^.]*?\\.`
- Matching regex components to output:
  - `(?<=^|\\.\\s{1,5})` ⇒ start of the paragraph (via ^)
  - `[^.]*?` ⇒ `In fiscal year 2021, the`
  - `COVID` ⇒ `COVID`
  - `[^.]*?` ⇒ `-19 pandemic continued to impact our business operations and financial results`
  - `\\.` ⇒ `.`

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

```r
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

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

```
[1] "STADLAUER MALZFABRIK"        "ELECT ET EAUX DE MADAGASCAR"
[3] "JOAO FORTES ENGENHARIA SA"
```

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

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

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

```
 [1] "INTERNATIONAL ENTERTAINMENT"  "PREMIERE HORIZON ALLIANCE"
 [3] "ELECT ET EAUX DE MADAGASCAR"  "HUA YIN INTERNATIONAL HOLDIN"
 [5] "JOAO FORTES ENGENHARIA SA"    "TIANJIN TROILA INFORMATION"
 [7] "OVERSEAS CHINESE TOWN (ASIA)" "ASIA-PACIFIC STRATEGIC INVES"
 [9] "FUTURA CONSORCIO INMOBILIARI" "FRANCE TOURISME IMMOBILIER"
[11] "BONEI HATICHON CIVIL ENGINE"
```

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

```
[1] "INTERNATIONAL ENTERTAINMENT"  "SHANDONG XINNENG TAISHAN"
[3] "CHONG HONG CONSTRUCTION CO"   "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 × 2
  SG_firm pct_SG
    <dbl>  <dbl>
1       0  0.746
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 5 entries

Search:

	fic	avg_name_length
1	PER	28
2	TUR	27
3	ZAF	26
4	CHN	25.2
5	EGY	24.5

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
  - Available at: 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)
library(quanteda.textstats)
textstat_readability(doc, "Flesch.Kincaid")
```

```
  document Flesch.Kincaid
1    text1       16.85874
```

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 20.88005
```

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*

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

```
  document Coleman.Liau.short
1    text1           15.48359
```

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("0001564590-21-039151"), 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

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

ID	word
0001564590-21-039151	this
0001564590-21-039151	report
0001564590-21-039151	includes
0001564590-21-039151	estimates
0001564590-21-039151	projections
0001564590-21-039151	statements

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?

For right now, not much

- If we have every publicly available government filed press release in the US?
  - 1,479,068 files through July 2018…
    - ~2TB if we include all English words
    - ~45GB if we restrict just to the 3,752 words in the Microsoft annual report…

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))
nrow(df_doc)
```

```
[1] 37234
```

```r
nrow(df_doc_stop)
```

```
[1] 21171
```

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    0001564590-21-039151                     services 250 21171 1.180861e-02
2    0001564590-21-039151                     products 194 21171 9.163478e-03
3    0001564590-21-039151                    financial 166 21171 7.840914e-03
4    0001564590-21-039151                     business 144 21171 6.801757e-03
5    0001564590-21-039151                          tax 140 21171 6.612819e-03
6    0001564590-21-039151                      revenue 137 21171 6.471116e-03
7    0001564590-21-039151                    customers 132 21171 6.234944e-03
8    0001564590-21-039151                     software 130 21171 6.140475e-03
9    0001564590-21-039151                        cloud 124 21171 5.857069e-03
10   0001564590-21-039151                         2021 118 21171 5.573662e-03
11   0001564590-21-039151                         year 113 21171 5.337490e-03
12   0001564590-21-039151                        based 110 21171 5.195787e-03
13   0001564590-21-039151                      billion 109 21171 5.148552e-03
14   0001564590-21-039151                    microsoft 105 21171 4.959615e-03
```

Sentiment

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

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

```
# A tibble: 11 × 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 × 2
  word    sentiment
  <chr>   <chr>
1 2-faces negative
2 abound  positive
```

Sentiment

NRC Word-Emotion

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

```
# A tibble: 10 × 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

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

```
# A tibble: 6 × 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"))
tf_sent[1:5,]
```

```
                     ID      word   n total          tf sentiment
1 0001564590-21-039151  services 250 21171 0.011808606      <NA>
2 0001564590-21-039151  products 194 21171 0.009163478      <NA>
3 0001564590-21-039151 financial 166 21171 0.007840914      <NA>
4 0001564590-21-039151  business 144 21171 0.006801757      <NA>
5 0001564590-21-039151       tax 140 21171 0.006612819      <NA>
```

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

```
                       ID      word  n total          tf    sentiment
96  0001564590-21-039151  required 34 21171 0.001605970 constraining
102 0001564590-21-039151     risks 33 21171 0.001558736  uncertainty
117 0001564590-21-039151      laws 31 21171 0.001464267    litigious
141 0001564590-21-039151 effective 27 21171 0.001275329     positive
144 0001564590-21-039151    losses 27 21171 0.001275329     negative
```

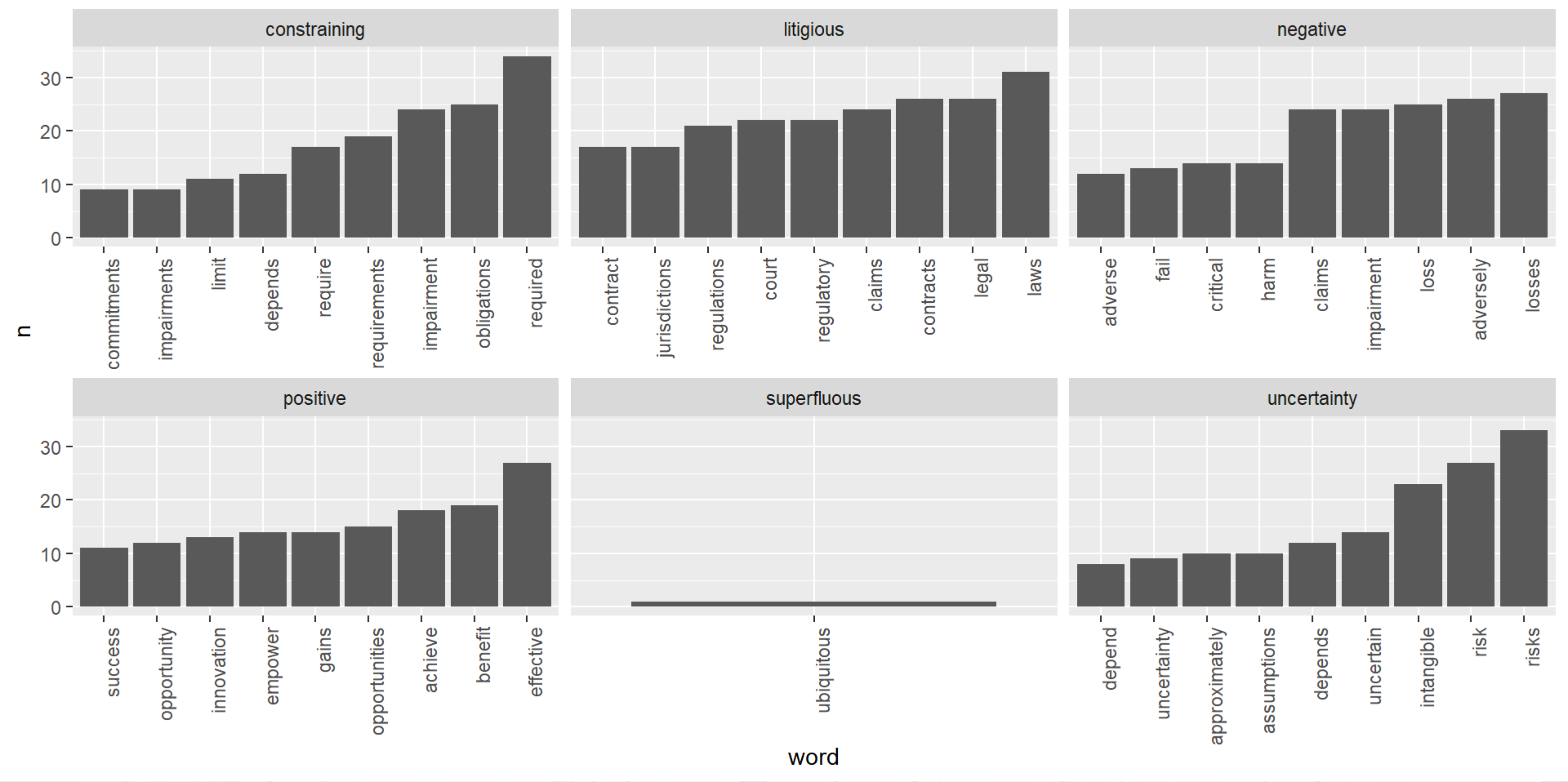
Summarizing document sentiment

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

```
constraining    litigious     negative     positive  superfluous  uncertainty
1.284776e-02 1.690992e-02 3.226111e-02 1.785461e-02 4.723442e-05 1.218648e-02
```

Visualizing sentiment

```r
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")
```



Visualizing a document as a word cloud

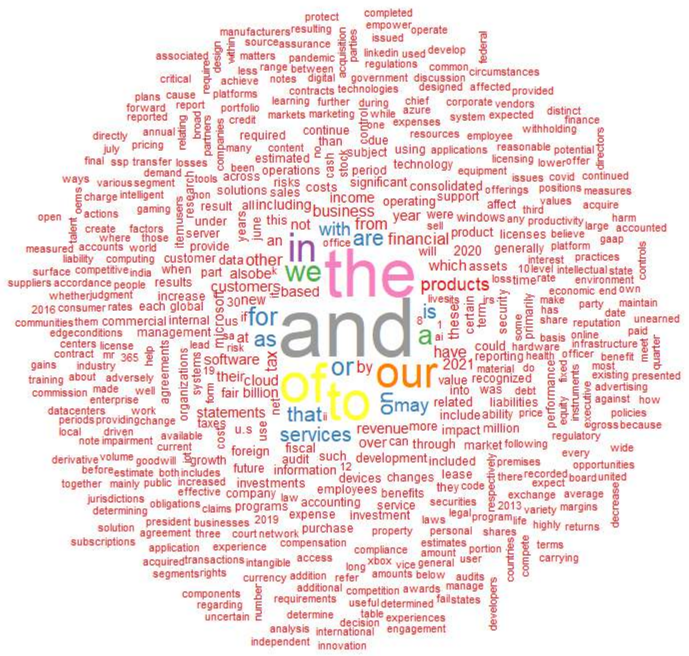
- quanteda provides `textplot_wordcloud()`
- `cast_dfm()` converts tidy term frequencies to Quanteda

```r
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

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



- You can also check out the wordcloud and wordcloud2 packages

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
  - Available at: rmc.link/420r7

End Matter

Wrap up

- For next session (in 2 weeks):
    - Finish the third assignment
        - Submit on eLearn
    - Datacamp
        - Check out the recommended chapter on text analysis
    - Start on the group project
- Survey on the class session at this QR code:

Packages used for these slides

- DT
- downlit
- httr
- kableExtra
- knitr
- plotly
- quanteda
- quarto
- {RColorBrewer}
- readtext
- revealjs
- tidytext
- tidyverse, including 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)
  }
}
```