THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Week 9: Text as Data

MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2025

Ryan Hübert
Associate Professor of Methodology

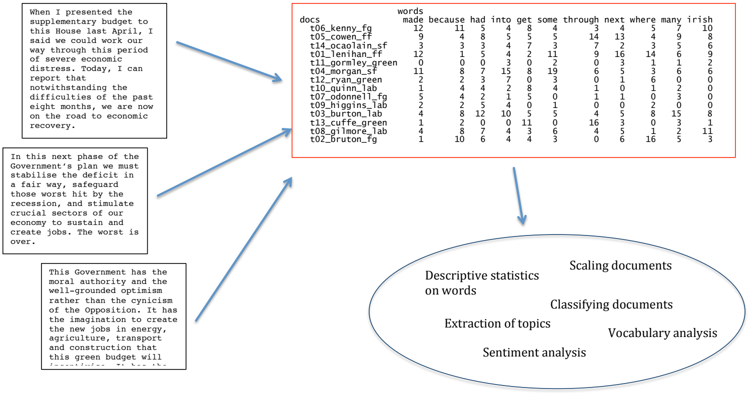
Slides last updated: 25 November 2025

Outline

Text as (quantitative) data

Quantitative text analysis (QTA): use quantitative methods to analyse and work with texts

But first, we need to *quantify* the texts!



```
When I presented the
supplementary budget to
this House last April, I
said we could work our
way through this period
of severe economic
distress. Today, I can
report that
notwithstanding the
difficulties of the past
eight months, we are now
on the road to economic
recovery.
```

```
In this next phase of the
Government's plan we must
stabilise the deficit in
a fair way, safeguard
those worst hit by the
recession, and stimulate
crucial sectors of our
economy to sustain and
create jobs. The worst is
over.
```

```
This Government has the
moral authority and the
well-grounded optimism
rather than the cynicism
of the Opposition. It has
the imagination to create
the new jobs in energy,
agriculture, transport
and construction that
this green budget will
```

```
                     words
docs                  made because had into get some through next where many irish
  t06_kenny_fg          12      11   5    4   8    4       3    4     5    7    10
  t05_cowen_ff           9       4   8    5   5    5      14   13     4    9     8
  t14_ocaolain_sf        3       3   3    4   7    3       7    2     3    5     6
  t01_lenihan_ff        12       1   5    4   2   11       9   16    14    6     9
  t11_gormley_green      0       0   0    3   0    2       0    3     1    1     2
  t04_morgan_sf         11       8   7   15   8   19       6    5     3    6     6
  t12_ryan_green         2       2   3    7   0    3       0    1     6    0     0
  t10_quinn_lab          1       4   4    2   8    4       1    0     1    2     0
  t07_odonnell_fg        5       4   2    1   5    0       1    1     0    3     0
  t09_higgins_lab        2       2   5    4   0    1       0    0     2    0     0
  t03_burton_lab         4       8  12   10   5    5       4    5     8   15     8
  t13_cuffe_green        1       2   0    0  11    0      16    3     0    3     1
  t08_gilmore_lab        4       8   7    4   3    6       4    5     1    2    11
  t02_bruton_fg          1      10   6    4   4    3       0    6    16    5     3
```

Quantifying texts

We'll be quantifying texts using the `{quanteda}` package in `R`

- `{quanteda}` was developed by a former LSE prof, Ken Benoit
- Offers a seamless package for doing a lot of QTA tasks.

```r
install.packages("quanteda")
install.packages("quanteda.textplots")
install.packages("quanteda.textstats")
```

There are other tools in python, probably more common in industry

Document selection

A **document** is the fundamental unit of analysis for quantitative text analysis, for exampe:

- n-word sequences
- Sentences
- Pages
- Paragraphs
- Natural units (a speech, a poem, a manifesto)
- Aggregation of units (e.g. all speeches by party and year)

A collection of documents is called a **corpus**

How you define documents depends on your goal

Side note on terminology

Let’s keep some words straight for this lecture:

- a **document** is the chosen unit of analysis; could be a full “document” in colloquial sense, or not
- a **text** (used as a countable noun) is what we often refer to as a “document” in the colloquial sense, e.g. a novel, a speech, a legal opinion, a tweet, etc.
- **text** (used as an uncountable noun) is a general word used to label a type of data, similar to “string”

I’ll try to be consistent, but context will usually be informative

Example: US President Trump’s tweets

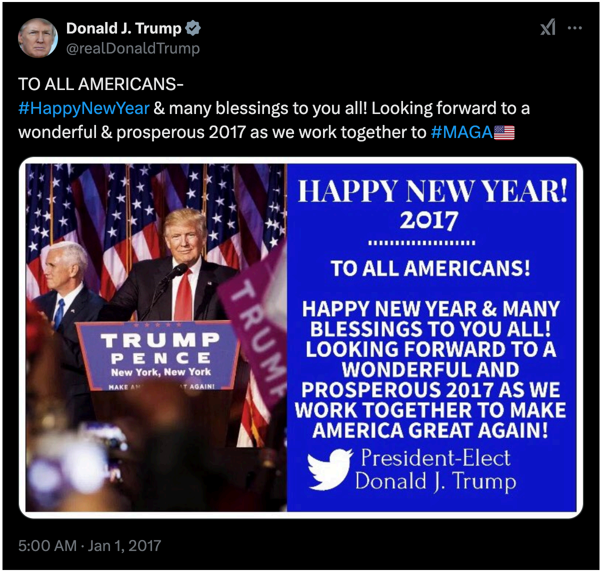
To demonstrate quantifying texts, we’ll use a corpus of US President Trump’s tweets

- The corpus covers 2017 and half of 2018
- Available on the course website

Data is stored in a `.json` file based on the Twitter API

- Convenient `{streamR}` package to parse JSON in the Twitter API format

Example: US President Trump’s tweets



Donald J. Trump
@realDonaldTrump

TO ALL AMERICANS-
#HappyNewYear & many blessings to you all! Looking forward to a wonderful & prosperous 2017 as we work together to #MAGA🇺🇸

HAPPY NEW YEAR!
2017

TO ALL AMERICANS!

HAPPY NEW YEAR & MANY BLESSINGS TO YOU ALL! LOOKING FORWARD TO A WONDERFUL AND PROSPEROUS 2017 AS WE WORK TOGETHER TO MAKE AMERICA GREAT AGAIN!

President-Elect
Donald J. Trump

5:00 AM · Jan 1, 2017

Source: https://x.com/realDonaldTrump/status/815422340540547073

Example: US President Trump’s tweets

```
{ "created_at": ["Sun Jan 01 05:00:10 +0000 2017"],
  "id": [8.15422340540547e+17],
  "id_str": ["815422340540547073"],
  "full_text": ["TO ALL AMERICANS-\n#HappyNewYear &amp; many blessings to
               you all! Looking forward to a wonderful &amp; prosperous
               2017 as we work together to #MAGA\U0001F1FA\U0001F1F8
               https://t.co/UaBFaoDYHe"],
  "truncated": [false],
  "display_text_range": [[0],[148]],
  "entities": {"hashtags": [{"text":["HappyNewYear"],"indices":[[18],[31]]},
                            {"text":["MAGA"], "indices":[[141],[146]]}],
               "symbols": [],
               "user_mentions":[],
               "urls":[],
               "media": [{"id":[8.15422333510816e+17],
                          "id_str":["815422333510815746"],
                          "indices":[[149],[172]],
                          "media_url":["http://pbs.twimg.com/media/C1D2SsLVE
                          "media_url_https":["https://pbs.twimg.com/media/C1
                          "url":["https://t.co/UaBFaoDYHe"],
                          ...
```

Defining document features

Documents contain **features**, which can be:

- characters
- words
- word “stems” or “lemmas” (more later)
- word segments, especially for languages using compound words, such as German, e.g. *Saunauntensitzer*
- “word” sequences, especially when inter-word delimiters (usually white space) are not commonly used, e.g., in Chinese
- linguistic features, such as parts of speech
- coded or annotated text segments
- word embeddings

The most common approach: bag of words

Most common approach to quantifying text: **bag of words** model

- ▷ Single *words* are the relevant *features* of each document
- ▷ Documents are quantified by counting occurrences of words
- ▷ Word order does not matter
- ▷ Discards grammar and syntax

Why bag of words?

Most obvious reason: it's very simple

But also, context is often uninformative and conditional on *presence* of words

- ▷ Individual word usage tends to be associated with a particular degree of affect, position, etc. without regard to context
- ▷ So presence of words by itself captures info about context

Plus, *single* words tend to be the most informative since co-occurrences of multiple words ("n-grams") are relatively rare

In social science applications: bag of words works well most of the time (i.e., it's been validated *a lot*)

Why bag of words?

There are times where word order is important, e.g.

- ▷ **Text reuse**: plagiarism detection software used for social science applications, such as Corley (2007) and Grimmer (2010)
- ▷ **Parts of speech tagging**: tagging words in documents with grammatical information, with applications such as Bamman and Smith (2014) and Handler et al (2016)
- ▷ **Named entity recognition (NER)**: finding and tagging words in documents that are people, organisations, or places, with applications such as Copus, Hübert and Pellaton (2024)

As usual: whether word order matters depends on the QTA task

Implementing bag of words

Goal: get from a set of texts to a quantitative dataset

There is a basic four step process for implementing bag of words to quantify texts:

1. Choose unit of analysis
2. Tokenise
3. Reduce complexity
4. Create **document feature matrix**

People often refer to this as **preprocessing**

The “reducing complexity” part is where most of the discretion is

You should justify the preprocessing steps you take

Tokenising

You start by **tokenising** each document:

- ▷ Split into an array of words, each called a **token**
- ▷ For English (and many other languages), use white space; trickier for logographic languages (e.g. Chinese)
- ▷ Tokens are *mostly* “words”—but not always

A list of all the distinct tokens used in an entire corpus is a **vocabulary**

- ▷ Each element of the vocabulary is a **type**

Tokenising Trump’s tweet

Original (plain) text of Trump tweet:

```
TO ALL AMERICANS-\n#HappyNewYear &amp; many blessings
to you all! Looking forward to a wonderful &amp;
prosperous 2017 as we work together to
#MAGA\U0001F1FA\U0001F1F8 https://t.co/UaBFaoDYHe
```

Tokenising Trump’s tweet

Use white space to tokenise:

```
c("TO", "ALL", "AMERICANS-", "#HappyNewYear", "&amp;",
  "many", "blessings", "to", "you", "all!", "Looking",
  "forward", "to", "a", "wonderful", "&amp;",
  "prosperous", "2017", "as", "we", "work", "together",
  "to", "#MAGA", "https://t.co/UaBFaoDYHe")
```

Notice some issues here:

▷ some useless punctuation: “AMERICANS-”

▷ some “non-words” included: links, ampersands

▷ the words “to” and “TO” are considered different words

This will create a vocabulary that is too large and redundant

Reduce complexity: cleaning up formatting

To deal with these problems, we can: remove “non-words” and punctuation, and make text lowercase

```
c("to", "all", "americans", "#happynewyear", "many",
  "blessings", "to", "you", "all", "looking", "forward",
  "to", "a", "wonderful", "prosperous", "as", "we",
  "work", "together", "to", "#maga")
```

Note:

- Had to manually get rid of symbols written in HTML syntax (e.g. `&amp;`) using regex pattern `"[&][#]?[A-z]+;"`

- Decided to keep Twitter hashtags intact (why?)

Reduce complexity: removing stop words

Stop words are words that occur very frequently in a language but do not provide much information

Some very common English stop words are:

```
a, able, about, across, after, all, almost, also, am, among, an, and,
any, are, as, at, be, because, been, but, by, can, cannot, could,
dear, did, do, does, either, else, ever, every, for, from, get, got,
had, has, have, he, her, hers, him, his, how, however, I, if, in,
into, is, it, its, just, least, let, like, likely, may, me, might,
most, must, my, neither, no, nor, not, of, off, often, on, only, or,
other, our, own, rather, said, say, says, she, should, since, so,
some, than, that, the, their, them, then, there, these, they, this,
tis, to, too, twas, us, wants, was, we, were, what, when, where,
which, while, who, whom, why, will, with, would, yet, you, your
```

In QTA, it is very common to remove stop words

▷ But this depends on your task, see Pennebaker (2011)

▷ There are different lists out there, some longer, some shorter

Removing stop words from Trump’s tweet

We can remove all the stop words (defined by the list above) from the Trump tweet:

```
c("americans", "#happynewyear", "many", "blessings",
  "looking", "forward", "wonderful", "prosperous",
  "work", "together", "#maga")
```

Reduce complexity: stemming and lemmatisation

Many words have multiple “forms” with different spellings, e.g.

- Tenses: “I see” and “I saw”
- Pluralisation: “family” and “families”
- Contractions: “families” and “family’s”

So far, we have treated all these as different words

But, you may wish to create **equivalence classes** of words considered to convey same meaning

- Basic idea: for every token, identify the “root” word, and replace the token with root word
- This reduces vocabulary, and can be quite useful for comparing across documents

You may or may not want to do this depending on your task

Reduce complexity: stemming and lemmatisation

Two common approaches:

1. **Lemmatisation**: refers to the algorithmic process of converting words to their lemma
   - ▷ A word's **lemma** is its "canonical form"

2. **Stemming**: removing the ends of words using a set of rules

Basic difference: stemmers operate on single words without knowledge of the context, lemmatisers are more powerful

- ▷ There is a computational tradeoff

Example: `production, producer, produce, produces, produced` all replaced with `produc`

Tools for each available in `{quanteda}` package

Reduce complexity: stemming and lemmatisation

Lots of different ways to stem and lemmatise

Porter stemmer is most common, but gets many stems wrong:

- ▷ `policy` and `police` considered (wrongly) equivalent
- ▷ `general` becomes `gener`, `iteration` becomes `iter`

There are other corpus-based, statistical, and mixed approaches designed to overcome these limitations

Plus, sometimes stemming isn't appropriate: Schofield and Mimno (2016) find that "stemmers produce no meaningful improvement in likelihood and coherence (of topic models) and in fact can degrade topic stability"

Take away: you have to read and validate!

Reduce complexity: stemming and lemmatisation

We can use `{quanteda}`'s default stemming function `tokens_wordstem()` on the Trump tweet, yielding:

```
c("american", "#happynewyear", "mani", "bless", "look",
  "forward", "wonder", "prosper", "work", "togeth",
  "#maga")
```

Note:

- `{quanteda}` uses the Snowball stemmer by default
- The stemmer did many transformations, e.g.:
  - `americans` → `american`
  - `many` → `mani`
  - `blessing` → `bless`
  - `together` → `togeth`

The document-feature matrix (DFM)

So far, we've just "preprocessed" one example document

You repeat this process for every document

- This yields a vocabulary of types for the corpus
- These are the *features* to be analysed (again: remember we're assuming bag of words!)

Each document uses some of the features in the vocabulary

- You can count how many times

A **document-feature matrix** (math: $\mathbf{W}$) is a matrix of N documents (rows) by J features (columns) where:

- each W_{ij} counts the number of times the jth feature appears in the ith document

The document-feature matrix (DFM)

All of Trump's tweets from January 2017:

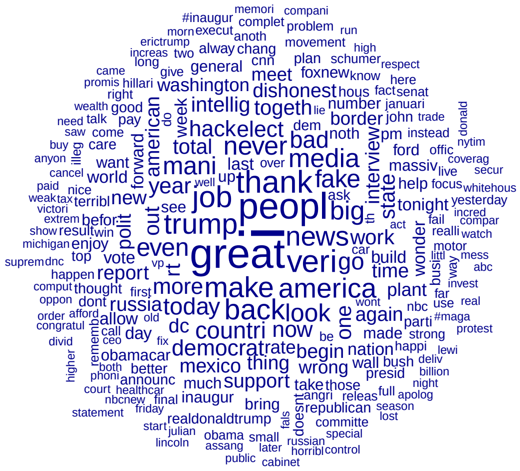
```
Document-feature matrix of: 212 documents, 1,039 features (98.94%
  sparse) and 0 docvars.
                     features
docs                  american #happynewyear mani bless
  2017-01-01 05:00:10        1             1    1     1
  2017-01-01 05:39:13        0             1    0     0
  2017-01-01 05:43:23        0             0    0     1
  2017-01-01 05:44:17        0             0    0     0
  2017-01-01 06:49:33        0             0    0     0
[ reached max_ndoc ... 207 more documents, reached max_nfeat ...
  1,035 more features ]
```

This DFM has $J = 1,039$ features and $N = 212$ documents

- ▷ Each cell is a count, e.g. $W_{11} = 1$ and $W_{21} = 0$

Wordclouds

Wordclouds are basic visualisations of the data in a DFM



Outline

Not all tokens *should* be unigrams

When we tokenise using white spaces, we create **unigrams**

We could define tokens differently by choosing a **collocation**:

- ▷ **bigrams**: pairs of adjacent words
- ▷ **trigrams**: triples of adjacent words
- ▷ more generally: n**-grams**: n adjacent words

Example: `capital gains tax` can be represented as

- ▷ unigrams: `c("capital", "gains", "tax")`
- ▷ bigrams: `c("capital gains", "gains tax")`
- ▷ trigrams: `c("capital gains tax")`

Not all tokens *should* be unigrams

Why would you want to depart from unigrams?

1. You might want to do your entire analysis using n-grams instead of unigrams
   - ▷ It could be useful for situations where word order matters like simple text completion

2. Many collocations have independent meaning that you might want to retain in your analysis
   - ▷ For example, `United Kingdom` makes more sense left as a bigram than as two unigrams
   - ▷ In cases like this: need to manually define which phrases to leave as n-grams

How do you decide which words to collocate into n-grams?

Important collocations

$C(w^1\ w^2)$	w^1	w^2
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the
13183	with	the
12622	from	the
11428	New	York
10007	he	said
9775	as	a
9231	is	a
8753	has	been
8573	for	a

Table 5.1 Finding Collocations: Raw Frequency. $C(\cdot)$ is the frequency of something in the corpus.

Source: Manning and Schütze (ch. 5)

Important collocations

$C(w^1\ w^2)$	w^1	w^2
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the
13183	with	the
12622	from	the
11428	New	York
10007	he	said
9775	as	a
9231	is	a
8753	has	been
8573	for	a

Table 5.1 Finding Collocations: Raw Frequency. $C(\cdot)$ is the frequency of something in the corpus.

Source: Manning and Schütze (ch. 5)

Identifying collocations in `{quanteda}`

Find frequent collocations with: `textstat_collocations()`

Let’s find the common bigrams in the Trump tweet corpus:

```
# A tibble: 5,335 x 6
   collocation   count count_nested length lambda     z
   <chr>         <int>        <int>  <dbl>  <dbl> <dbl>
 1 fake news       217            0      2   8.01  40.1
 2 tax cut         130            0      2   7.11  35.8
 3 make america     99            0      2   5.60  35.6
 4 unit state      101            0      2   7.25  30.6
 5 north korea     116            0      2   9.32  30.0
 6 news media       63            0      2   5.02  28.7
 7 america great    91            0      2   4.00  28.5
 8 great again      98            0      2   4.75  28.2
 9 illeg immigr     39            0      2   6.56  26.1
10 work hard        46            0      2   5.42  25.9
# i 5,325 more rows
```

Word embeddings

Bag of words: each word has a distinct, unique meaning

- ▷ In math terms: we treat words as **one-hot encodings**

Concrete example: consider a vocabulary of three words:
`c("cat", "dog", "rat")`

Can represent each word in **vector format**:

- ▷ The word `cat` is `(1,0,0)`
- ▷ The word `dog` is `(0,1,0)`
- ▷ The word `rat` is `(0,0,1)`

(Should be obvious why this is called “one-hot encoding”)

These are *orthogonal*: zero similarity between the words

Word embeddings

But what if words are actually representations of a smaller group of concepts, where each word is a *mix* of concepts?

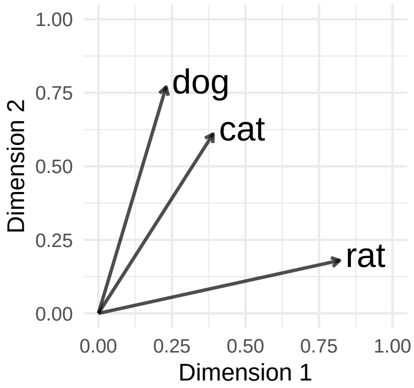
E.g., what if these words can be represented in a two dimensional **embedding** (e.g., two distinct “concepts”)

Then, you might have

- ▷ The word `cat` represented by `(0.39,0.61)`
- ▷ The word `dog` represented by `(0.23,0.77)`
- ▷ The word `rat` represented by `(0.82,0.18)`

Word embeddings have to be *estimated* from a corpus

Word embeddings



Machine “learns” these dimensions from patterns in corpus

Analyst interprets their meaning

- ▷ Dimension 1 might represent something like “wildness” or “pest-like” traits.
- ▷ Dimension 2 might represent “domesticated companionship.”

You can measure the similarity of words using concepts from linear algebra, like **cosine similarity**

- ▷ Roughly: vectors pointing in same direction are very “similar”
- ▷ One-hot encodings are all perfectly dissimilar using this metric

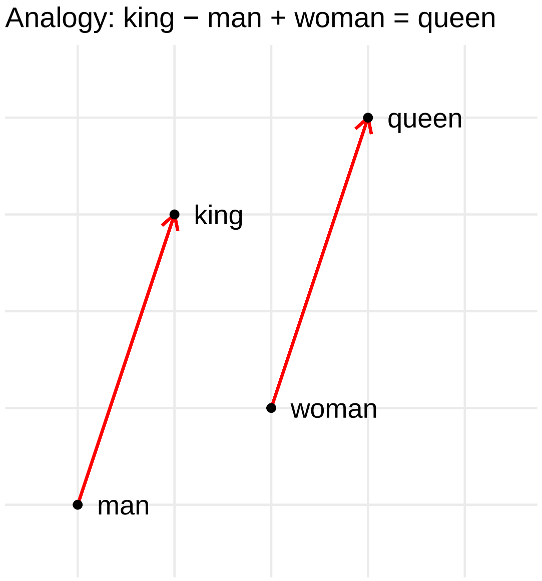
Word embeddings

Main purpose: use word embeddings as document features instead of words… but why?

1. Technical reason: reduce DFM dimensionality/sparsity
   - ▷ ML models perform better with “denser” vectors

2. Allows for correlations between genuinely related words
   - ▷ One-hot encodings: “dog” and “puppy” are as unrelated as “dog” and “refrigerator”
   - ▷ Embeddings: words like “dog”, “puppy”, “canine” will have similar vectors
   - ▷ Models can generalise better from the data
   - ▷ Capture linguistic relationships through “analogies,” like: king - man + woman $\approx$ queen

Word embeddings



Outline

XML

XML = e**X**tensible **M**arkup **L**anguage

- ▷ XML has similar structure as HTML, but:
  - ○ XML itself has almost no predefined, domain-specific tags
  - ○ Author can invent XML tags to structure document
- ▷ XML is mostly used to store and distribute data, often as an alternative to JSON (which also has nested structure)
- ▷ HTML is used for *displaying* data (as you know)

Often used to store *textual* data

Reference and further information:
`https://www.w3schools.com/xml/xml_whatis.asp`

XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<courses>
    <course>
        <title>Data for Data Scientists</title>
        <code>MY472</code>
        <year>2024</year>
        <term>Autumn</term>
        <description>A course about collecting, processing,
                     and storing data.</description>
    </course>
    <course>
        <title>Computer Programming</title>
        <code>MY470</code>
        <year>2024</year>
        <term>Autumn</term>
        <description>An introduction to programming.</description>
    </course>
</courses>
```

Steps in XML parsing in R

1. Parse an XML file with `read_xml()` in `{xml2}` package
2. Select elements with `html_elements()` or `xml_find_all()`
3. Extract text with `html_text()` or `xml_text()`

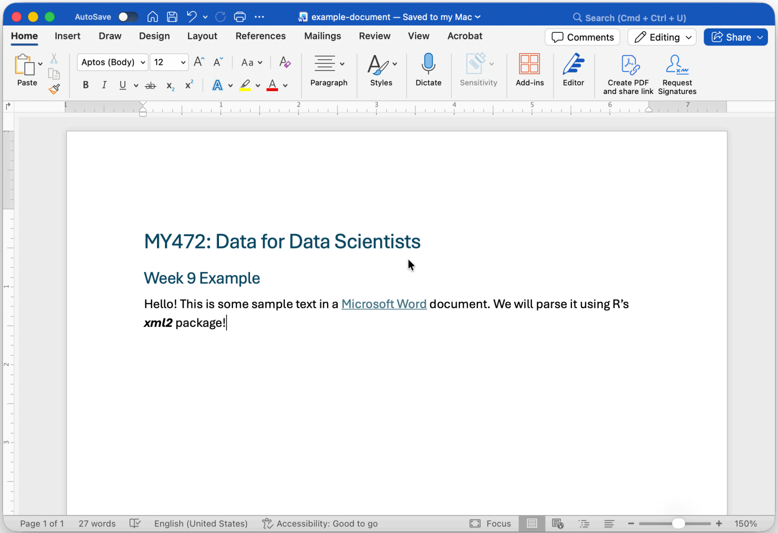
Working with XML in R is nearly identical to working with HTML (as we covered in week 7)

- Just use `{xml2}` to read XML files, and then use `{rvest}` to navigate and parse the tags

Some XML use cases

- API data, e.g., Google Maps Geocoding API
- Data storage, e.g. Canadian members of parliament: `https://www.ourcommons.ca/Members/en/search` (select “Export as XML” at the bottom)
- Scalable Vector Graphics (SVG), e.g. London borough map `https://upload.wikimedia.org/wikipedia/commons/b/be/BlankMap-LondonBoroughs.svg`
- ePub electronic books, e.g. Project Gutenberg books
- Office documents, e.g. OpenOffice, Microsoft Office
  - Notice difference between `.doc` and `.docx`
- RSS web feeds, e.g. `http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907`

XML example: MS Word documents


AutoSave
example-document — Saved to my Mac
Search (Cmd + Ctrl + U)
Home Insert Draw Design Layout References Mailings Review View Acrobat
Comments Editing Share
Paste Aptos (Body) 12 Paragraph Styles Dictate Sensitivity Add-ins Editor Create PDF and share link Request Signatures
Page 1 of 1 27 words English (United States) Accessibility: Good to go Focus 150%

MY472: Data for Data Scientists

Week 9 Example

Hello! This is some sample text in a Microsoft Word document. We will parse it using R's ***xml2*** package!

XML example: MS Word documents

```r
library("rvest")
library("xml2")
word.path <- file.path(ddir, "example-document.docx")
unzip(word.path, exdir = ddir)
unzip(word.path, list = TRUE)
```

```
                           Name Length       Date
1           [Content_Types].xml   1312 1980-01-01
2                   _rels/.rels    590 1980-01-01
3             word/document.xml   3966 1980-01-01
4  word/_rels/document.xml.rels   1003 1980-01-01
5         word/theme/theme1.xml   8391 1980-01-01
6             word/settings.xml   3147 1980-01-01
7               word/styles.xml  43339 1980-01-01
8          word/webSettings.xml   1069 1980-01-01
9            word/fontTable.xml   1831 1980-01-01
10            docProps/core.xml    749 1980-01-01
11             docProps/app.xml    714 1980-01-01
```

XML example: MS Word documents

```r
word.doc <- read_xml(file.path(ddir, "word/document.xml"))
print(word.doc)
```

```
{xml_document}
<document Ignorable="w14 w15 w16se w16cid w16 w16cex w16sdtdh w16sd
[1] <w:body>\n  <w:p w14:paraId="264D80B7" w14:textId="61 ...
```

XML example: MS Word documents

```r
t <- html_elements(word.doc, xpath = "//w:p")
# t <- xml_find_all(word.doc, xpath = "//w:p") # identical
print(t)
```

```
{xml_nodeset (3)}
[1] <w:p w14:paraId="264D80B7" w14:textId="6130D9C8" w:rs ...
[2] <w:p w14:paraId="1B6C64AF" w14:textId="27DAEC61" w:rs ...
[3] <w:p w14:paraId="18DD7CC0" w14:textId="2D8C7BE6" w:rs ...
```

XML example: MS Word documents

```r
t <- html_text(t)
# t <- xml_text(t) # Identical
print(t)
```

```
[1] "MY472: Data for Data Scientists"
[2] "Week 9 Example"
[3] "Hello! This is some sample text in a Microsoft Word document. \
```

XML example: MS Word documents

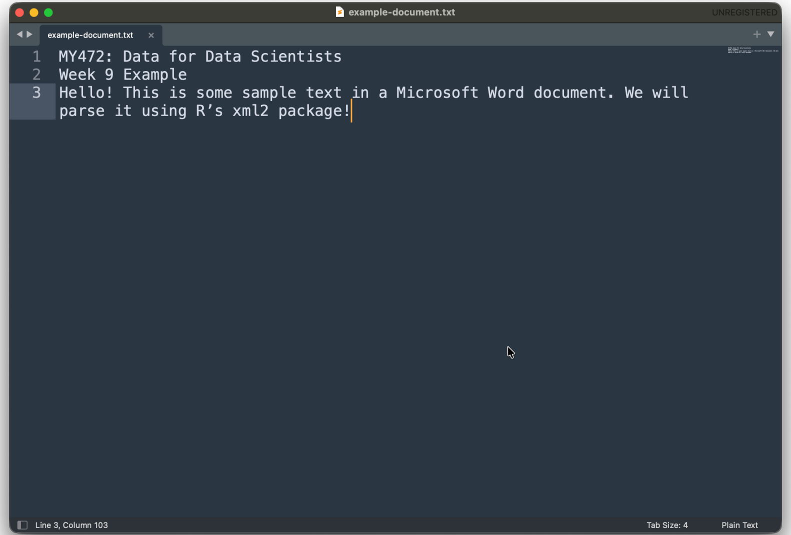
```r
t <- t |>
  paste0(collapse = "\n")
cat(t)
```

```
MY472: Data for Data Scientists
Week 9 Example
Hello! This is some sample text in a Microsoft Word document. We wi
```

XML example: MS Word documents

```r
library("readr")
library("stringr")
write_file(t, str_replace(word.path, ".docx", ".txt"))
```

XML example: MS Word documents



```
MY472: Data for Data Scientists
Week 9 Example
Hello! This is some sample text in a Microsoft Word document. We will
parse it using R’s xml2 package!
```

Other sources of text

Text is available in many other formats, two important ones:

1. **Cloud based documents**: text that is available on a cloud-based editor, e.g., Google Docs

- ▷ These are usually plain text formats “under the hood”
- ▷ Often can be exported in some kind of plain text format
- ▷ There are often APIs to access text directly, such as the Google Docs API

2. **“Image-like” files**: text is human-readable, might be computer-readable; stores text as images rendered on a page, e.g., `.pdf`, `.jpg`, `.png`, `.tif`, etc.

Optical character recognition (OCR)

Image-like files: often how analog texts are digitized

- For example, scans of old paper documents

PDFs are most common image-like file type for storing text

- There are different types, see `https://pdf.abbyy.com/learning-center/pdf-types/`
- Some complications around document security

To extract text:

- PDFs that are digitally generated; can export text directly in e.g., Adobe Acrobat
- Otherwise, need **optical character recognition (OCR)**
- Seminar materials: R code for extracting text from PDFs

Outline

Data security in storage

When you get data from other people, you have an ethical obligation to store it and transfer it securely

- ▷ Especially if it is restricted access data, or it contains personally identifiable information

In academic contexts (e.g., as an LSE student), you *cannot* collect sensitive data without ethics approval

- ▷ LSE-specific information: `https://info.lse.ac.uk/staff/divisions/research-and-innovation/research/Research-integrity-at-LSE`

In UK (and most of Europe), bound by, for example GDPR

Let's look at some of the technical aspects of data security that can help you meet your obligation (but not guarantee it!)

Data security in storage

Absolute bare minimum: use aggressive authentication practices!

- ▷ Do not disable password/PIN protection
- ▷ Use complex passwords and longer PINs
- ▷ Devices should auto-lock after short period of inactivity
- ▷ Enable biometric authentication (fingerprint, facial recognition)
- ▷ Turn on security settings that protect your digital data if your device is lost or stolen
- ▷ Do not share passwords or user accounts

Two issues:

1. Data security **at rest**: on a storage device
2. Data security **in transit**: transferring over network

Encryption at rest

Best practice: encrypt digital data

- ▷ **Encryption** is the process by which “meaningful” digital data is transformed into a format that is “unmeaningful”
- ▷ In other words, encryption “locks” your data
- ▷ An **encryption key** is needed to “unlock” (or **decrypt**) encrypted data
- ▷ Core issues: who has the key? is the key secure?

You can encrypt your computer’s hard drive:

- ▷ On macOS, you turn on FileVault
- ▷ On Windows, you turn on BitLocker

(You should do this!)

Encryption at rest

Symmetric encryption is used to encrypt files

- ▷ There is *one* key that encrypts and decrypts
- ▷ If you want multiple people to be able to encrypt/decrypt, need to (securely) share the key

Asymmetric encryption is used to share keys

- ▷ Two keys: a public and private key
- ▷ User A's public key + user B's private key $\rightarrow$ **shared secret**
- ▷ B's public key + A's private key $\rightarrow$ *same* shared secret
- ▷ More relevant for encryption in transit (more later)

Encryption at rest

An encryption key is a sequence of randomly generated bytes

- $\triangleright$ More bytes $\rightarrow$ more secure because hackers need to make more guesses
- $\triangleright$ Key with 1 byte: can be guessed in less than 256 tries (2^8)

The key is used for encryption via an encryption algorithm

The modern standard is 256-bit AES, an algorithm that uses keys with 32 random bytes (8 $\times$ 32 = 256 random bits)

- $\triangleright$ nearly impossible to break by “brute force”

Encrypting and decrypting data in R

Can create 256-bit keys in R using `{openssl}` package

```r
library("openssl")

# Create a key: sequence of 32 random bytes
key <- rand_bytes(32)
# key <- aes_keygen(length = 32) # equivalent

key
```

```
 [1] 2f bb 0b 17 37 c7 7d 01 95 eb 13 31 b7 68 2e 2c 18 de
[19] 33 a4 14 f5 86 a5 29 16 84 49 c0 77 88 b0
```

Encrypting and decrypting data in R

If you plan to use this as an encryption key, you better not lose it!

This is a very insecure way to store it:

```
writeBin(key, "my_key.bin")
```

Saving to keychain is much, much better:

```
# Save key to keychain
library("keyring")
key_set_with_raw_value("test_enc_key", password = key)
# Delete key
rm(key)
```

But be careful: if someone has access to R on your computer, they may be able to get this key!

- ▷ In seminar, you'll learn how to lock keychains

Encrypting and decrypting data in R

To encrypt text using R (and a key), you first the text to raw bytes, then encrypt

```r
# What's my secret message
secret <- "A top secret message!"

# Convert to bytes
secret <- charToRaw(secret)

# Encrypt the message using key
secret <- aes_gcm_encrypt(secret,
                          key = key_get_raw("test_enc_key"))
rawToChar(secret) # Secret has been encrypted!
```

```
[1] "\xe6\xbd\"\022V\xe9\xb6▯\003\xe1\035,\xbbHiy\002)I"
```

Note: you can encrypt *any* digital data, not just text

Encrypting and decrypting data in R

To decrypt the message, you need the key

```
# Decrypt the message using key (pulled from keychain)
secret <- aes_gcm_decrypt(secret,
                          key = key_get_raw("test_enc_key"))
rawToChar(secret)
```

```
[1] "A top secret message!"
```

Encryption in transit

Lots of your digital data lives off your device

- ▷ E.g., cloud storage, emails, text messages, photos, etc.
- ▷ This is becoming even more common, e.g. “optimised” storage on Apple devices and default sync settings in Dropbox

You can control security of your local device, but not remote devices (the “cloud”) or network protocols

Bare minimum requirements for cloud storage:

- ▷ Encryption in transit using **Transport Layer Security (TLS)**
- ▷ Encryption at rest (on server) using AES 256-bit encryption

Reputable providers do this (often for legal reasons)

Encryption in transit

How does this apply to web browsing?

- ▷ `http` protocol does not use encryption in transit (yikes!), but `https` protocol does
- ▷ Do not put your sensitive information (like credentials) into a webpage with an `http://` url
- ▷ If you have a website, make sure it uses `https` or many browsers may block access to it

Private browsing does *not* protect your data in transit

- ▷ But it can be useful for other things: avoiding web tracking, limiting browsing history, etc.

Encryption in transit

Gold standard is **end-to-end (E2E) encryption**:

- ▷ Encrypt data on local device *before* transit
- ▷ Decrypt on remote device *after* transit
- ▷ Core challenge: how to share encryption key between sender and recipient

Examples: Signal, WhatsApp, many iCloud services, Apple's Advanced Data Protection

Most cloud storage services do not use/offer E2E encryption

- ▷ Dropbox does not use E2EE!
- ▷ Only major provider offering E2EE to individual consumers is Apple iCloud (but not in the UK…yikes!)

Encryption in transit

Still: hackers are always working to break encryption

- ▷ For example, recent reports of a trojan that can read decrypted Signal and WhatsApp messages on Android devices

Best practices:

- ▷ Regularly delete old messages and old files you no longer need
- ▷ Turn on auto-delete for your messaging apps (if available)
- ▷ Keep up to date on the security of the services you use
- ▷ Buy devices and use operating systems and software that are known to be more secure

Encryption in transit

You can set up a simple E2E encryption pipeline in R

You will see an example in seminar, but basic structure:

- Person A and Person B create **asymmetric keys** with public and private components
- Share the public keys with one another
- When A creates an encrypted message:
  - Use B's public key + A's private key to make a **shared secret**
  - Use shared secret to create a symmetric encryption key that is then used to encrypt the message
  - Send the encrypted message
- Person B uses their private key + A's public key to decrypt the symmetric key and decrypt the message

Data security: other considerations

So far, narrowly focused on technical aspects of digital data security assuming otherwise benevolent cloud providers

But companies might be accessing your data

- ▹ LLM providers like OpenAI are opaque about whether/how they use personal data you give them
- ▹ Gmail uses email data to personalise services
- ▹ Cloud storage providers have access to the information in your private files (except iCloud, if ADP is turned on)

Companies are sometimes vague or misleading

- ▹ E.g., WhatsApp chat backups do not use E2EE by default

To be fair: consumer demands are partly to blame

Encryption can be controversial

Encryption can be controversial

Some governments do not want individuals to strongly encrypt their digital data (e.g., the UK government and Apple devices)

So, traveling with encrypted data poses a risk

- You may be required to decrypt your digital data at borders
- Laws vary from jurisdiction to jurisdiction
- Your ethical obligations are not waived simply because you did not do your due diligence before transporting data

LSE's guidelines: `https://info.lse.ac.uk/staff/divisions/dts/assets/documents/policies/Guidelines-Encryption-Guidelines-v1-1.pdf`

What should you do?

What’s the take-away for you as a data scientist?

- ▷ Start by shoring up your digital security
- ▷ To be safest: never store sensitive data off your (fully encrypted) devices
- ▷ If you still need to use cloud services:
  - ○ Use services provided by your institution, that have gone through security and legal reviews
  - ○ For E2E encryption, use iCloud (with ADP enabled) or manually encrypt on using third-party software
- ▷ Spend some time reviewing policies
- ▷ Learn security practices of your service providers