Week 3: Quantifying Texts

POP77032 Quantitative Text Analysis for Social Scientists

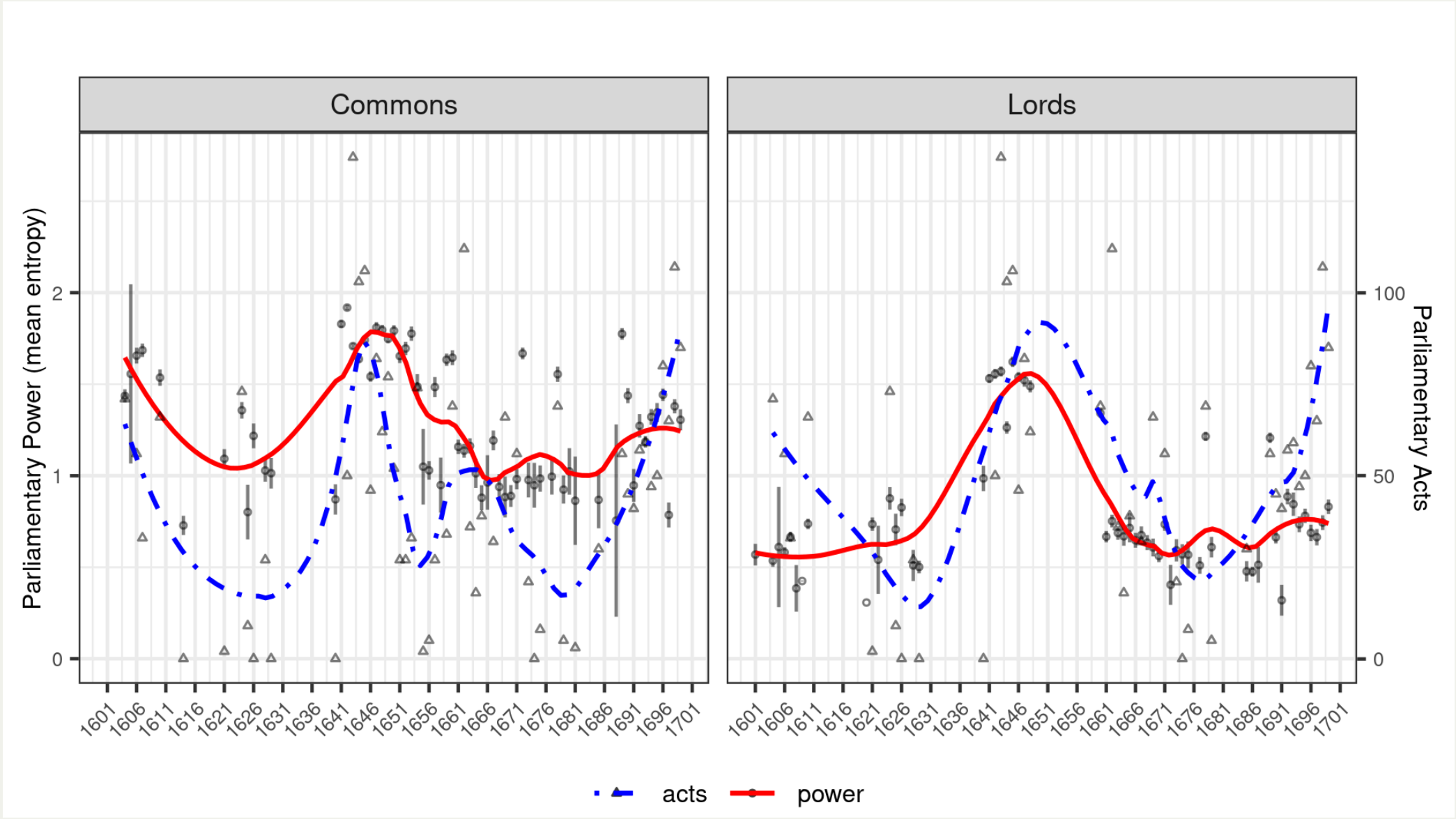
Tom Paskhalis

Overview

- Motivation
- Digital Text Storage
- Text Preprocessing
- APIs
- JSON

Motivation

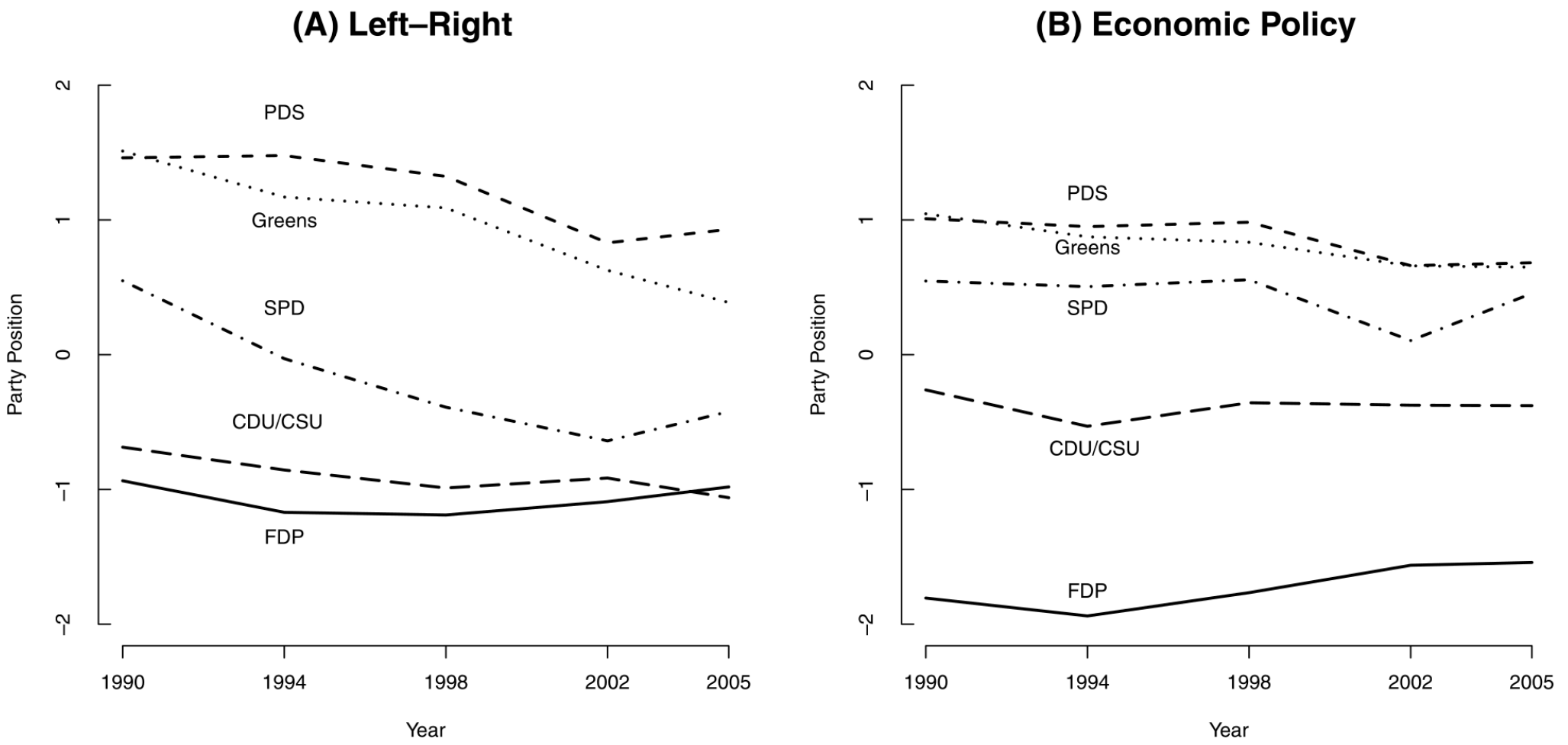
Parliamentary Power in 17th c. England



(Rodon & Paskhalis, 2024)

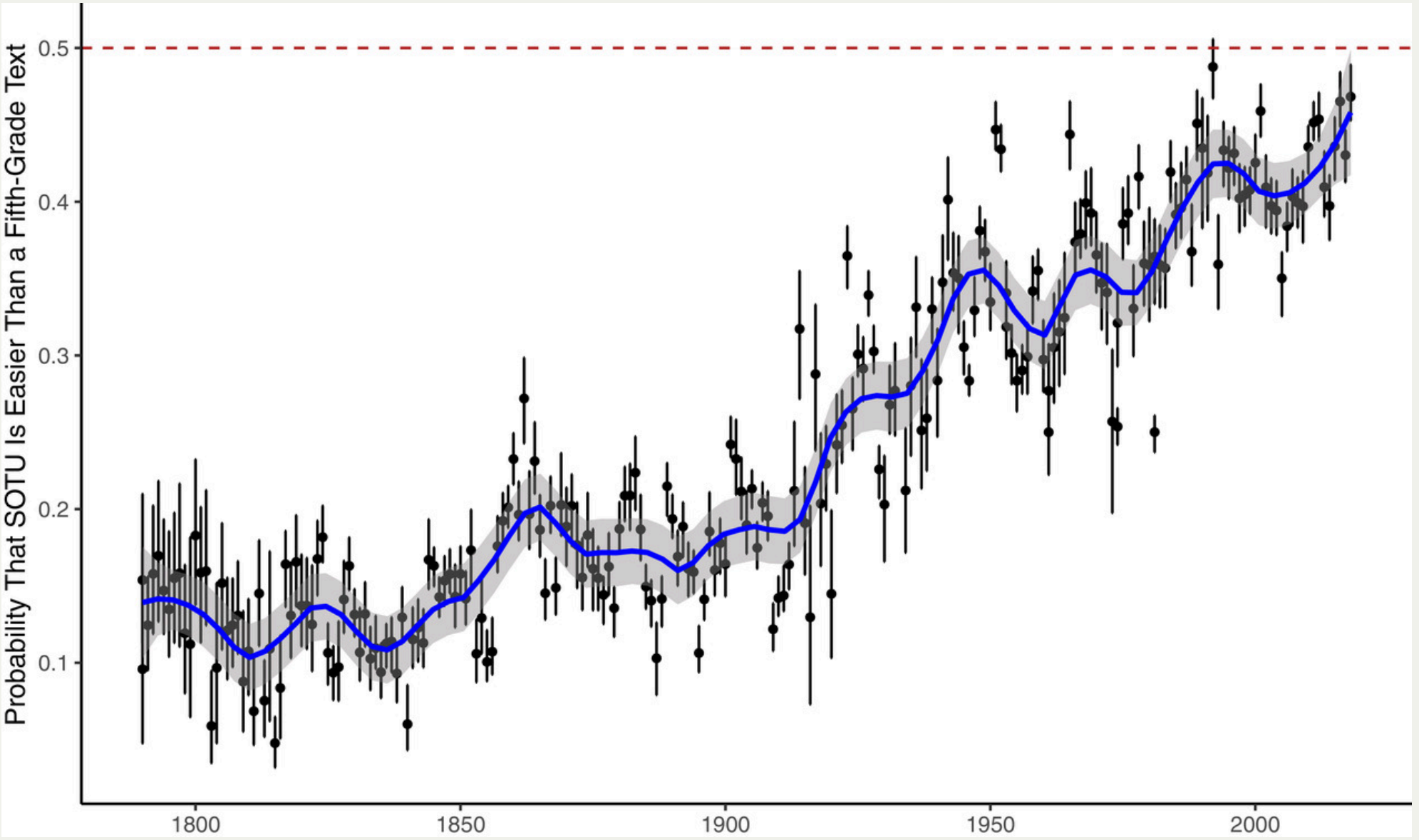
Ideological Positions in Germany

FIGURE 1 Estimated Party Positions in Germany, 1990–2005



(Slapin & Proksch, 2008)

Complexity of US State of the Union Addresses



(Benoit, Munger & Spirling, 2019)

Digital Storage of Text

Plain Text & Binary Files

- **Plain text** files contain only human-readable characters.
  - "Simple" text, e.g. `.txt`
  - Markup languages, e.g. `.md`, `.Rmd`, `qmd`, `.html`
  - Data storage, e.g. `.csv`, `.tsv`, `.tab`, `.json`, `.xml`
  - Images, e.g. `.svg`, `.eps`
  - Computer code, e.g. `.py`, `.R`, `.tex`, `.sh`
  - Some other: `.ipynb` (effectively, `.json`), `.docx` (effectively, zipped `.xml`)
- **Binary** files contain computer-readable data (parts can be human-readable).
  - Text, e.g. `.doc`, `.rtf`, `.pdf`
  - Data storage, e.g. `.pickle`, `.rds`, `.feather`
  - Images, e.g. `.png`, `.jpg`, `.gif`
- Not always dichotomous (e.g. `.docx`, `.pdf`, `.svg`).

Text Encoding Recap

- All text files stored in digital form are represented as numbers.
- These numbers correspond to certain code points,
- Which are values assigned to characters from some set.
- *Encoding* is then the mapping between characters and code points.
- Unicode (particularly, UTF-8) is the most widely used encoding.
- It provides representations for the vast majority of writing systems.

Text Encoding Caveats

- Plain text files don't contain information about encoding.
- Instead, each software "guesses" (often, assumes the default).
- If the guess is wrong, text can be displayed incorrectly (*mojibake*).
- UTF-8 is the most common encoding (the one you should use).
- However, many texts still use other encodings.
- Windows is often a problem (can use `Windows-1252` or `UTF-16`).
- In general, no easy way to know the encoding of a text file.

Text Encoding: Example

Write out text using Python in `ISO-8859-1` encoding.

```python
tain_bo_cualinge = "Fecht n-óen do Ailill & do Meidb íar ndérgud a rígleptha dóib i Crúachanráith Chonnacht,

with open("../temp/latin1.txt", "w", encoding = "ISO-8859-1") as f:
    f.write(tain_bo_cualinge)
```

```
130
```

Read in text in R using the default (`UTF-8`) encoding.

```r
tain_bo_cualinge <- readLines("../temp/latin1.txt")
tain_bo_cualinge
```

```
[1] "Fecht n-\xf3en do Ailill & do Meidb \xedar nd\xe9rgud a r\xedgleptha d\xf3ib i Cr\xfaachanr\xe1ith
Chonnacht, arrecaim comr\xe1d chind cherchailli eturru."
```

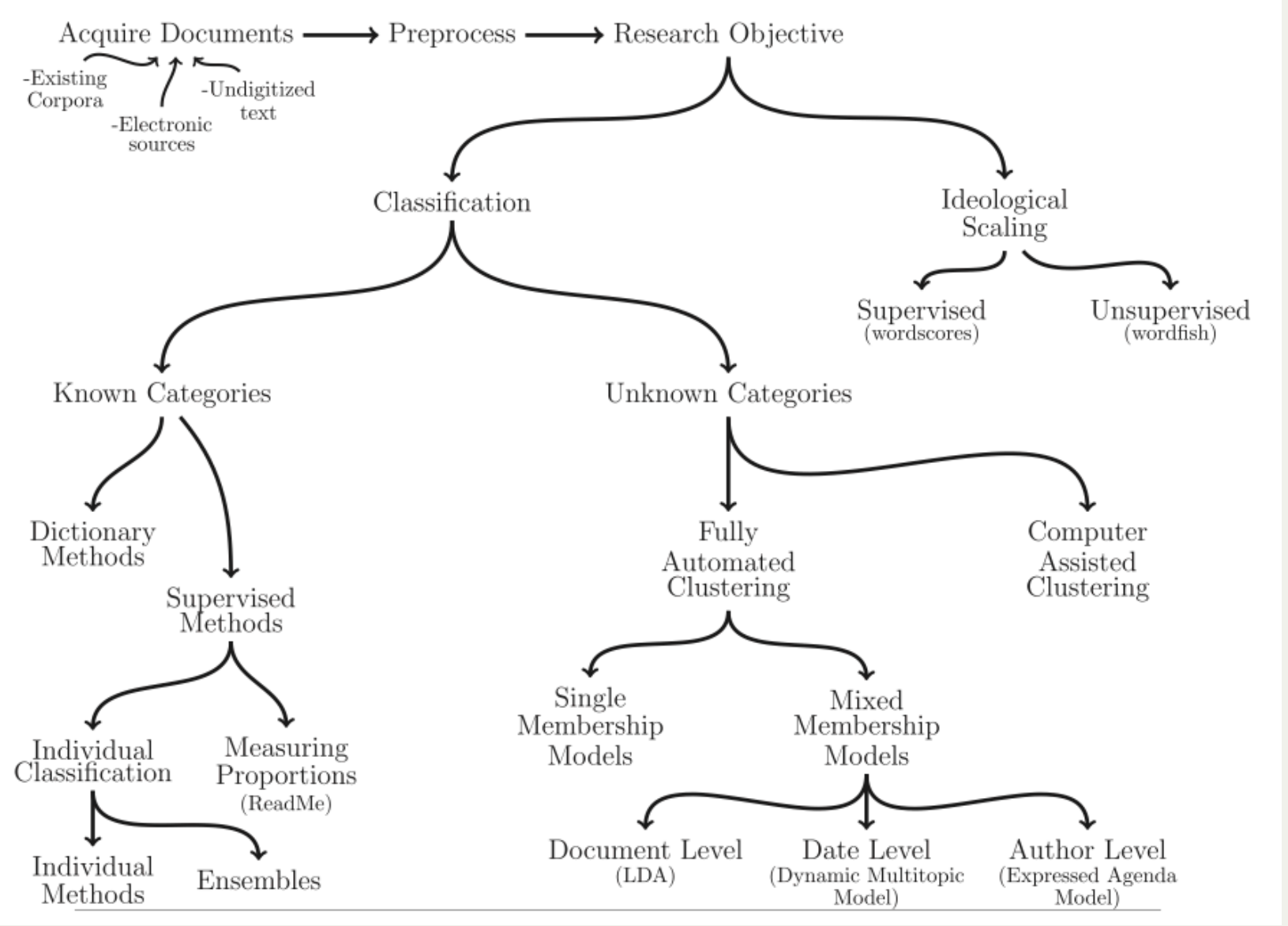
Using `ISO-8859-1` (note the difference in the encoding name).

```r
tain_bo_cualinge <- readLines("../temp/latin1.txt", encoding = "latin1")
tain_bo_cualinge
```

```
[1] "Fecht n-óen do Ailill & do Meidb íar ndérgud a rígleptha dóib i Crúachanráith Chonnacht, arrecaim comrád
chind cherchailli eturru."
```

Designing a Text Analysis Study

Workflow in Text Analysis



(Grimmer & Stewart, 2013)

Sample vs Population

- Basic Idea: Observed text is a stochastic realization.
- Systematic features shape most of observed verbal content.
- Non-systematic, random features also shape verbal content.

Implications of a Stochastic View

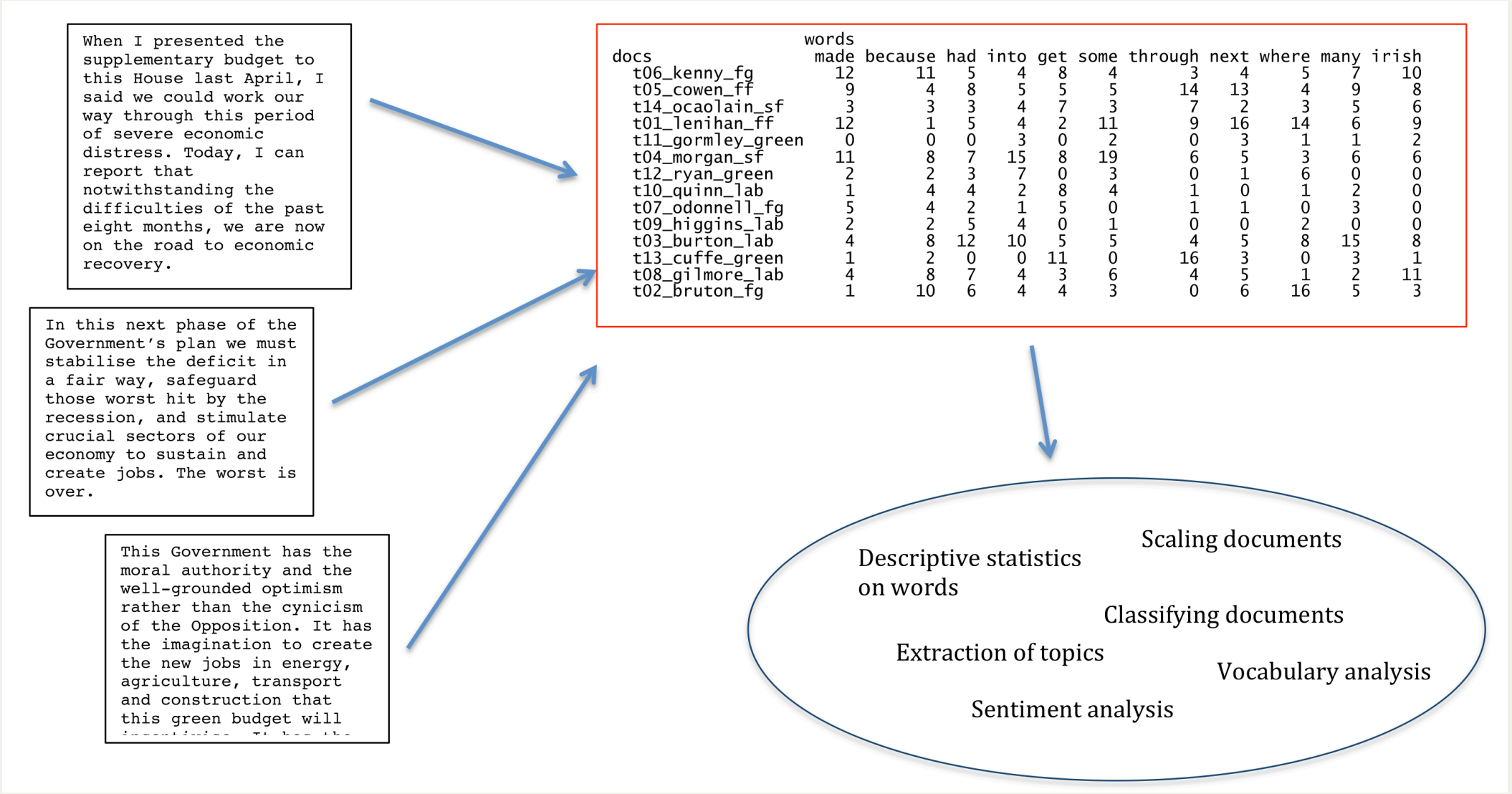
- Observed text is not the only text that could have been generated.
- Research (system) design would depend on the question and quantity of interest.
- Very different if you are trying to monitor something like hate speech, where what you actually say matters, not the value of your “expected statement”.
- Means that having “all the text” is still not a “population”.

Sampling Strategies

- Be clear what is you *sample* and your *population*.
- May not be feasible to perform any sampling.
- Different types of sampling vary from random to purposive:
  - Random sampling (e.g. politician's speeches)
  - Non-random sampling (e.g. messages containing hate speech on a social media platform)
- Key is to make sure that what is being analyzed is a valid representation of the phenomenon as a whole - a question of research design.

Text Preprocessing

Quantifying Texts



docs	words 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

Some QTA Terminology

- **Corpus** - a collection of texts for analysis.
  - E.g. SOTU addresses, Hansard debates, party manifestos, etc.
- **Document** - a single text in the corpus.
  - E.g. a single SOTU address, one speech, a specific party manifesto, etc.
- **Token** - a single unit of text.
  - E.g. typically a word, but can include punctuation, numbers, hashtags, etc.
- **Type** - a unique token.
  - E.g. articles like “the” and “a” appearing throughout the corpus.
- **Tokenization** - the process of breaking a text into tokens.

Some Linguistic Terminology

- Tokens constitute the basic unit of analysis (particularly in NLP applications).
- But how tokens are constructed can vary.
- It might be useful to consider different tokens as the same.
  - E.g. “runs”, “running”, “ran” are all forms of the same word.
- **Stemming** - mechanically removing affixes (usually, suffixes) from tokens.
  - E.g. “running” -> “run”, “runs” -> “run”.
- **Lemmatization** - reducing tokens to their base (root) or dictionary form.
  - E.g. “ran” -> “run”, “runner” -> “run”.
- While lemmatization is more accurate, it also requires more built-in knowledge about a language.

Tokenization: R Example

```r
library("quanteda")
```

```r
text <- "Hohohoho, Mister Finn, you're going to be Mister Finnagain!"
```

```r
tokens <- quanteda::tokens(text)
tokens
```

```
Tokens consisting of 1 document.
text1 :
 [1] "Hohohoho"  ","         "Mister"    "Finn"      ","         "you're"
 [7] "going"     "to"        "be"        "Mister"    "Finnagain" "!"
```

```r
quanteda::ntoken(tokens)
```

```
text1
   12
```

```r
quanteda::ntype(tokens)
```

```
text1
   10
```

```r
tokens <- quanteda::tokens_tolower(tokens)
quanteda::ntoken(tokens)
```

```
text1
   12
```

```r
quanteda::ntype(tokens)
```

```
text1
   10
```

```r
tokens_stemmed <- quanteda::tokens_wordstem(tokens, language = "english")
tokens_stemmed
```

```
Tokens consisting of 1 document.
text1 :
 [1] "hohohoho"  ","         "mister"    "finn"      ","         "you'r"
 [7] "go"        "to"        "be"        "mister"    "finnagain" "!"
```

Tokenization: Python Example

```
1 import transformers
```

```
1 text = "Hohohoho, Mister Finn, you're going to be Mister Finnagain!"
```

```
1 basic_tokenizer = transformers.BasicTokenizer()
2 basic_tokens = basic_tokenizer.tokenize(text)
3 basic_tokens
```

```
['hohohoho', ',', 'mister', 'finn', ',', 'you', "'", 're', 'going', 'to', 'be', 'mister', 'finnagain', '!']
```

```
1 gpt2_tokenizer = transformers.GPT2Tokenizer.from_pretrained("gpt2")
2 gpt2_tokens = gpt2_tokenizer.tokenize(text)
3 gpt2_tokens
```

```
['H', 'oh', 'oh', 'oho', ',', 'ĠMister', 'ĠFinn', ',', 'Ġyou', "'re", 'Ġgoing', 'Ġto', 'Ġbe', 'ĠMister',
'ĠFinn', 'again', '!']
```

```
1 len(basic_tokens)
```

```
14
```

```
1 len(gpt2_tokens)
```

```
17
```

```
1 len(set(basic_tokens))
```

```
12
```

```
1 len(set(gpt2_tokens))
```

```
13
```

Stopwords

- Not all words can be assumed to be equally informative.
- E.g. “the”, “a”, “and”, etc. are common in most texts.
- **Stopwords** are words that are removed from the text before analysis.

```
1 library("stopwords")
```

```
1 stopwords::stopwords(language = "en")
```

```
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
 [51] "does"       "did"        "doing"      "would"      "should"
 [56] "could"      "ought"      "i'm"        "you're"     "he's"
 [61] "she's"      "it's"       "we're"      "they're"    "i've"
 [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
 [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
 [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
 [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
 [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
 [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
 [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
```

API

APIs

- **API** - **A**pplication **P**rogramming **I**nterface.
- Programmatic way to interact with a software application/service.
- Widely used in computing (even on a single machine).
- Here, focus on web APIs, that provide interface to web services.
- A set of structured HTTP/S requests returns some responses.
  - E.g. in XML or JSON format.

APIs vs Web Scraping

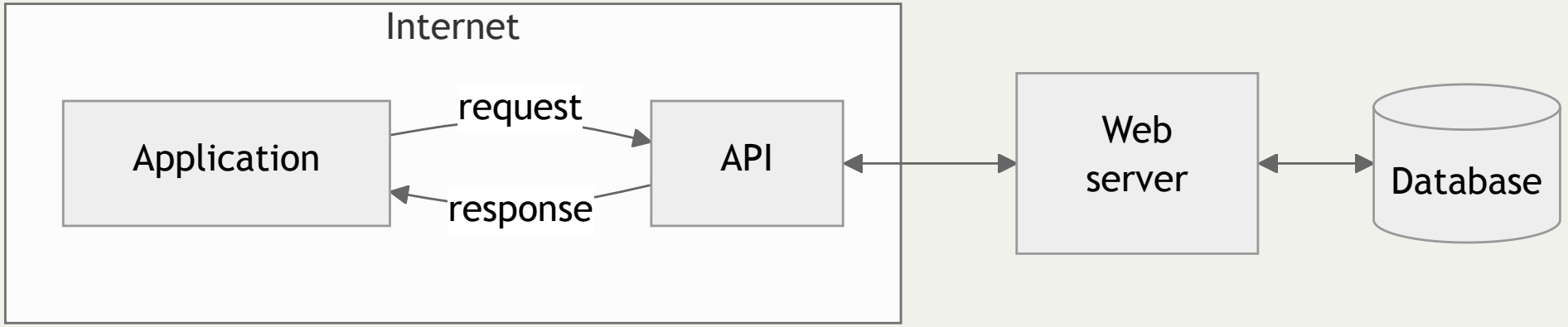
Advantages:

- Cleaner data collection: no malformed HTML, consistency, fewer legal issues, etc.
- Standardised data access processes.
- Scalability.
- Potentially, pre-existing robust packages for handling common tasks.

Disadvantages:

- Limited availability.
- Dependency on API providers.
- Access limits
- Rate limits.
- Price

Principles of APIs



Working with Web APIs

- Types of APIs:
  - **RESTful APIs** - queries for static information at a given moment,
  - **Streaming APIs** - tracking real-time changes (e.g. posts, economic indicators, etc.)
- API documentation varies by provider.
- But usually is rather technical in nature (written for developers).
- Some key terms:
  - **Endpoint** - URL (web location) that receives requests and sends responses.
  - **Parameters** - custom information that can be passed to the API.
  - **Response** - the data returned by the API.

Authentication

- Many APIs require a **key** or **tokens**.
- Most APIs are rate-limited
  - E.g. restrictions by user/key/IP address/time period.
- Make sure that you understand the terms of service/use.
- Even providers of public free APIs can impose some restrictions.

Example: Guardian API

```r
library("httr")
```

```r
# It is a good idea to not hard-code the API key in the script
# and, instead, load it dynamically from a file
api_key <- readLines("../temp/guardian_api_key.txt")
```

```r
# Endpoint
base_url <- "https://content.guardianapis.com/search"
```

```r
# Parameters
params <- list(
  "api-key" = api_key,
  "q" = "ireland",
  "page-size" = 1
)
```

```r
# Make the request and receive the response
response <- httr::GET(url = base_url, query = params)
```

```r
# Check the status of the response (200 means successful)
# Status codes 4xx are client errors; 5xx are server errors
response$status_code
```

```
[1] 200
```

```r
response
```

```
Response [https://content.guardianapis.com/search?api-key=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX&q=ireland&page-size=1]
  Date: 2026-02-05 10:55
  Status: 200
  Content-Type: application/json
  Size: 627 B
```

JSON

- **JSON** (**J**ava**S**cript **O**bject **N**otation) is a lightweight data-interchange format
- It is commonly used in web APIs (as well as elsewhere, e.g. Jupyter Notebooks).
- At its core, JSON objects are key-value pairs (often deeply nested).
- Keys have to be strings with double quotes.
- Values can be one of the following types:
  - String (e.g., “example”)
  - Number (e.g., 42, 3.141)
  - Array (e.g., [“a”, “b”, “c”])
  - Boolean (e.g., true, false)
  - null

Extra

JSON Syntax

JSON: Example

```r
library("jsonlite")
```

```r
json <- httr::content(response, as = "text", encoding = "UTF-8")
```

```r
json |>
  jsonlite::prettify()
```


```
{
    "response": {
        "status": "ok",
        "userTier": "developer",
        "total": 115673,
        "startIndex": 1,
        "pageSize": 1,
        "currentPage": 1,
        "pages": 115673,
        "orderBy": "relevance",
        "results": [
            {
                "id": "travel/2026/jan/05/i-ran-1400-miles-around-ireland",
                "type": "article",
                "sectionId": "travel",
                "sectionName": "Travel",
                "webPublicationDate": "2026-01-05T07:00:29Z",
                "webTitle": "I ran 1,400 miles around Ireland",
                "webUrl": "https://www.theguardian.com/travel/2026/jan/05/i-ran-1400-miles-around-ireland",
                "apiUrl": "https://content.guardianapis.com/travel/2026/jan/05/i-ran-1400-miles-around-ireland",
                "isHosted": false,
                "pillarId": "pillar/lifestyle",
                "pillarName": "Lifestyle"
            }
        ]
    }
}
```

JSON Representation

- As JSON is just a text format, its representation in code will vary by language.
  - E.g. in R JSON -> list, in Python -> dictionary

```r
json_parsed <- jsonlite::fromJSON(json)
```

```r
str(json_parsed)
```

```
List of 1
 $ response:List of 9
  ..$ status     : chr "ok"
  ..$ userTier   : chr "developer"
  ..$ total      : int 115673
  ..$ startIndex : int 1
  ..$ pageSize   : int 1
  ..$ currentPage: int 1
  ..$ pages      : int 115673
  ..$ orderBy    : chr "relevance"
  ..$ results    :'data.frame': 1 obs. of  11 variables:
  .. ..$ id                 : chr "travel/2026/jan/05/i-ran-1400-miles-around-ireland"
  .. ..$ type               : chr "article"
  .. ..$ sectionId          : chr "travel"
  .. ..$ sectionName        : chr "Travel"
  .. ..$ webPublicationDate: chr "2026-01-05T07:00:29Z"
  .. ..$ webTitle           : chr "I ran 1,400 miles around Ireland"
  .. ..$ webUrl             : chr "https://www.theguardian.com/travel/2026/jan/05/i-ran-1400-miles-around-
ireland"
  .. ..$ apiUrl             : chr "https://content.guardianapis.com/travel/2026/jan/05/i-ran-1400-miles-around-
```

JSON Representation

```
1 dim(json_parsed$response$results)
```

```
[1]  1 11
```

```
1 head(json_parsed$response$results)
```

```
                                                 id    type sectionId
1 travel/2026/jan/05/i-ran-1400-miles-around-ireland article    travel
  sectionName   webPublicationDate                        webTitle
1      Travel 2026-01-05T07:00:29Z I ran 1,400 miles around Ireland
                                                                                webUrl
1 https://www.theguardian.com/travel/2026/jan/05/i-ran-1400-miles-around-ireland
                                                                                   apiUrl
1 https://content.guardianapis.com/travel/2026/jan/05/i-ran-1400-miles-around-ireland
  isHosted         pillarId pillarName
1    FALSE pillar/lifestyle  Lifestyle
```

Next

- Tutorial: Text Preprocessing and APIs
- Next week: Dictionaries
- Assignment 1: Due 15:59 on Wednesday, 11th February (submission on Blackboard)