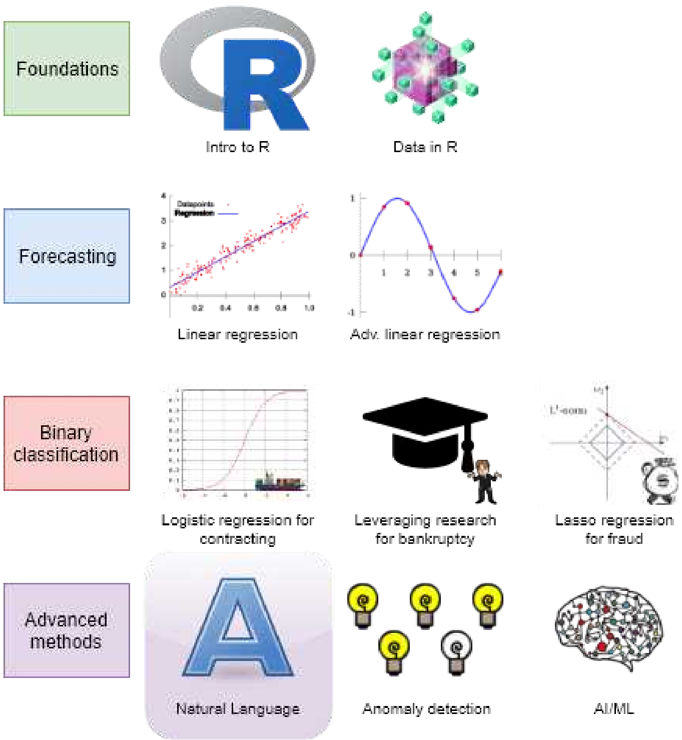
Forecasting and Forensic Analytics

Session 8: Textual Analysis

Dr. Wang Jiwei

Preface

Learning objectives



- **Theory:**
  - Natural Language Processing
- **Application:**
  - Analyzing a DBS annual report
- **Methodology:**
  - Text analysis
  - Machine learning
- **Additional Readings:**
  - R for Data Science
  - Text Mining with R

Reminder: Group project

1. All groups have been assigned topics
2. Data is available on Kaggle.com
3. Submission deadline: check eLearn

Textual data and textual analysis

Review of Last Session

- Last session we saw that textual measures can help improve our fraud detection algorithm
- We 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

> We'll get started on these today -- sentiment and readability
>
> We *will* cover making topic models in a later session

Why is textual analysis harder?

- Thus far, everything we've worked with is what is known as *structured data*
  - Structured data is formatted, nicely indexed, and easy to use
- Text data is *unstructured*
  - If we get an annual report with 500 pages of text...
    - Where is the information we want?
    - What can we get?
    - How do we crunch 500 pages into something that is...
      1. Manageable?
      2. Meaningful?

> This is what we will work on today, and we will revisit some of this in the remaining seminar sessions

Structured data

- Our long or wide format data

Long 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
```

Wide format

```
## # A tibble: 3 x 4
##   RegionID `2008-01` `2008-02` `2008-03`
##      <dbl>     <dbl>     <dbl>     <dbl>
## 1    61639        NA        NA        NA
## 2    84654        NA        NA        NA
## 3    61637        NA        NA        NA
```

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

Unstructured data

- Text
  - Open responses to question, reports, etc.
- Images
  - Satellite imagery
- Audio
  - Phone call recordings
- Video
  - Security camera footage

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

Some examples of unstructured data in business:

1. Business contracts, Legal documents, Any other paperwork
2. News articles and social media posts
3. Customer reviews or feedback, incl. transcription (call centers)
4. Chatbots and AI assistants

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?

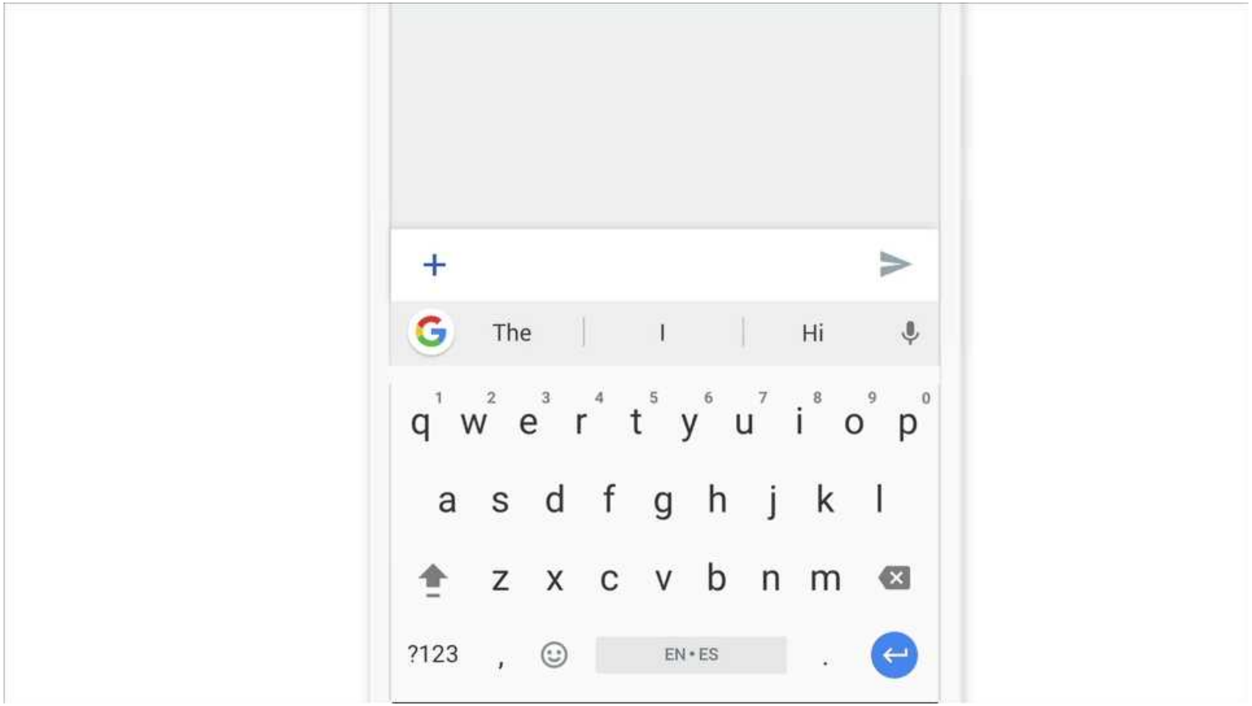
Natural Language Processing (NLP)

SMU SINGAPORE MANAGEMENT UNIVERSITY School of Accountancy

- NLP is the subfield of computer science focused on analyzing large amounts of unstructured language information such as speech and text
  - Much of the work builds from computer science, linguistics, and statistics
  - We will cover NLP of text, rather than speech recognition
- Unstructured text actually has some structure -- language
  - Word selection
  - Grammar
  - Word relations
- 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



Cases: How NLP helps

- How Analytics, Big Data and AI Are Changing Call Centers Forever

> What are call centers using NLP for?
>
> How does NLP help call centers with their business?

- ESG News Score by Truvalue Labs under Factset

> Using machine learning to generate scores on millions of documents on ESG
>
> It is fundamentally sentiment analysis which we will cover later

String manipulation in R

Special characters

```
#create strings with either double quotes "" (preferred) or single quotes ''
string <- c("string", 'string')
string
```

```
## [1] "string" "string"
```

```
#To include quotes in a string, we need to precede it with a backslash `\`
string <- c("\"")
string #the printed output of a string is not the same as string itself
```

```
## [1] "\""
```

```
writeLines(string) #to print out the raw content
```

```
## "
```

```
#To include '\', we need to put '\' before '\'
string <- c("\\")
writeLines(string)
```

```
## \
```

Special characters

- Also, some spacing characters have special symbols:
  - `\t` is tab
  - `\n` is newline
  - type `?'"'` to see more

```
string <- c("What is this? \tIt is a dog.")
writeLines(string)
```

```
## What is this?     It is a dog.
```

```
string <- c("What is this? \nIt is a dog.")
writeLines(string)
```

```
## What is this?
## It is a dog.
```

Loading in text data from files

- Use `read_file()` from `package:tidyverse`'s `package:readr` package to read in text data
- We'll use DBS's annual report from 2017
  - a .txt file we purchased from a data vendor

```
# Read text from a .txt file using read_file()
doc <- read_file("../../Data/Session_8-dbs2017.txt")
class(doc)
```

```
## [1] "character"
```

```
# str_wrap is from stringr from tidyverse
cat(str_wrap(substring(doc, 1, 160), 80))
```

```
## |$!$| DBS Group Holdings Ltd Annual Report 2017 Development Bank of Singapore |
## $!$| Digital Bank of Singapore 2018 marks DBS’ 50th anniversary. We trace our ro
```

- `cat()` is to concatenate and print

Loading from other file types

- Ideally you have a .txt file already
- 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
      - Things like `<a>`, `<table>`, `<img>`, etc.
    - Load from a URL using `package:httr` or `package:RCurl`
    - Use `package:XML` or `package:rvest` to parse out specific pieces of html files
    - For python, use `package:lxml` or `package:beautifulsoup4` (bs4) to turn into structured documents
  - If you are interested in web scraping in R, you may study the Datacamp course: Working with Web Data in R

Loading from other file types

- Ideally you have a .txt file already
- Other common file types:
  - PDF files
    - Use `package:pdftools` and you can extract text into a vector
    - Use `package:tabulizer` to extract tables straight from PDF files!
      - Requires `package:tabulizerjars` and `package:rJava`
    - PDF files with images?
      - `package:tesseract`: optical character recognition (OCR) engine

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 `package:stringr` as opposed to base R -- `package:stringr`'s commands are much more consistent

- Every function in `package:stringr` can take a vector of strings for the first argument

Subsetting text

- Base R: Use `substr()` or `substring()`
- `package: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)

```
cat(str_wrap(str_sub(doc, 16141, 16249), 80))
```

```
## “Having invested time and resources in digitalising the bank, we have seen
## visible results.” CEO Piyush Gupta
```

```
cat(str_wrap(str_sub(doc, 75315, 75465), 80))
```

```
## retail and wealth management business acquired from ANZ added another SGD 8
## billion of loans, resulting in overall constant-currency loan growth of 11%
```

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: Common functions

```
sentence <- str_sub(doc, 15272, 15306)
str_to_lower(sentence)
```

```
## [1] "another example is posb smart buddy"
```

```
str_to_upper(sentence)
```

```
## [1] "ANOTHER EXAMPLE IS POSB SMART BUDDY"
```

```
str_to_title(sentence)
```

```
## [1] "Another Example Is Posb Smart Buddy"
```

Examples: `paste()` function

```
# board is a list of DBS's director names
# titles is a list of the director's titles
paste(board[1:5], titles[1:5], sep=", ")
```

```
## [1] "Peter Seah, Non-Executive Chairman"  "Piyush Gupta, CEO"
## [3] "Bonghan Cho, Independent Director"   "Euleen Goh, Non-Executive Director"
## [5] "Ho Tian Yee, Non-Executive Director"
```

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

```
## DBS's board consists of: Peter Seah, Piyush Gupta, Bonghan Cho, Euleen Goh, Ho
## Tian Yee, Punita Lal, Anthony Lim, Oliver Lim, Ow Foong Pheng, Andre Sekulic,
## and Tham Sai Choy.
```

- `collapse` = is to collapse the output into a single string. Refer to `paste()` documentation. Try replacing it with `sep = ", "`

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
  - Use `str_replace()` to replace the first occurrence

```
sentence
```

```
## [1] "Another example is POSB Smart Buddy"
```

```
str_replace_all(sentence, "is", "was")
```

```
## [1] "Another example was POSB Smart Buddy"
```

Transforming text

- Split text using `str_split()`
  - This function returns a list of vectors!
    - This is because it will turn every splitted 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

```
#doc contains 1 vector of 1 string, ie, DBS's 2017 annual report
paragraphs <- str_split(doc, '\n') #split by new line (paragraph)
str(paragraphs) #list consisting of 1 vector of 423 strings
```

```
## List of 1
##  $ : chr [1:423] "|$!$|" "DBS Group Holdings Ltd Annual Report 2017 Development Bank of Si
```

```
length(paragraphs) #length of the list
```

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

```
paragraphs <- paragraphs[[1]] #extract first element of the list
length(paragraphs) #length of the vector of strings
```

```
## [1] 423
```

Example: Splitting text

```
# the longest paragraph
nchar <- str_length(paragraphs) #no. of char for each paragraph
ncharmax <- max(nchar) #max number of paragraphs
index <- match(ncharmax, nchar) #index number for the longest paragraphs
cat(str_wrap(paragraphs[index], 80))
```

```
## 42 DBS Annual Report 2017 Institutional Banking Institutional Banking’s
## performance remained stable in 2017. Total income increased 1% to SGD 5.28
## billion. Net interest income grew 4% supported by broad-based asset and deposit
## growth offset by lower treasury customer income. Underlying growth from the
## large corporate segment was healthy, supported by strong balance sheet growth
## despite the impact of residual headwinds from the oil and gas support services
## sector. The SME segment grew 11%. Expenses were tightly managed and grew at
## 1%. Total allowances increased by SGD 827 million to SGD 2.33 billion to remove
## lingering uncertainty over the oil and gas support services portfolio. Key
## highlights We place customers at the centre of all we do. We are committed to
## helping our large corporate and SME clients with their financial needs. Our
## relationship teams, organised by industry segments, are able to understand and
## anticipate our customers’ business needs better. Our insights into the region
## and digital engagements with clients have elevated our partnership with them
## well beyond conventional bank-customer interactions. This has helped us foster
## deeper conversations and relationships with clients. In 2017, we continued
## to make investments in product capabilities and digital innovation to support
## the transformational and financial objectives of our clients. We delivered
## cutting-edge, industry leading solutions centred around increasing efficiency,
## enhancing risk management and reducing costs for clients. Here are some key
## highlights during the year. A leading innovative cash management franchise
## Our cash management income grew 32% and growth was broad-based across all key
## markets. We closed a record number of mandates as clients continued to reap
```

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

```r
# 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 FALSE FALSE FALSE FALSE FALSE
```

```r
sum(x)
```

```
## [1] 9
```

```r
# What is the most net income mentioned in any paragraph
x <- str_count(str_to_lower(paragraphs), "net income")
x[1:10]
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```r
max(x)
```

```
## [1] 2
```

Example: Finding phrases

```
# the paragraph mentions "net income" the most
# match() returns an index/position number
cat(str_wrap(paragraphs[match(2, x)], 80))
```

```
## 142 DBS Annual Report 2017 The table below shows the movements in specific and
## general allowances during the year for the Group. The Group Charge/ Balance at
## In $ millions 2017 Specific allowances Loans and advances to customers (Note 18)
## Investment securities Properties and other fixed assets Off-balance sheet credit
## exposures Others Total specific allowances Total general allowances for credit
## losses Total allowances 2016 Specific allowances Loans and advances to customers
## (Note 18) Investment securities Properties and other fixed assets Off-balance
## sheet credit exposures Others Total specific allowances Total general allowances
## for credit losses Total allowances 12 Income Tax Expense The Group In $ millions
## Current tax expense – Current year – Prior years' provision Deferred tax expense
## – Prior years' provision – Origination of temporary differences Total 2017
## 2016 In $ millions 820 (79) 4 (74) 671 804 (59) – (22) 723 The deferred tax
## credit in the income statement comprises the following temporary differences:
## The Group In $ millions Accelerated tax depreciation Allowances for loan losses
## Other temporary differences Deferred tax credit to income statement 2017 2016
## 5 3 30 (105) (70) (46) 21 (22) Profit before tax Prima facie tax calculated at
## a tax rate of 17% (2016: 17%) Effect of different tax rates in other countries
## Net income not subject to tax Net income taxed at concessionary rate Expenses
## not deductible for tax Others Income tax expense charged to income statement
## The Group 2017 2016 5,175 5,083 880 864 6 (112) (99) 13 (17) 671 (1) (60) (114)
## 15 19 723 Deferred income tax relating to available-for-sale financial assets
## and others of $4 million (2016: $12 million) and own credit risk of $3 million
## was credited directly to equity. Refer to Note 21 for further information on
## deferred tax assets/liabilities. (Write-back) Net write-off Acquisition Exchange
## to income during 1 January statement 1,270 2,238 81 19 28 69 the year (1,210)
```

Example: Finding phrases

- Where is net income first mentioned in the document?

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

```
##      start   end
## [1,] 90422 90431
```

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

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

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

Example: Finding all locations of a phrase

- Where is "net income" mentioned in the text?

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

```
## [[1]]
##        start    end
##  [1,]  90422  90431
##  [2,] 450392 450401
##  [3,] 452098 452107
##  [4,] 457843 457852
##  [5,] 477161 477170
##  [6,] 512648 512657
##  [7,] 515898 515907
##  [8,] 515928 515937
##  [9,] 600314 600323
## [10,] 632526 632535
```

> why 10, not 9 as shown in previous slide?

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
  - use `str_length()` to find the number of characters in a string vector

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:]]+`

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

```
## [[1]]
## [1] "investor@dbs.com."     "info-sg@dbsonline.com" "info@nets.com.sg"
## [4] "investor@dbs.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 in R
- 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`
- Warning: this is not a perfect email regex but good enough for documents with genuine email

Useful regex components: Content

- There's a nice cheat sheet here
  - More detailed documentation
- 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, ie, no space
  - `[:blank:]` matches whitespace except newlines

Example: Regex content

```
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`, or `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
    - Append `?` to any quantity operator to make it prefer the shortest match possible. Eg, x??: 0 or 1, prefer 0 x

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: Real estate firms

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

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

```r
# 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"
```

```r
# 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"
```

```r
# Real estate firm names with at least 11 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"
```

Example: Real estate firms

```r
# Real estate firm names with at least 11 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"
```

- within ( ) is a group component
- [^AEIOU]*[AEIOU] means zero or more consonants plus one vowel
- This group component will repeat 11 times
- If you remove [^AEIOU]*, it will look for 11 vowels in a row.

Why is regex so important?

- Regex can be used to match anything in text
  - Simple things like phone numbers
  - More complex things like addresses, date, email, etc
- It can be used to parse through large markup documents
  - HTML, XML, LaTeX, etc.
- Very good for validating the format of text

> 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
  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

```
# Compustat firm names example
df_RE_names <- df_RE %>%
  group_by(isin) %>%
  slice(1) %>%
  mutate(SG_in_name = str_detect(conm, "(SG|SINGAPORE)"),
         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
```

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

Readability

Readability

- Thanks to the `package:quanteda`, readability is very easy to calculate in R
  - Use the `textstat_readability()` function to compute 46 different measures of readability
- The following are some populars measures of readability:
  - Flesch Kinkaid: 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
  - Gunning fog: An index that was commonly used in business and publishing
  - Coleman-Liau: An index with a unique calculation method

Readability: Flesch score

$$206.835 - 1.015\left(\frac{\#\ words}{\#\ sentences}\right) - 84.6\left(\frac{\#\ syllables}{\#\ words}\right)$$

- A score generally below 100
  - *Higher is more readable*
  - Conversational English should be around 80-90
  - A JC or poly graduate should be able to read anything 50 or higher
  - A Bachelor's degree could be necessary for anything below 30

```r
library(quanteda)
textstat_readability(doc, "Flesch")
```

```
##   document   Flesch
## 1    text1 26.15754
```

Readability: Fog

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

- *Lower is more readable*
- An approximate grade level required for reading a document
  - A JC or poly graduate should read at a level of 12
    - New York Times articles are usually around 13
  - A Bachelor's degree holder should read at 17

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

```
##   document      FOG
## 1    text1 21.32477
```

Readability: Coleman-Liau

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

- *Lower is more readable*
- Provides an approximate grade level like Fog, on the same scale as Fog

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

```
##   document Coleman.Liau.ECP
## 1    text1         32.98475
```

Tokenization and tidy text

Converting text to words

- Tidy text is when you have *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 `package:tidytext` can handle this conversion for us!
  - Use the `unnest_tokens()` function
  - Note: it also converts to lowercase by default. Use the option `to_lower = FALSE` to avoid this if needed
- `package:tidytext` uses the `package: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 =`)
  - "words": The default, individual words
  - "ngrams": Collections of words (default of 2, specify with `n =`)
  - A few other less commonly used tokenizers

Example: tokenization

```
# Example of "tokenizing"
library(tidytext)
df_doc <- data.frame(ID=c("DBS"), text = c(doc))
df_doc <- unnest_tokens(df_doc, output = word, input = text, token = "words")
html_df(df_doc[1:7, ])
```

	ID	word
1	DBS	dbs
1.1	DBS	group
1.2	DBS	holdings
1.3	DBS	ltd
1.4	DBS	annual
1.5	DBS	report
1.6	DBS	2017

```
# word is the name for the new column
# text is the name of the string column in the input data
```

Why convert to lowercase?

- The `unnest_tokens()` function converts to lowercase by default.
    - Use the option `to_lower = FALSE` to avoid this if needed

> 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*: when the dimensionality increases, the volume of the space increases so fast that the available data become sparse.

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
- `package:stopwords` covers various languages
- The popular stopwords are as follows:

```
library(stopwords) # get a list of stopwords
stop_en <- stopwords("english")  # Snowball English
paste0(length(stop_en), " words: ", paste(stop_en[1:10], collapse=", "))
```

```
## [1] "175 words: i, me, my, myself, we, our, ours, ourselves, you, your"
```

```
stop_SMART <- stopwords(source = "smart")  # SMART English
paste0(length(stop_SMART), " words: ", paste(stop_SMART[1:8], collapse = ", "))
```

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

```
stop_fr <- stopwords("french")  # Snowball French
paste0(length(stop_fr), " words: ", paste(stop_fr[1:10], collapse = ", "))
```

```
## [1] "164 words: au, aux, avec, ce, ces, dans, de, des, du, elle"
```

Delete the stopwords

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

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

```
## [1] 116348
```

```
nrow(df_doc_stop)
```

```
## [1] 72726
```

Converting to term frequency

- term frequency: how frequently a term/token occurs in a document

```
# to count n = how many times for each token/word in each ID
terms <- df_doc_stop %>%
  count(ID, word, sort = TRUE) %>%
  ungroup()
# to sum total words per ID and term frequency per word and ID
total_terms <- terms %>%
  group_by(ID) %>%
  summarize(total = sum(n)) %>% ungroup()
tf <- left_join(terms, total_terms) %>% mutate(tf = n/total)
tf[1:10, ]
```

```
##      ID       word   n total          tf
## 1   DBS       risk 796 72726 0.010945192
## 2   DBS        dbs 750 72726 0.010312680
## 3   DBS  financial 677 72726 0.009308913
## 4   DBS       2017 674 72726 0.009267662
## 5   DBS management 598 72726 0.008222644
## 6   DBS      group 545 72726 0.007493881
## 7   DBS          1 461 72726 0.006338861
## 8   DBS     credit 397 72726 0.005458846
## 9   DBS     income 393 72726 0.005403845
## 10 DBS      total 367 72726 0.005046338
```

Sentiment analysis

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 `package:tidytext`

```r
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

```
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

```
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

```
tf_sent <- tf %>% left_join(get_sentiments("loughran"))
tf_sent[1:5, ]
```

```
##    ID       word   n total          tf   sentiment
## 1 DBS       risk 796 72726 0.010945192 uncertainty
## 2 DBS        dbs 750 72726 0.010312680        <NA>
## 3 DBS  financial 677 72726 0.009308913        <NA>
## 4 DBS       2017 674 72726 0.009267662        <NA>
## 5 DBS management 598 72726 0.008222644        <NA>
```

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

```
##      ID         word   n total          tf   sentiment
## 1   DBS         risk 796 72726 0.010945192  uncertainty
## 63  DBS         loss 128 72726 0.001760031     negative
## 83  DBS        risks 112 72726 0.001540027  uncertainty
## 92  DBS requirements 104 72726 0.001430025 constraining
## 106 DBS    exposures  97 72726 0.001333773  uncertainty
```

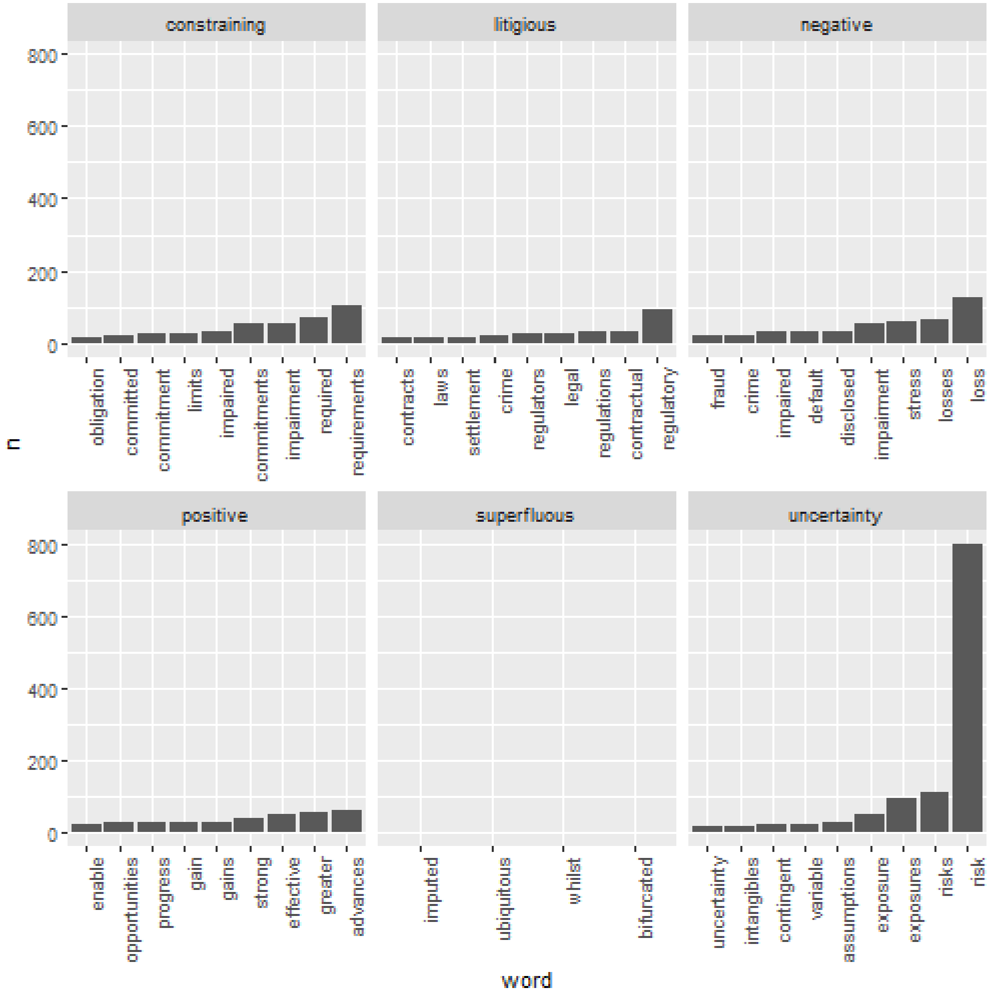
- You may note that *risk* and *risks* are counted as different words. We will consider similar words as the same in the next topic. The process is called stemming.

Summarizing document sentiment

```
# spread a key-value pair across multiple columns (long to wide)
tf_sent %>%
  spread(sentiment, tf, fill = 0) %>% # missing values will be replaced with 0
  select(constraining, litigious, negative, positive, superfluous,
         uncertainty) %>%
  colSums() %>% # Form column sums for numeric arrays (or data frames)
  scales::percent(accuracy = 0.001)
```

```
## constraining    litigious     negative     positive  superfluous  uncertainty
##     "0.828%"     "0.628%"     "1.656%"     "1.356%"     "0.007%"     "1.848%"
```

visualizing sentiment



Visualizing as a word cloud

- `package:quanteda` also provides an easy way to make a word cloud
  - `textplot_wordcloud()`
- There is also the `package:wordcloud2` packages for this

```
# You may need to add unique_docnames = FALSE if there is an error from corpus()
df_doc_stop %>% filter(!str_detect(word, "([:digit:]|dbs|group)")) %>%
  corpus(docid_field = "ID", text_field = "word", unique_docnames = F) %>%
  dfm() %>% textplot_wordcloud(color = RColorBrewer::brewer.pal(10, "RdBu"))
```

Another reason to use stopwords

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

```r
# You may need to add unique_docnames = FALSE if there is an error from corpus()
corp <- corpus(df_doc, docid_field = "ID", text_field = "word",
               unique_docnames = F)
textplot_wordcloud(dfm(corp), color = RColorBrewer::brewer.pal(10, "RdBu"))
```

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 using the offline files on eLearn, not in the DataCamp light console
- Do exercises 6 through 8 in today's practice file
  - R Practice

Summary of Session 8

For next week

- Try to replicate the code
- Continue your Datacamp career track
- Third individual assignment
  - Submit on eLearn on 8th March

Supplementary readings

- Reading text files with readtext
- The Use of Word Lists in Textual Analysis
- When Is a Liability Not a Liability?

R packages used in this slide

This slide was prepared on 2021-02-23 from Session_8s.Rmd with R version 4.0.3 (2020-10-10) Bunny-Wunnies Freak Out on Windows 10 x64 build 18362 😃.

The attached packages used in this slide are:

```
##   stopwords    tidytext    quanteda     forcats     stringr       dplyr       purrr
##       "2.0"     "0.2.4"     "2.0.1"     "0.5.1"     "1.4.0"     "1.0.4"     "0.3.4"
##       readr       tidyr      tibble     ggplot2   tidyverse  kableExtra       knitr
##     "1.4.0"     "1.1.2"     "3.0.6"     "3.3.3"     "1.3.0"     "1.1.0"      "1.31"
```