SM-2302 Special Lectures

Dr. Haziq Jamil

2024-09-18

Welcome

The three topics for this year's special lectures are:

1. Web scraping and modelling
2. GIS and spatial data
3. Quantitative text analysis

You will find the lecture notes and materials for each topic by clicking on the menu button. These lectures are meant to be more "hands-on" than the previous R1-4 lectures, so be prepared to get your hands dirty with some code!

You may also browse the GitHub repo for the special lecture notes. In particular, you will find `.qmd` files, which you can open in RStudio. Best to clone the repo to your local computers and open the `special-lectures.Rproj` file in RStudio. You will find the source code exactly as it is displayed in the html files. You can also turn on "visual" mode in RStudio to see a friendlier version of the `.qmd` files.

Part I

Lectures

1 Data Scraping and Linear Regression

For topic 1, we will cover linear regression. But before diving into that topic, we will talk about how to scrape data from the web.

- https://r4ds.hadley.nz/webscraping

Libraries:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(rvest)
```

```
Attaching package: 'rvest'

The following object is masked from 'package:readr':

    guess_encoding
```

```
library(polite)
```

1.1 Ethics

Data scraping is the process of extracting data from websites. This can be done manually, but it is often done using a program. In this section, we will use the `rvest` package to scrape data from a website.

When scraping car prices from sellers' websites in Brunei, it's important to consider legal and ethical aspects:

1. **Legal Considerations**: If the data is public, non-personal, and factual, scraping is generally acceptable. However, laws vary by location. If the data is behind a login or used for commercial purposes, consulting a lawyer is advisable.
2. **Terms of Service**: Many websites prohibit scraping in their terms of service. In some regions, like the US, these terms may not be binding unless you explicitly agree to them (e.g., by creating an account). In Europe, these terms are often enforceable even without explicit consent.
3. **Personally Identifiable Information**: Avoid scraping data that includes personal information (names, contact details, etc.) due to strict privacy laws like the GDPR in Europe. Ethical concerns arise even if the data is public.
4. **Copyright**: Data like car prices is generally not protected by copyright, as it's factual. However, if scraping includes original content (like descriptions or images), consider copyright laws and whether "fair use" applies.

1.2 HTML basics

HTML stands for "HyperText Markup Language" and looks like this:

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

HTML has a hierarchical structure formed by **elements** which consist of a start tag (e.g. `<tag>`), optional **attributes** (`id='first'`), an end tag[1] (like `</tag>`), and **contents** (everything in between the start and end tag).

Since `<` and `>` are used for start and end tags, you can't write them directly. Instead you have to use the HTML **escapes** `&gt;` (greater than) and `&lt;` (less than). And since those escapes use `&`, if you want a literal ampersand you have to escape it as `&amp;`. There are a wide range of possible HTML escapes but you don't need to worry about them too much because rvest automatically handles them for you.

1.2.1 Elements

All up, there are over 100 HTML elements. Some of the most important are:

- Every HTML page must be in an `<html>` element, and it must have two children: `<head>`, which contains document metadata like the page title, and `<body>`, which contains the content you see in the browser.
- Block tags like `<h1>` (heading 1), `<p>` (paragraph), and `<ol>` (ordered list) form the overall structure of the page.
- Inline tags like `<b>` (bold), `<i>` (italics), and `<a>` (links) formats text inside block tags.

If you encounter a tag that you've never seen before, you can find out what it does with a little googling. I recommend the MDN Web Docs which are produced by Mozilla, the company that makes the Firefox web browser.

1.2.2 Contents

Most elements can have content in between their start and end tags. This content can either be text or more elements. For example, the following HTML contains paragraph of text, with one word in bold.

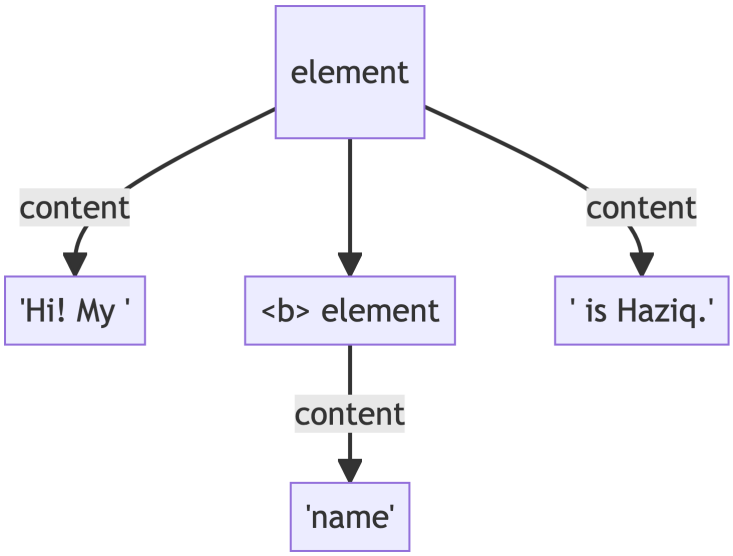
```
<p>
  Hi! My <b>name</b> is Haziq.
</p>
```

This renders as

The **children** of a node refers only to elements, so the `<p>` element above has one child, the `<b>` element. The `<b>` element has no children, but it does have contents (the text "name").

[1] A number of tags (including `<p>` and `<li>`) don't require end tags, but I think it's best to include them because it makes seeing the structure of the HTML a little easier.

Conceptually, this can be represented as follows:



Some elements, like `<img>` can't have children. These elements depend solely on attributes for their behavior.

1.2.3 Attributes

Tags can have named **attributes** which look like `name1='value1' name2='value2'`. Two of the most important attributes are `id` and `class`, which are used in conjunction with CSS (Cascading Style Sheets) to control the visual appearance of the page. These are often useful when scraping data off a page.

1.3 Reading HTML with rvest

You'll usually start the scraping process with `read_html()`. This returns an `xml_document`[2] object which you'll then manipulate using rvest functions:

```
html <- read_html("http://rvest.tidyverse.org/")
class(html)
```

```
[1] "xml_document" "xml_node"
```

[2]This class comes from the xml2 package. xml2 is a low-level package that rvest builds on top of.

For examples and experimentation, rvest also includes a function that lets you create an `xml_document` from literal HTML:

```
html <- minimal_html("
  <p>This is a paragraph<p>
  <ul>
    <li>This is a bulleted list</li>
  </ul>
")
html
```

```
{html_document}
<html>
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
[2] <body>\n<p>This is a paragraph</p>\n<p>\n  </p>\n<ul>\n<li>This is a bull ...
```

Regardless of how you get the HTML, you'll need some way to identify the elements that contain the data you care about. rvest provides two options: CSS selectors and XPath expressions. Here I'll focus on CSS selectors because they're simpler but still sufficiently powerful for most scraping tasks.

1.4 CSS selectors

CSS is short for cascading style sheets, and is a tool for defining the visual styling of HTML documents. CSS includes a miniature language for selecting elements on a page called **CSS selectors**. CSS selectors define patterns for locating HTML elements, and are useful for scraping because they provide a concise way of describing which elements you want to extract.

CSS selectors can be quite complex, but fortunately you only need the simplest for rvest, because you can also write R code for more complicated situations. The four most important selectors are:

- `p`: selects all `<p>` elements.
- `.title`: selects all elements with `class` "title".
- `p.special`: selects all `<p>` elements with `class` "special".
- `#title`: selects the element with the `id` attribute that equals "title". Id attributes must be unique within a document, so this will only ever select a single element.

If you want to learn more CSS selectors I recommend starting with the fun CSS dinner tutorial and then referring to the MDN web docs.

Lets try out the most important selectors with a simple example:

```
html <- minimal_html("
  <h1>This is a heading</h1>
  <p id='first'>This is a paragraph</p>
  <p class='important'>This is an important paragraph</p>
")
```

In rvest you can extract a single element with `html_element()` or all matching elements with `html_elements()`. Both functions take a document[3] and a css selector:

```
html |> html_element("h1")
```

```
{html_node}
<h1>
```

```
html |> html_elements("p")
```

```
{xml_nodeset (2)}
[1] <p id="first">This is a paragraph</p>
[2] <p class="important">This is an important paragraph</p>
```

```
html |> html_elements(".important")
```

```
{xml_nodeset (1)}
[1] <p class="important">This is an important paragraph</p>
```

```
html |> html_elements("#first")
```

```
{xml_nodeset (1)}
[1] <p id="first">This is a paragraph</p>
```

Selectors can also be combined in various ways using **combinators**. For example,The most important combinator is " ", the **descendant** combination, because `p a` selects all `<a>` elements that are a child of a `<p>` element.

If you don't know exactly what selector you need, I highly recommend using SelectorGadget, which lets you automatically generate the selector you need by supplying positive and negative examples in the browser.

[3] Or another element, more on that shortly.

1.5 Extracting data

Now that you've got the elements you care about, you'll need to get data out of them. You'll usually get the data from either the text contents or an attribute. But, sometimes (if you're lucky!), the data you need will be in an HTML table.

1.5.1 Text

Use `html_text2()` to extract the plain text contents of an HTML element:

```
html <- minimal_html("
  <ol>
    <li>apple &amp;amp; pear</li>
    <li>banana</li>
    <li>pineapple</li>
  </ol>
")
html |>
  html_elements("li") |>
  html_text2()
```

```
[1] "apple & pear" "banana"       "pineapple"
```

Note that the escaped ampersand is automatically converted to `&`; you'll only ever see HTML escapes in the source HTML, not in the data returned by rvest.

You might wonder why I used `html_text2()`, since it seems to give the same result as `html_text()`:

```
html |>
  html_elements("li") |>
  html_text()
```

```
[1] "apple & pear" "banana"       "pineapple"
```

The main difference is how the two functions handle white space. In HTML, white space is largely ignored, and it's the structure of the elements that defines how text is laid out. `html_text2()` does its best to follow the same rules, giving you something similar to what you'd see in the browser. Take this example which contains a bunch of white space that HTML ignores.

```
html <- minimal_html("<body>
  <p>
  This is
  a
  paragraph.</p><p>This is another paragraph.

  It has two sentences.</p>
")
```

`html_text2()` gives you what you expect: two paragraphs of text separated by a blank line.

```
html |>
  html_element("body") |>
  html_text2() |>
  cat()
```

```
This is a paragraph.

This is another paragraph. It has two sentences.
```

Whereas `html_text()` returns the garbled raw underlying text:

```
html |>
  html_element("body") |>
  html_text() |>
  cat()
```

```


  This is
  a
  paragraph.This is another paragraph.

  It has two sentences.
```

1.5.2 Attributes

Attributes are used to record the destination of links (the `href` attribute of `<a>` elements) and the source of images (the `src` attribute of the `<img>` element):

```
html <- minimal_html("
  <p><a href='https://en.wikipedia.org/wiki/Cat'>cats</a></p>
  <img src='https://cataas.com/cat' width='100' height='200'>
")
```

The value of an attribute can be retrieved with `html_attr()`:

```
html |>
  html_elements("a") |>
  html_attr("href")
```

```
[1] "https://en.wikipedia.org/wiki/Cat"
```

```
html |>
  html_elements("img") |>
  html_attr("src")
```

```
[1] "https://cataas.com/cat"
```

Note that `html_attr()` always returns a string, so you may need to post-process with `as.integer()`/`readr::parse_integer()` or similar.

```
html |>
  html_elements("img") |>
  html_attr("width")
```

```
[1] "100"
```

```
html |>
  html_elements("img") |>
  html_attr("width") |>
  as.integer()
```

```
[1] 100
```

1.5.3 Tables

HTML tables are composed four main elements: `<table>`, `<tr>` (table row), `<th>` (table heading), and `<td>` (table data). Here's a simple HTML table with two columns and three rows:

```
html <- minimal_html("
  <table>
    <tr>
      <th>x</th>
      <th>y</th>
    </tr>
    <tr>
      <td>1.5</td>
      <td>2.7</td>
    </tr>
    <tr>
      <td>4.9</td>
      <td>1.3</td>
    </tr>
    <tr>
      <td>7.2</td>
      <td>8.1</td>
    </tr>
  </table>
  ")
```

Because tables are a common way to store data, rvest includes the handy `html_table()` which converts a table into a data frame:

```
html |>
  html_node("table") |>
  html_table()
```

```
# A tibble: 3 x 2
      x     y
  <dbl> <dbl>
1   1.5   2.7
2   4.9   1.3
3   7.2   8.1
```

1.6 Element vs elements

When using rvest, your eventual goal is usually to build up a data frame, and you want each row to correspond some repeated unit on the HTML page. In this case, you should generally start by using `html_elements()` to select the elements that contain each observation then use `html_element()` to extract the variables from each observation. This guarantees that you'll get the same number of values for each variable because `html_element()` always returns the same number of outputs as inputs.

To illustrate this problem take a look at this simple example I constructed using a few entries from `dplyr::starwars`:

```
html <- minimal_html("
  <ul>
    <li><b>C-3PO</b> is a <i>droid</i> that weighs <span class='weight'>167 kg</span></li>
    <li><b>R2-D2</b> is a <i>droid</i> that weighs <span class='weight'>96 kg</span></li>
    <li><b>Yoda</b> weighs <span class='weight'>66 kg</span></li>
    <li><b>R4-P17</b> is a <i>droid</i></li>
  </ul>
  ")
```

If you try to extract name, species, and weight directly, you end up with one vector of length four and two vectors of length three, and no way to align them:

```
html |> html_elements("b") |> html_text2()
```

```
[1] "C-3PO"  "R2-D2"  "Yoda"   "R4-P17"
```

```
html |> html_elements("i") |> html_text2()
```

```
[1] "droid" "droid" "droid"
```

```
html |> html_elements(".weight") |> html_text2()
```

```
[1] "167 kg" "96 kg"  "66 kg"
```

Instead, use `html_elements()` to find a element that corresponds to each character, then use `html_element()` to extract each variable for all observations:

```
characters <- html |> html_elements("li")

characters |> html_element("b") |> html_text2()
```

```
[1] "C-3PO"  "R2-D2"  "Yoda"   "R4-P17"
```

```
characters |> html_element("i") |> html_text2()
```

```
[1] "droid" "droid" NA      "droid"
```

```
characters |> html_element(".weight") |> html_text2()
```

```
[1] "167 kg" "96 kg"  "66 kg"  NA
```

`html_element()` automatically fills in `NA` when no elements match, keeping all of the variables aligned and making it easy to create a data frame:

```
data.frame(
  name = characters |> html_element("b") |> html_text2(),
  species = characters |> html_element("i") |> html_text2(),
  weight = characters |> html_element(".weight") |> html_text2()
)
```

```
    name species weight
1  C-3PO   droid 167 kg
2  R2-D2   droid  96 kg
3   Yoda    <NA>  66 kg
4 R4-P17   droid   <NA>
```

1.7 Scraping house prices

(LIVE DEMO)

```r
# This is how you get read the HTML into R
url <- "https://www.bruhome.com/v3/buy.asp?p_=buy&id=&district=&propose=&property=&price=&lo
html <- read_html(url)

# Extract the house prices
prices <-
  html |>
  html_elements(".property-price") |>
  html_text2()

# Clean up
prices <-
  str_remove_all(prices, "[^0-9]") |>  # Remove non-numeric characters
  as.integer()

# Do same thing for number of beds, baths, location, and other remarks
beds <-
  html |>
  html_elements(".property-bed") |>
  html_text2() |>
  as.integer()

baths <-
  html |>
  html_elements(".property-bath") |>
  html_text2() |>
  as.integer()

location <-
  html |>
  html_elements(".property-address") |>
  html_text2()

remarks <-
  html |>
  html_elements("div p .mt-3") |>
  html_text2()

remarks <- tail(remarks, length(prices))

# Put it all in a data frame
hsp_df <- tibble(
```

```
  price = prices,
  beds = beds,
  baths = baths,
  location = location,
  remarks = remarks
)
```

Some pages require you to click a "load more" button to see all the data.

```
# Extract the links
properties <-
  html |>
  html_elements(".property-link") |>
  html_attr("href")

# Suppose I have a function that can extract the info I want from a single page
extract_info <- function(i) {
  link <- paste0("https://www.bruhome.com/v3/", properties[i])
  html <- read_html(link)
  out <-
    html |>
    html_elements("p") |>
    html_text2()
  out[1]
}

# Now what I could do is the following:
# res <- c()
# for (i in seq_along(properties)) {
#   res[i] <- extract_info(i)
# }

# A better way:
res <- map(
  .x = seq_along(properties),
  .f = extract_info,
  .progress = TRUE
)
```

1.8 Cleaning using LLM

ADVANCED TOPIC!!!

```
# remotes::install_github("AlbertRapp/tidychatmodels")
library(tidychatmodels)

chat <-
  create_chat("ollama") |>
  add_model("llama3.1") |>
  add_message('What is love? IN 10 WORDS.') |>
  perform_chat()

extract_chat(chat)

# Try to prime it to clean the data set
clean_desc <- function(i) {
  create_chat("ollama") |>
  add_model("llama3.1") |>
  add_message(glue::glue("
    The following is a description of house for sale in Brunei obtained from the web site of

    1. Built up area (usually in square feet) [NUMERIC]
    2. Type of house (whether it is detached, semi-detached, terrace, apartment, or other) [C

  Please return semicolon separated values like this:
  2500; detached
  3000; semi-detached
  2000; terrace
  etc.
  NUMBERS SHOULD NOT CONTAIN comma (,) for thousands separator

  Please only return these two values and nothing else. Do not return any other information.

  NOTE: Some of these listings may be related to LAND or COMMERCIAL properties. In this case

  IF YOU DO NOT SEE ANY DESCRIPTION it may mean that the description is missing. In this case

  IF YOU SEE MULTIPLE DESCRIPTIONS, please return the first one only.

  ----------
```

```
  {res[[i]]}
  ")) |>
  perform_chat() |>
  extract_chat(silent = TRUE) |>
  filter(role == "assistant") |>
  pull(message)
}

# Now map it over the descriptions!
cleaned_descriptions <-
  map(
    .x = seq_along(res),
    .f = clean_desc,
    .progress = TRUE
  )

# Now add to the hsp_df
hsp_df$desc <- unlist(cleaned_descriptions)
hsp_df <-
  hsp_df |>
  mutate(
    desc = unlist(res),
    cleaned_desc = unlist(cleaned_descriptions)
  ) |>
  separate(cleaned_desc, into = c("sqft", "type"), sep = ";") |>
  mutate(
    sqft = as.integer(sqft),
    type = case_when(
      grepl("detached", type, ignore.case = TRUE) ~ "detached",
      grepl("semi-detached", type, ignore.case = TRUE) ~ "semi-detached",
      grepl("terrace", type, ignore.case = TRUE) ~ "terrace",
      grepl("apartment", type, ignore.case = TRUE) ~ "apartment",
      grepl("land", type, ignore.case = TRUE) ~ "land",
      grepl("commercial", type, ignore.case = TRUE) ~ "commercial",
      TRUE ~ NA
    )
  )
# save(hsp_df, file = "data/hsp_df.RData")
```

1.9 Linear regression

In statistical modelling, the aim is to describe the relationship between one or more **predictor variables** (usually denoted x) and a **response variable** (usually denoted y). Mathematically, we can say

$$y = f(x) + \epsilon.$$

Here f is some *regression* function that we want to estimate, and ϵ is an error term that captures the difference between the true relationship and our estimate.

The simplest type of modelling is called **linear regression**, where we assume that the relationship between x and y is linear. That is,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon.$$

When we ask software to estimate the β coefficients, it will find the values that optimise a certain criterion (typically, one that yields the smallest error values). In R, you need to supply two things:

1. A formula that describes the relationship between the variables.
2. The data frame that contains the variables.

1.9.1 Model fit

Here's an example:

```
load("data/hsp_df.RData")  # I saved this data set earlier and load it back

# First clean the data a bit
htypes <- c("detached", "semi-detached", "terrace", "apartment")
hsp_mod_df <-
  hsp_df |>
  filter(type %in% htypes) |>
  mutate(
    type = factor(type, levels = htypes),
    priceK = price / 1000,  # Price in thousands
    sqftK = sqft / 1000  # Square feet in thousands
  ) |>
  select(priceK, beds, baths, sqftK, type) |>
  drop_na()

fit <- lm(priceK ~ beds + baths + sqftK + type, data = hsp_mod_df)
summary(fit)
```

```
Call:
lm(formula = priceK ~ beds + baths + sqftK + type, data = hsp_mod_df)

Residuals:
     Min       1Q   Median       3Q      Max
-122.637  -38.483   -2.603   32.274  165.839

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -178.288     47.360  -3.765 0.000387 ***
beds            75.147     16.837   4.463 3.70e-05 ***
baths           44.832     10.470   4.282 6.92e-05 ***
sqftK           -1.654      1.894  -0.873 0.386067
typeterrace    -41.525     20.878  -1.989 0.051350 .
typeapartment   62.151     34.485   1.802 0.076615 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 56.59 on 59 degrees of freedom
Multiple R-squared:  0.7965,    Adjusted R-squared:  0.7792
F-statistic: 46.18 on 5 and 59 DF,  p-value: < 2.2e-16
```

1.9.2 Interpretation

Here's a breakdown of the key components to aid your understanding:

1. **Residuals**: These represent the differences between the observed and predicted values of `priceK`. The five-number summary (Min, 1Q, Median, 3Q, Max) helps you understand the distribution of residuals. Ideally, the residuals should be symmetrically distributed around zero, indicating a good model fit.
2. **Coefficients**: This section shows the estimated effect (Estimate) of each predictor on `priceK`:
    - **(Intercept)**: The expected `priceK` when all predictors are zero. Here, it's -178.288, but this value often doesn't have a real-world interpretation if the predictor values can't actually be zero.
    - **beds**: Each additional bedroom increases the expected `priceK` by about 75.15 (thousand).
    - **baths**: Each additional bathroom increases the expected `priceK` by about 44.83 (thousand).

- **sqftK**: The effect of the square footage in thousands on `priceK`. Here, it's not statistically significant (p-value = 0.386), meaning it doesn't contribute much to predicting `priceK`.
- **type**: This is a categorical variable with three levels. The coefficients for `typeterrace` and `typeapartment` are relative to the reference category (likely another property type not shown here, such as "detached house"). For example, `typeterrace` lowers the expected `priceK` by 41.53 (thousand) compared to the reference category.

3. **Significance Codes**: Indicators of statistical significance for each predictor:
   - `***` highly significant ($p < 0.001$)
   - `**` significant ($p < 0.01$)
   - `*` moderately significant ($p < 0.05$)
   - `.` marginally significant ($p < 0.1$)
   - None of these symbols indicate non-significance.

4. **Residual Standard Error**: This is the standard deviation of the residuals. A smaller value suggests a better fit, as it indicates that the observed values are closer to the fitted values.

5. **R-squared and Adjusted R-squared**:
   - **R-squared** (0.7965) indicates that about 79.65% of the variability in `priceK` is explained by the model.
   - **Adjusted R-squared** (0.7792) is a modified version of R-squared that accounts for the number of predictors, providing a more accurate measure for models with multiple variables.

6. **F-statistic**: This tests whether at least one predictor variable is significantly related to the dependent variable. A p-value of $< 2.2e\text{-}16$ indicates the model is highly significant.

Key Takeaway: The model shows that `beds` and `baths` significantly predict `priceK`, while `sqftK` does not have a significant effect. The `type` variable shows some variation, with `typeterrace` having a marginally significant negative effect on `priceK`. Overall, the model explains a large proportion of the variation in house prices.

1.9.3 Predictions

One of the main uses of a linear regression model is to make predictions. That is, given a set of predictor values (typically unseen data), we can estimate the response variable. In the context of the house price data, this means we can estimate the price of a house given its number of bedrooms, bathrooms, square footage, and type.

First, let's set up the data for a new house:

```r
new_house <- tibble(
  beds = 3,
  baths = 2,
  sqftK = 2.5,
  type = factor("detached", levels = htypes)
)
```

Then, to predict the price, we run the following command:

```r
predict(fit, newdata = new_house, interval = "prediction", level = 0.95)
```

```
       fit      lwr      upr
1 132.6803 14.74783 250.6128
```

This also gives the 95% prediction interval, which is a range of values within which we expect the true price to fall with 95% confidence. What we can see is that the model predicts the price of the new house to be around 133,000 Brunei dollars, with a 95% prediction interval of approximately [15,000, 251,000] Brunei dollars.

You might be wondering why the prediction interval is so wide. This is because the model is uncertain about the price of a new house, given the limited information we have. Generally, the more data you have, the narrower the prediction interval will be.

> **Note**
>
> You can get model predictions for the original data set by using the `predict()` without `newdata` argument. Alternatively, `fitted()` works too.

1.10 More advanced models

Linear regression is a simple and powerful tool, but it has its limitations. If you were more interested in predictions, then you might want to consider more advanced machine learning (ML) models. Here are a couple of suggestions:

1. **Random Forest**: This is an ensemble learning method that builds multiple decision trees and merges them together to get a more accurate and stable prediction.
2. **Gradient Boosting Machines (GBM)**: GBM is another ensemble learning method that builds multiple decision trees sequentially, with each tree correcting the errors of the previous one.
3. **Neural Networks**: These are a set of algorithms that are designed to recognize patterns, with the ability to model complex relationships between inputs and outputs.

1.10.1 Random forests

Random forests are popular because they are easy to use and generally provide good results. Here's how you can fit a random forest model to the house price data:

```
library(randomForest)
```

```
randomForest 4.7-1.1
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```

Attaching package: 'randomForest'
```

```
The following object is masked from 'package:dplyr':
```

```

    combine
```

```
The following object is masked from 'package:ggplot2':
```

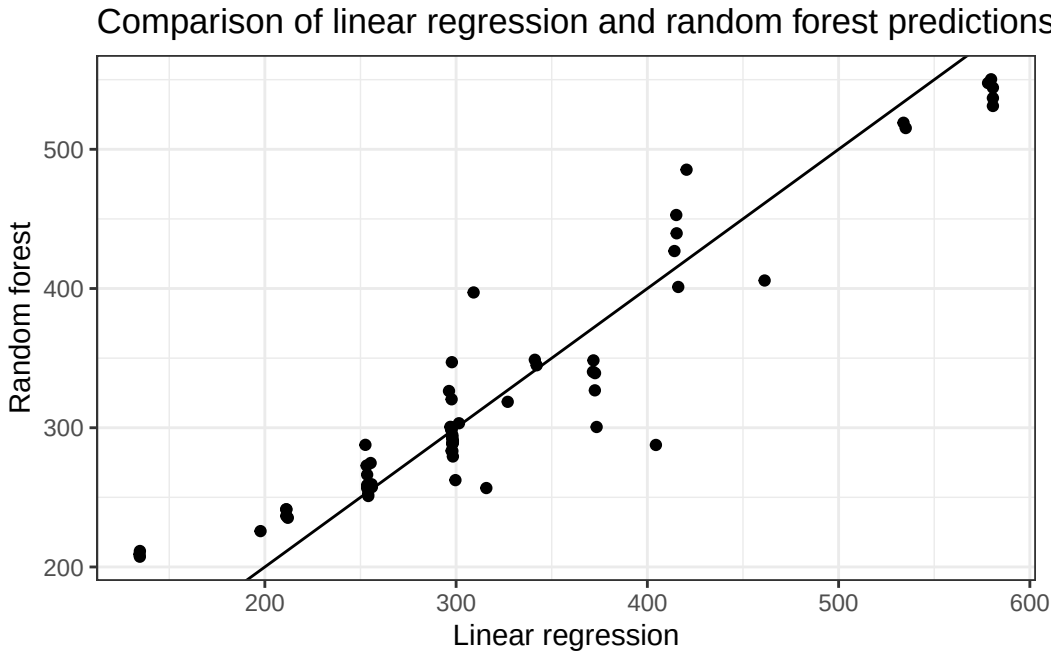
```

    margin
```

```
fit_rf <- randomForest(priceK ~ beds + baths + sqftK + type, data = hsp_mod_df)
```

With random forests, you don't really get "beta" coefficients. So there's no point running `summary()`. Instead, it's mainly used as a black box to obtain predicted values.

Let's compare the predictions from the random forest model to the linear regression model:

```
tibble(
  lm = predict(fit),
  rf = predict(fit_rf)
) |>
  ggplot(aes(lm, rf)) +
  geom_point() +
  geom_abline() +
  labs(
    x = "Linear regression",
    y = "Random forest",
    title = "Comparison of linear regression and random forest predictions"
  ) +
  theme_bw()
```



To see which model gives smaller errors, we can run the following code:

```
resid_lm <- hsp_mod_df$priceK - predict(fit)
resid_rf <- hsp_mod_df$priceK - predict(fit_rf)

# Residual sum of squares
sum(resid_lm ^ 2)
```

```
[1] 188934.9
```

```
sum(resid_rf ^ 2)
```

```
[1] 210449.4
```

In this case, the linear regression model has a smaller residual sum of squares, indicating that it fits the data better.

Out of curiosity, let's see the predictions for the new house using the random forest model:

```
predict(fit_rf, newdata = new_house, )
```

```
       1
235.3756
```

Which seems very different to the `lm()` predictions.

2 Geographical Information System (GIS) data

Libraries needed:

```
library(tidyverse)
# remotes::install_github("propertypricebn/bruneimap")
library(bruneimap)
library(ggrepel)
library(kernlab)
library(osrm)
library(osmdata)
```

More info:

- https://github.com/propertypricebn/bruneimap

> **Note**
>
> The `{bruneimap}` package contains the following data sets:
>
> 1. `dis_sf`: Brunei districts geometries.
> 2. `mukim_sf`: Brunei mukim geometries.
> 3. `kpg_sf`: Brunei kampong geometries.
> 4. `brn_df`: Brunei outline geometries.
> 5. `bn_census2021`: Brunei 2021 census data.

2.1 Introduction

> **What we'll learn**
>
> - Types of GIS data and how these are handled in R.
> - Difference between spatial and non-spatial data analysis.
> - Importance of geocoding your data for spatial analysis.

Roughly speaking, there are 4 types of GIS data.

1. **Points**
    - Having (X, Y) coordinates (latitude, longitude, or projected coordinates, and are "zero-dimensional".
    - E.g. shopping malls, hospitals, outbreaks, etc.
2. **Lines**
    - A collection of points that form a path or a boundary. Has length.
    - E.g. roads, rivers, pipelines, etc.
3. **Polygons**
    - A closed area made up of line segments or curves.
    - E.g. countries, districts, buildings, etc.
4. **Raster**
    - Pixelated (or gridded) data where each pixel is associated with a geographical area and some measurement.
    - E.g. satellite images, elevation data, etc.

The first three are usually referred to as *vector data*. GIS data can be stored in various formats such as `.shp` or `.geojson`. The handling of GIS data (at least vector type data) is facilitated by the `{sf}` package (Pebesma and Bivand 2023) which uses the *simple features* standard.

> **Note**
>
> *Simple features* refers to a formal standard (ISO 19125-1:2004) that describes how objects in the real world can be represented in computers, with emphasis on the spatial geometry of these objects.

It's helpful to think about the shape of this spatial data set. As an example, here's a random slice of 10 kampong-level population data for Brunei:

```
left_join(
  kpg_sf,
  bn_census2021,
  join_by(id, kampong, mukim, district)
) |>
  select(
    kampong, district, population, geometry
  ) |>
  slice_sample(n = 10)
```

```
Simple feature collection with 10 features and 3 fields
```

```
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 114.5686 ymin: 4.745099 xmax: 115.0427 ymax: 4.996198
Geodetic CRS:  WGS 84
# A tibble: 10 x 4
   kampong                          district population                  geometry
   <chr>                            <chr>         <dbl>              <POLYGON [°]>
 1 Pulau Sibungor                   Brunei ~         NA ((114.9557 4.866652, 114~
 2 Kg. Batong                       Brunei ~       1319 ((114.8262 4.842174, 114~
 3 Kg. Tasek Meradun                Brunei ~        312 ((114.8916 4.872824, 114~
 4 Kg. Maraburong                   Tutong          457 ((114.777 4.831059, 114.~
 5 Kg. Panchor Lumapas              Brunei ~        664 ((114.9198 4.857752, 114~
 6 Kg. Panapar-Danau                Tutong          274 ((114.5755 4.748835, 114~
 7 Pulau ...                        Brunei ~         NA ((115.0419 4.995619, 115~
 8 Perumahan Negara Rimba Kawasan~ Brunei ~       1594 ((114.9026 4.966143, 114~
 9 Pusat Bandar                     Brunei ~        314 ((114.9417 4.894914, 114~
10 Kg. Buang Sakar                  Brunei ~         28 ((114.9462 4.865839, 114~
```

Spatial data analysis must have these two components:

1. The study variables (in the above example, this is population data).
2. GIS data regarding that study variable.

If we only have 1 without 2, then it really is just a regular data analysis (stating the obvious). Adding the GIS data is a process called "geocoding" the data points.

> **Note**
>
> In R, geocoding using `{tidyverse}` can be achieved using the `dplyr::left_join()` or similar `xxx_join()` family of functions.

2.2 `(MULTI)POINT` data

> **What we'll learn**
>
> - Loading data sets in R using `readr::read_csv()`.
> - Identifying data types and their implications.

Use the data from Jaafar and Sukri (2023) on the physicochemical characteristics and texture classification of soil in Bornean tropical heath forests affected by exotic Acacia mangium. There are three datasets provided.

1. GIS data (WGS84 coordinates) of all study plots.
2. Soil physicochemical property data. This contains details of soil physical, chemical, nutrient concentration of the three habits studied.
3. Soil texture classification. Provides details on the classification of the soil texture in the habitats studied.

We will first load the data sets in R.

```
# Load the data sets
soil_gps <- read_csv(
  "data/8389823/GPS - Revised.csv",
  # IMPORTANT!!! The csv file has latin1 encoding as opposed to UTF-8
  locale = readr::locale(encoding = "latin1")
)

soil_physico <- read_csv("data/8389823/Soil physicochemical properties.csv")
soil_texture <- read_csv("data/8389823/Soil texture classification.csv")
```

2.2.1 Clean up the point data

> What we'll learn
>
> - Highlighting the need for cleaning and preprocessing data.
> - Using `glimpse()` to peek at the data.
> - Using `mutate()` to change stuff in the data set.
> - Using `str()` to look at the structure of an R object.

Let's take a look at the point data set.

```
glimpse(soil_gps)
```

```
Rows: 18
Columns: 5
$ Forest_type  <chr> "Kerangas", "Kerangas", "Kerangas", "Kerangas", "Kerangas~
$ Habitat_type <chr> "Intact", "Intact", "Intact", "Intact", "Intact", "Intact~
$ Plot_name    <chr> "KU1", "KU2", "KU3", "KU4", "KU5", "KU6", "KI1", "KI2", "~
$ Latitude     <chr> "4° 35' 53.40\"N", "4° 35' 38.37\"N", "4° 35' 53.89\"N", ~
$ Longitude    <chr> "114° 30' 39.09\"E", "114° 31' 05.89\"E", "114° 30' 38.90~
```

The first three columns are essentially the identifiers of the plots (forest type, habitat type, and the unique identification code for the study plot). However, the latitude and longitude

needs a bit of cleaning up, because it's currently in character format. This needs to be in a formal Degree Minute Second `DMS` class that R can understand. For this we will use the `sp::char2dms()` function.

As an example let's take a look at the first latitude.

```
x <- soil_gps$Latitude[1]
x
```

```
[1] "4° 35' 53.40\"N"
```

```
# convert it using sp::char2dms() function
x <- sp::char2dms(x, chd = "°")
x
```

```
[1] 4d35'53.4"N
```

```
str(x)
```

```
Formal class 'DMS' [package "sp"] with 5 slots
  ..@ WS : logi FALSE
  ..@ deg: int 4
  ..@ min: int 35
  ..@ sec: num 53.4
  ..@ NS : logi TRUE
```

This is a special class that R understands as being a latitude from Earth. To convert it to decimal, we just do `as.numeric()`:

```
as.numeric(x)
```

```
[1] 4.598167
```

Now let's do this for all the values in the `soil_gps` data. We will use the `dplyr::mutate()` function in a pipeline.

```
soil_gps <-
  soil_gps |>
  mutate(
    Latitude = as.numeric(sp::char2dms(Latitude, chd = "°")),
    Longitude = as.numeric(sp::char2dms(Longitude, chd = "°"))
  )
soil_gps
```

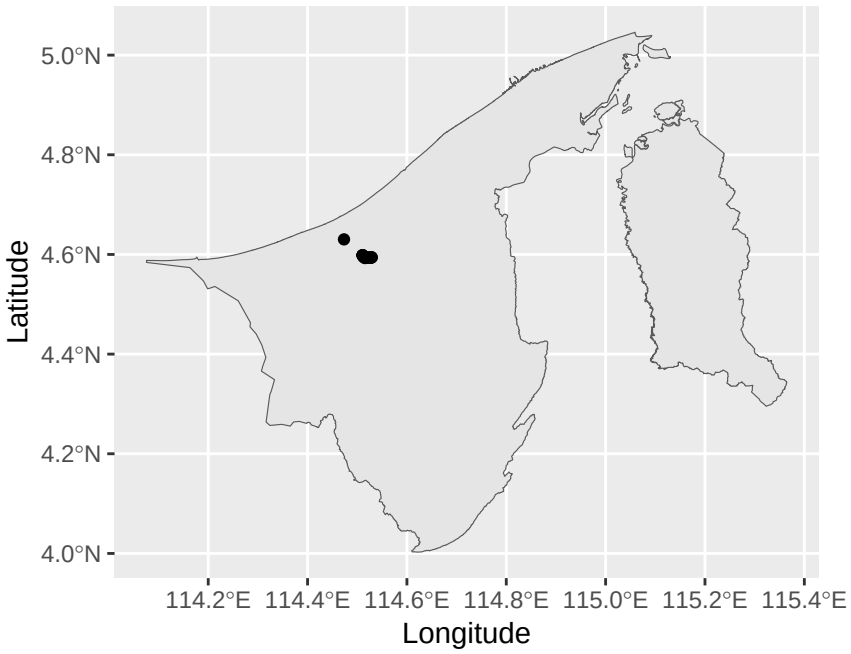
```
# A tibble: 18 x 5
   Forest_type Habitat_type Plot_name Latitude Longitude
   <chr>       <chr>        <chr>        <dbl>     <dbl>
 1 Kerangas    Intact       KU1           4.60      115.
 2 Kerangas    Intact       KU2           4.59      115.
 3 Kerangas    Intact       KU3           4.60      115.
 4 Kerangas    Intact       KU4           4.63      114.
 5 Kerangas    Intact       KU5           4.60      115.
 6 Kerangas    Intact       KU6           4.60      115.
 7 Kerangas    Invaded      KI1           4.59      115.
 8 Kerangas    Invaded      KI2           4.59      115.
 9 Kerangas    Invaded      KI3           4.59      115.
10 Kerangas    Invaded      KI4           4.59      115.
11 Kerangas    Invaded      KI5           4.59      115.
12 Kerangas    Invaded      KI6           4.59      115.
13 Kerangas    Plantation   AP1           4.59      115.
14 Kerangas    Plantation   AP2           4.59      115.
15 Kerangas    Plantation   AP3           4.59      115.
16 Kerangas    Plantation   AP4           4.59      115.
17 Kerangas    Plantation   AP5           4.59      115.
18 Kerangas    Plantation   AP6           4.59      115.
```

2.2.2 Preliminary plot of the data

> What we'll learn
>
> - Structure of a `ggplot()` (grammar of graphics).
> - Using `geom_sf()` to plot the GIS data, and adding points using `geom_point()`.

Using the data contained in the `{bruneimap}` package, we can plot the study areas on a map of Brunei. Use either the `brn_sf`, `dis_sf`, `mkm_sf` or `kpg_sf` data sets.

```
ggplot(brn_sf) +
  geom_sf() +
  geom_point(data = soil_gps, aes(Longitude, Latitude))
```



We can zoom in a bit… but we have to find out manually the correct bounding box. To do this, we can either:

1. Manually find the minimum and maximum values of the latitude and longitude.
2. Convert the `soil_gps` data set to an `sf` object and use the `st_bbox()` function.

```
# Manual way
c(
  xmin = min(soil_gps$Longitude), xmax = max(soil_gps$Longitude),
  ymin = min(soil_gps$Latitude), ymax = max(soil_gps$Latitude)
)
```

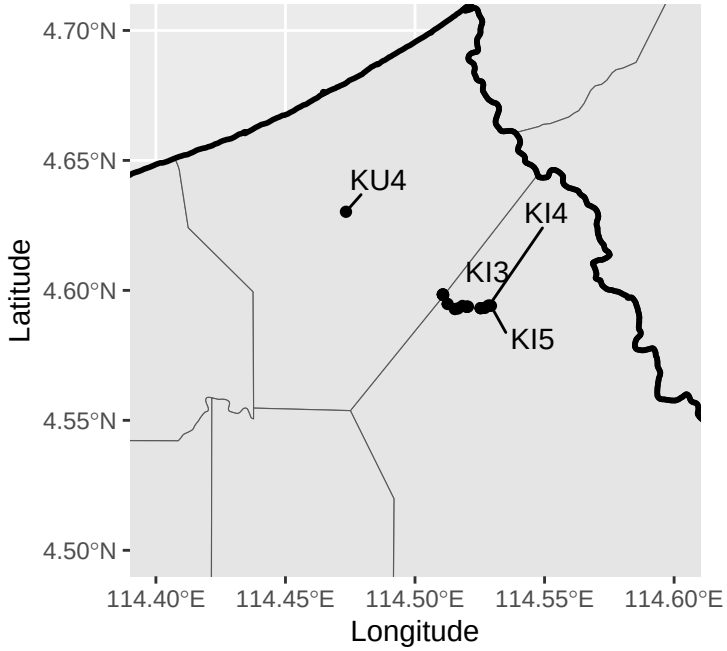
```
     xmin      xmax      ymin      ymax
114.473356 114.529297   4.592817   4.630242
```

```
# Using the sf object
soil_sf <- st_as_sf(soil_gps, coords = c("Longitude", "Latitude"), crs = 4326)
st_bbox(soil_sf)
```

```
     xmin      ymin      xmax      ymax
114.473356   4.592817 114.529297   4.630242
```

Now that we've found the bound box, we can plot better:

```
ggplot(mkm_sf) +
  geom_sf() +
  geom_sf(data = dis_sf, fill = NA, col = "black", linewidth = 1) +
  geom_point(data = soil_gps, aes(Longitude, Latitude)) +
  geom_text_repel(
    data = soil_gps,
    aes(Longitude, Latitude, label = Plot_name),
    box.padding = 0.5,
    max.overlaps = 30
  ) +
  coord_sf(
    xlim = c(114.4, 114.6),
    ylim = c(4.5, 4.7)
  )
```



2.2.3 Merge with the study data

> What we'll learn
>
> - Using `left_join()` to merge two data sets together.
> - Using `geom_jitter()` to plot the study variables that are overlapping.

Let's take a look at the data set.

```
glimpse(soil_physico)
```

```
Rows: 144
Columns: 16
$ Habitat_type              <chr> "Intact", "Intact", "Intact", "Intact", "Int~
$ Plot_name                 <chr> "KU1", "KU1", "KU1", "KU1", "KU1", "KU1", "K~
$ Subplot_name              <chr> "A", "A", "B", "B", "C", "C", "D", "D", "A",~
$ Soil_depth                <chr> "0-15", "30-50", "0-15", "30-50", "0-15", "3~
$ Nitrogen                  <dbl> 0.617, 0.188, 0.663, 0.200, 0.465, 0.255, 0.~
$ Phosphorus                <dbl> 0.248, 0.129, 0.259, 0.295, 0.172, 0.145, 0.~
$ Magnesium                 <dbl> 0.000, 0.045, 0.054, 0.035, 0.079, 0.043, 0.~
$ Calcium                   <dbl> 0.167, 0.187, 0.148, 0.113, 0.253, 0.229, 0.~
$ Potassium                 <dbl> 0.059, 0.037, 0.054, 0.022, 0.098, 0.033, 0.~
$ Exchangable_magnesium     <dbl> 0.009, 0.004, 0.007, 0.005, 0.029, 0.014, 0.~
$ Exchangable_calcium       <dbl> 0.010, 0.009, 0.008, 0.009, 0.109, 0.041, 0.~
$ Exchangable_potassium     <dbl> 0.101, 0.085, 0.092, 0.087, 0.101, 0.090, 0.~
$ Available_phosphorus      <dbl> 0.012, 0.012, 0.013, 0.012, 0.013, 0.014, 0.~
$ pH                        <dbl> 2.3, 2.7, 2.0, 2.0, 2.6, 2.5, 2.3, 2.1, 1.0,~
$ Gravimetric_water_content <dbl> 5.911, 3.560, 10.860, 5.082, 6.963, 4.549, 5~
$ Organic_matter            <dbl> 4.559, 1.399, 4.523, 2.309, 3.131, 2.209, 3.~
```

```
glimpse(soil_texture)
```

```
Rows: 144
Columns: 8
$ Habitat_type          <chr> "Intact", "Intact", "Intact", "Intact", "Intact~
$ Plot_name             <chr> "KU1", "KU1", "KU1", "KU1", "KU2", "KU2", "KU2"~
$ Subplot_name          <chr> "A", "B", "C", "D", "A", "B", "C", "D", "A", "B~
$ Soil_depth            <chr> "0-15", "0-15", "0-15", "0-15", "0-15", "0-15",~
$ Clay                  <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 2.5, 2.5, 2.5, 0.0, 2.~
$ Silt                  <dbl> 2.5, 0.0, 0.0, 2.5, 0.0, 0.0, 2.5, 2.5, 7.5, 7.~
$ Sand                  <dbl> 97.5, 100.0, 100.0, 97.5, 100.0, 97.5, 95.0, 95~
$ Texture_classification <chr> "Sand", "Sand", "Sand", "Sand", "Sand", "Sand",~
```

The `soil_physico` and `soil_texture` data sets contain the same columns, so we might as well merge them together. We will use the `dplyr::left_join()` function.

```
# Actually I just want to merge these two together
soil_df <- left_join(
  soil_physico,
  soil_texture,
  by = join_by(Habitat_type, Plot_name, Subplot_name, Soil_depth)
)
soil_df
```

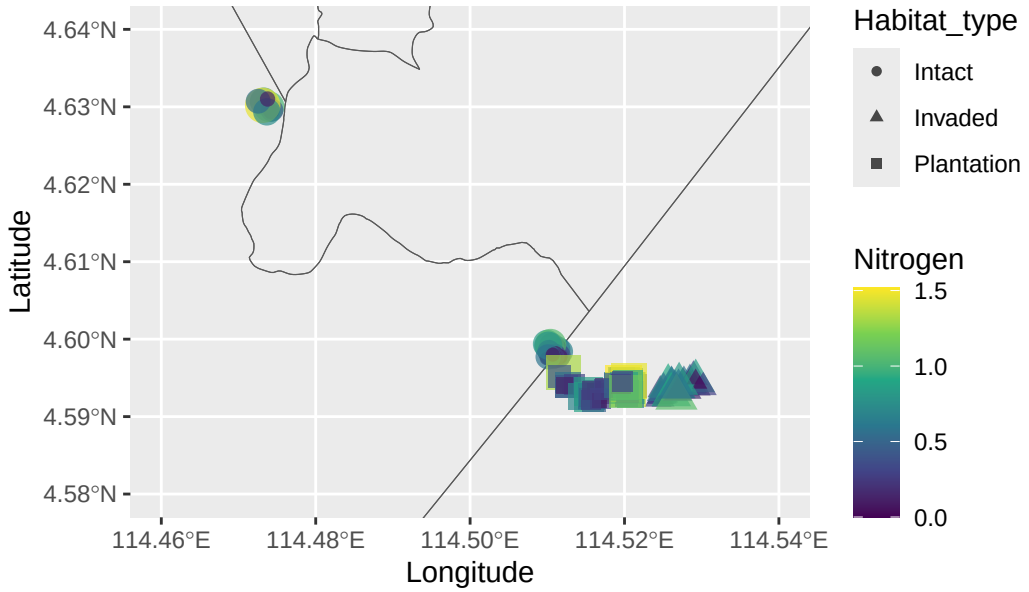
```
# A tibble: 144 x 20
   Habitat_type Plot_name Subplot_name Soil_depth Nitrogen Phosphorus Magnesium
   <chr>        <chr>     <chr>        <chr>         <dbl>      <dbl>     <dbl>
 1 Intact       KU1       A            0-15          0.617      0.248     0
 2 Intact       KU1       A            30-50         0.188      0.129     0.045
 3 Intact       KU1       B            0-15          0.663      0.259     0.054
 4 Intact       KU1       B            30-50         0.2        0.295     0.035
 5 Intact       KU1       C            0-15          0.465      0.172     0.079
 6 Intact       KU1       C            30-50         0.255      0.145     0.043
 7 Intact       KU1       D            0-15          0.285      0.225     0.052
 8 Intact       KU1       D            30-50         0.057      0.207     0.031
 9 Intact       KU2       A            0-15          0.37       0.135     0.038
10 Intact       KU2       A            30-50         0.114      0.168     0.021
# i 134 more rows
# i 13 more variables: Calcium <dbl>, Potassium <dbl>,
#   Exchangable_magnesium <dbl>, Exchangable_calcium <dbl>,
#   Exchangable_potassium <dbl>, Available_phosphorus <dbl>, pH <dbl>,
#   Gravimetric_water_content <dbl>, Organic_matter <dbl>, Clay <dbl>,
#   Silt <dbl>, Sand <dbl>, Texture_classification <chr>
```

Once we've done that, the `soil_df` data set (the study variables) is actually missing the spatial data. We need to geocode it with the `soil_gps` data set. Again, `dplyr::left_join()` to the rescue!

```
soil_df <- left_join(
  soil_df,
  soil_gps,
  by = join_by(Habitat_type, Plot_name)
)
```

Now we're in a position to plot the study variables on the map. Note that there are only 18 plots in the `soil_gps` data set, and each plot has repeated measurements. That means when we plot it, it will overlap and look like a single point. So a good thing to do is to jitter the point so it's easier to see.

```
ggplot(kpg_sf) +
  geom_sf(fill = NA) +
  geom_jitter(
    data = soil_df,
    aes(Longitude, Latitude, col = Nitrogen, size = Nitrogen,
        shape = Habitat_type),
    width = 0.001, height = 0.001, alpha = 0.7
  ) +
  coord_sf(
    xlim = c(114.46, 114.54),
    ylim = c(4.58, 4.64)
  ) +
  scale_color_viridis_c() +
  guides(size = "none")
```



2.3 Line data ((MULTI)LINESTRING)

> What we'll learn
>
> - How to load spatial data sets using `sf::read_sf()` and editing the CRS using `sf::st_transform()`.
> - How to filter data using `dplyr::filter()`.
> - How to plot line data using `ggplot2::geom_sf()`.

For this example, we'll play with the road network shape file obtained from OpenStreetMaps. The data is in geojson format, so let's import that into R.

```
brd <-
  read_sf("data/hotosm_brn_roads_lines_geojson/hotosm_brn_roads_lines_geojson.geojson") |>
  st_transform(4326)  # SET THE CRS!!! (WGS84)
glimpse(brd)
```

```
Rows: 25,570
Columns: 15
$ name       <chr> "Simpang 393", "Simpang 405", NA, NA, NA, NA, "Lebuhraya Tu~
$ `name:en`  <chr> NA, NA, NA, NA, NA, NA, "Tutong-Telisai Highway", NA, NA, N~
$ highway    <chr> "residential", "residential", "service", "residential", "tr~
$ surface    <chr> NA, NA, NA, NA, NA, "asphalt", "asphalt", NA, NA, NA, "asph~
$ smoothness <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ width      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ lanes      <chr> NA, NA, NA, NA, NA, "1", "2", NA, NA, NA, "2", NA, NA, NA, ~
$ oneway     <chr> NA, NA, NA, NA, NA, "yes", "yes", NA, NA, NA, "no", "yes", ~
$ bridge     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ layer      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ source     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ `name:ms`  <chr> NA, NA, NA, NA, NA, NA, "Lebuhraya Tutong-Telisai", NA, NA,~
$ osm_id     <int> 386886618, 481030903, 512405939, 664532755, 442044892, 6651~
$ osm_type   <chr> "ways_line", "ways_line", "ways_line", "ways_line", "ways_l~
$ geometry   <LINESTRING [°]> LINESTRING (114.6236 4.7910..., LINESTRING (114.~
```

There are 25,570 features in this data set, which may be a bit too much. Let's try to focus on the major roads only. This information seems to be contained in the `highway` column. What's in it?

```
table(brd$highway)
```

bridleway	construction	cycleway	footway	living_street
1	28	73	898	10
motorway	motorway_link	path	pedestrian	primary
116	152	140	60	865
primary_link	residential	road	secondary	secondary_link
332	9023	1	446	79
service	steps	tertiary	tertiary_link	track
9876	53	586	59	442
trunk	trunk_link	unclassified		
460	310	1560		

According to this wiki, In OpenStreetMap, the major roads of a road network are sorted on an importance scale, from motorway to quaternary road.

Drawing	Description	Wiki page
	Motorway, the most important roads in a road network. Equivalent to freeway, Autobahn (Germany), etc.	highway=motorway
	The link roads (sliproads / ramps) leading to and from a motorway	highway=motorway_link
	Motorway under construction / Motorway link under construction	highway=construction + construction=motorway / highway=construction + construction=motorway_link
	Trunks, the most important roads in a road network that aren't motorways	highway=trunk
	The link roads (sliproads / ramps) leading to and from a trunk highway	highway=trunk_link
	Trunk under construction / Trunk link under construction	highway=construction + construction=trunk / highway=construction + construction=trunk_link
	Primary road	highway=primary
	Connecting slip roads/ramps of primary highways	highway=primary_link

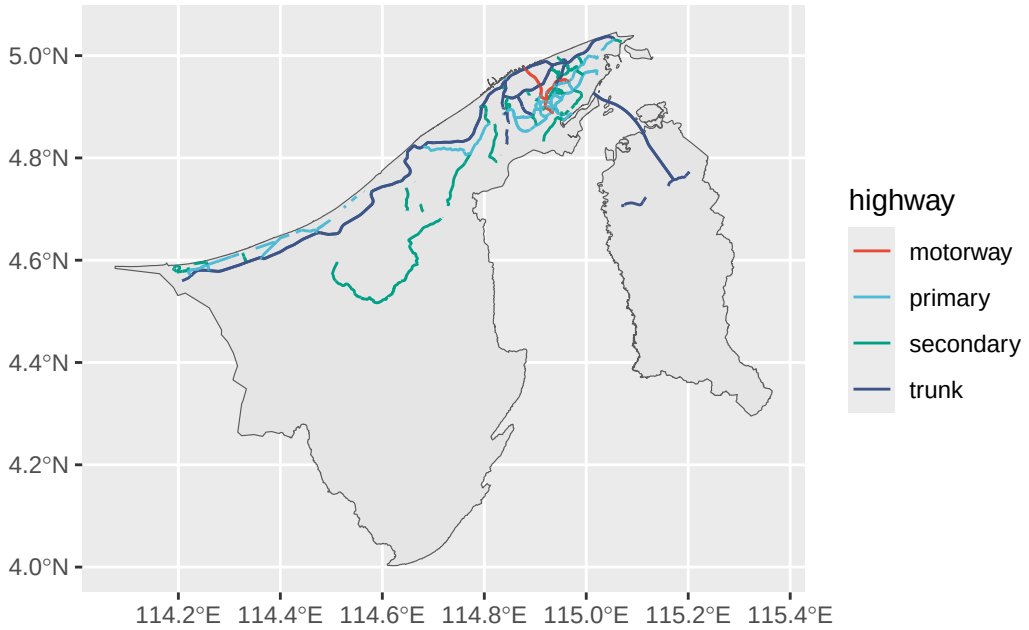
```
brd_mjr <-
  brd |>
  filter(highway %in% c("motorway", "trunk", "primary", "secondary"))
brd_mjr
```

```
Simple feature collection with 1887 features and 14 fields
Geometry type: LINESTRING
```

```
Dimension:     XY
Bounding box:  xmin: 114.1906 ymin: 4.516642 xmax: 115.2021 ymax: 5.037115
Geodetic CRS:  WGS 84
# A tibble: 1,887 x 15
   name     `name:en` highway surface smoothness width lanes oneway bridge layer
 * <chr>    <chr>     <chr>   <chr>   <chr>      <chr> <chr> <chr>  <chr>  <chr>
 1 Lebuhra~ Tutong-T~ trunk   asphalt <NA>       <NA>  2     yes    <NA>   <NA>
 2 Lebuhra~ Tutong-T~ trunk   asphalt <NA>       <NA>  3     yes    <NA>   <NA>
 3 Jalan S~ <NA>      primary asphalt <NA>       <NA>  2     yes    yes    1
 4 Jalan S~ <NA>      primary asphalt <NA>       <NA>  2     yes    <NA>   <NA>
 5 Lebuh R~ Seria-Be~ trunk   asphalt <NA>       <NA>  2     yes    <NA>   <NA>
 6 <NA>     <NA>      trunk   asphalt <NA>       <NA>  2     yes    <NA>   <NA>
 7 <NA>     <NA>      primary asphalt <NA>       <NA>  1     yes    <NA>   <NA>
 8 Lebuh R~ Seria-Be~ trunk   asphalt <NA>       <NA>  2     yes    yes    1
 9 <NA>     <NA>      primary asphalt <NA>       <NA>  2     yes    <NA>   <NA>
10 Lebuhra~ Telisai-~ trunk   asphalt <NA>       <NA>  2     yes    <NA>   <NA>
# i 1,877 more rows
# i 5 more variables: source <chr>, `name:ms` <chr>, osm_id <int>,
#   osm_type <chr>, geometry <LINESTRING [°]>
```

And now a plot of these roads.

```
ggplot() +
  geom_sf(data = brn_sf) +
  geom_sf(data = brd_mjr, aes(col = highway), size = 0.5) +
  # scale_colour_viridis_d(option = "turbo")
  ggsci::scale_colour_npg()
```



With this, I asked ChatGPT what kind of spatial analyses can be done on this data set. It said, when paired with appropriate data, we can do things like:

1. **Network Connectivity Analysis**
   - Assess reachability and identify disconnected road network components.
2. **Accessibility and Service Area Analysis**
   - Determine service areas and catchment areas for essential services.
3. **Traffic Simulation and Management**
   - Simulate traffic flow to identify bottlenecks and suggest optimal routing.
4. **Environmental Impact Assessment**
   - Estimate vehicular emissions and model noise pollution from roads.
5. **Urban and Regional Planning**
   - Examine land use compatibility and assess infrastructure development needs.
6. **Safety Analysis**
   - Identify accident hotspots and assess pedestrian safety.
7. **Economic Analysis**
   - Evaluate economic accessibility and the impact of road projects.

Let's pick one of these: Calculate the distance between the centroid of several regions and the major hospital in the Belait district. This analysis guides urban and healthcare planning by pinpointing areas with inadequate access to emergency services, enabling targeted infrastructure and service improvements.

2.3.1 Road networks in Belait region

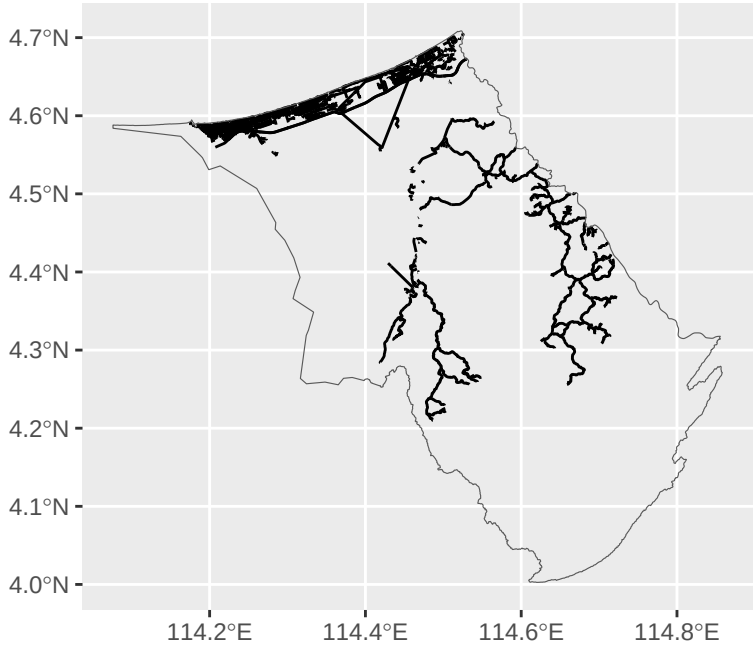
> What we'll learn
>
> - Manipulating GIS data using `sf::st_intersection()` and the like. Useful for reorganising the spatial structure (without having to do this in QGIS or ArcGIS).
> - Sampling points from a line data set.
> - Calculating distances between points and lines using `{osrm}` package.

First we "crop" the road network to the Belait region.

```
brd_belait <- st_intersection(
  brd,
  filter(dis_sf, name == "Belait")
)
```

```
Warning: attribute variables are assumed to be spatially constant throughout
all geometries
```

```
ggplot(brd_belait) +
  geom_sf() +
  geom_sf(data = filter(dis_sf, name == "Belait"), fill = NA)
```



If we were to sample random points from the Belait polygon, we might get non-sensical areas like the extremely rural areas or forest reserves. So the idea is to sample random points from the road network itself. For this, we need a function that will get us a random point on the path itself.

```
get_random_point <- function(linestring) {
  coords <- st_coordinates(linestring)
  samp_coord <- coords[sample(nrow(coords), 1), , drop = FALSE]
  samp_coord[, 1:3]
}
get_random_point(brd_belait$geometry[1])
```

```
         X          Y         L1
114.241433   4.594193   1.000000
```

Once we have this function, we need to `map()` this function onto each of the linestrings in the `brd_belait` data set. The resulting list of points is too large! So we will just sample 100 points (you can experiment with this number).

```
random_points <-
  map(brd_belait$geometry, get_random_point) |>
  bind_rows() |>
  slice_sample(n = 100)
```

What we have now is a data frame of 100 random points on the road network in the Belait district. We will use the `{osrm}` package to calculate the distance between these points and the Suri Seri Begawan Hospital in Kuala Belait. The output will be three things: 1) The duration (minutes); 2) The distance (km); and 3) a `LINESTRING` object that represents the path to get to the hospital. Unfortunately the `osrmRoute()` function is not vectorised, i.e. we have to do it one-by-one for each of the 100 points. Luckily, we can just make a `for` loop and store the results in a list.

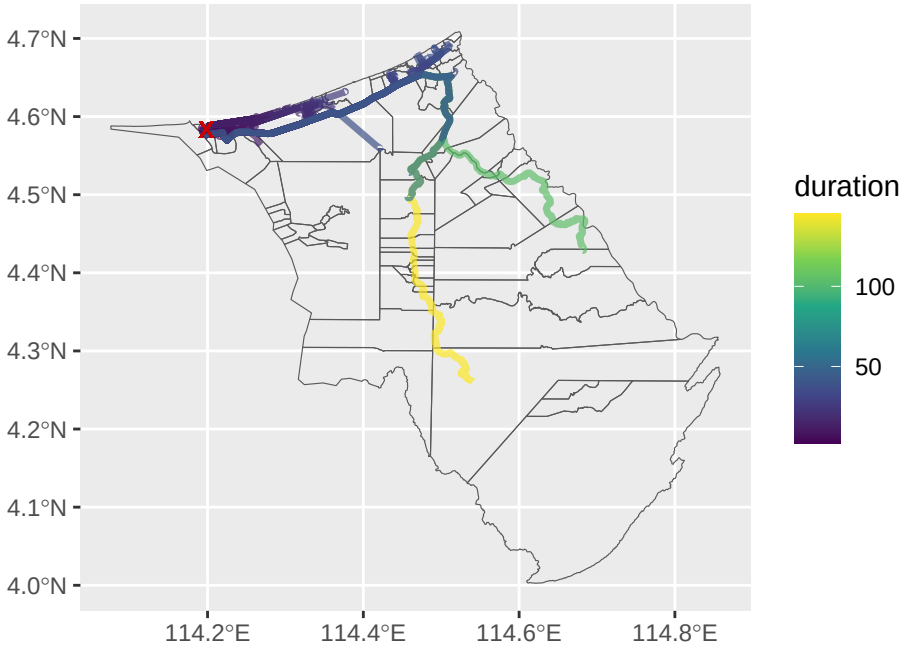
```
suriseri <- c(114.198778, 4.583444)

res <- list()
for (i in 1:100) {
  res[[i]] <- osrmRoute(src = random_points[i, 1:2], dst = suriseri, overview = "full")
}
res <-
  bind_rows(res) |>
  as_tibble() |>
  st_as_sf()
res
```

```
Simple feature collection with 100 features and 4 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: 114.1916 ymin: 4.26076 xmax: 114.6847 ymax: 4.69583
Geodetic CRS:  WGS 84
# A tibble: 100 x 5
   src   dst   duration distance                                        geometry
   <chr> <chr>    <dbl>    <dbl>                                <LINESTRING [°]>
 1 1     dst       31.0     36.3  (114.4818 4.65235, 114.4825 4.65219, 114.4827 ~
 2 1     dst       55.0     56.5  (114.4598 4.52845, 114.4599 4.52838, 114.46 4.~
 3 1     dst        8.64     7.60 (114.2471 4.58019, 114.2469 4.58015, 114.2468 ~
 4 1     dst      145.      99.7  (114.5408 4.26076, 114.5408 4.26076, 114.5396 ~
 5 1     dst        4.19     2.32 (114.2142 4.58474, 114.2142 4.58474, 114.2142 ~
 6 1     dst       30.5     36.3  (114.4812 4.66546, 114.4812 4.66552, 114.4812 ~
 7 1     dst        4.07     2.39 (114.216 4.59037, 114.216 4.59037, 114.2153 4.~
 8 1     dst        4.70     2.98 (114.2201 4.58964, 114.2199 4.59097, 114.2197 ~
 9 1     dst       19.1     15.6  (114.3259 4.61362, 114.3243 4.61309, 114.3227 ~
10 1     dst        6.94     5.36 (114.2361 4.58249, 114.2361 4.58249, 114.2358 ~
# i 90 more rows
```

So with all that done, we can now plot the paths taken by the 100 random points to the hospital. The map gives us an indication of which areas are underserved by the hospital,

and can guide urban and healthcare planning by pinpointing areas with inadequate access to emergency services, enabling targeted infrastructure and service improvements.

```
ggplot(res) +
  # geom_point(data = random_points, aes(x = X, y = Y), col = "red") +
  geom_sf(data = filter(kpg_sf, district == "Belait"), fill = NA) +
  geom_sf(aes(col = duration), linewidth = 1.2, alpha = 0.7) +
  geom_point(x = suriseri[1], y = suriseri[2], col = "red3", pch = "X",
             size = 3) +
  scale_colour_viridis_c()
```



Improving the analysis

- Weight analysis by populous areas. Outcalls to hospitals can be modelled using a Poisson distribution with the population as the rate parameter.
- Use a more sophisticated routing algorithm that accounts for traffic conditions and road quality (am vs pm, weekends vs weekdays, etc.).
- Simpler to analyse at the kampong or mukim level?

2.4 Areal data (`(MULTI)POLYGONS`)

> What we'll learn
>
> - Represent statistical data using colour mapping symbology (choropleth)
> - Use `ggplot2::geom_label()` or `ggrepel::geom_label_repel()` to add labels to the map
> - Using a binned colour scale, e.g. `ggplot2::geom_scale_fill_viridis_b()`

When your study data is made up a finite number of non-overlapping areas, then you can represent them as polygons in R. This is the case for the kampong and mukim data in Brunei. As an example, let us look at the population of each kampong in Brunei. This dataset comes from the 2021 Brunei Census data (DEPS 2022)

```
glimpse(bn_census2021)
```

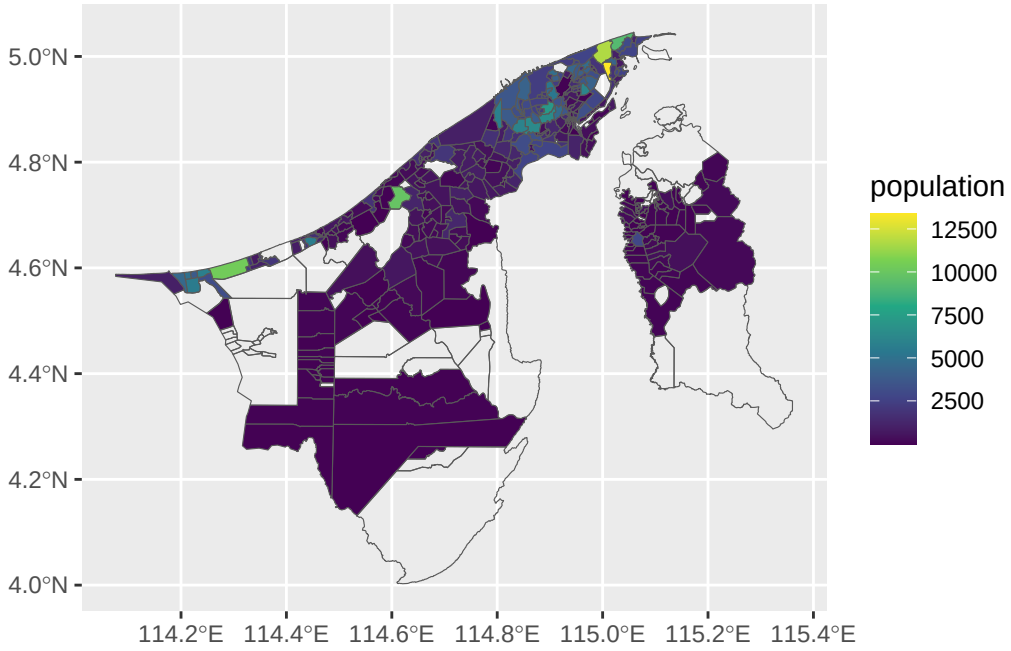
```
Rows: 365
Columns: 11
$ id           <dbl> 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 2~
$ kampong      <chr> "Kg. Biang", "Kg. Amo", "Kg. Sibut", "Kg. Sumbiling Baru"~
$ mukim        <chr> "Mukim Amo", "Mukim Amo", "Mukim Amo", "Mukim Amo", "Muki~
$ district     <chr> "Temburong", "Temburong", "Temburong", "Temburong", "Temb~
$ population   <dbl> 75, 394, 192, 91, 108, 143, 199, 123, 95, 90, 92, 2427, 4~
$ pop_male     <dbl> 46, 218, 98, 48, 60, 68, 115, 65, 52, 46, 73, 1219, 252, ~
$ pop_female   <dbl> 29, 176, 94, 43, 48, 75, 84, 58, 43, 44, 19, 1208, 150, 2~
$ pop_bruneian <dbl> 37, 280, 174, 55, 57, 64, 114, 88, 63, 35, 37, 1557, 235,~
$ pop_pr       <dbl> 33, 83, 17, 24, 41, 64, 64, 28, 29, 32, 2, 179, 3, 67, 32~
$ household    <dbl> 13, 83, 37, 23, 23, 23, 38, 26, 26, 23, 14, 517, 76, 691,~
$ occ_liv_q    <dbl> 13, 62, 27, 16, 22, 21, 37, 22, 12, 23, 14, 492, 71, 681,~
```

Each row of the data refers to a kampong-level observation. While there are unique identifiers to this (`id`, `kampong`, `mukim`, `district`), we would still need to geocode this data set so that we can do fun things like plot it on a map. Let's use (again) `left_join()` to do this.

```
bn_pop_sf <-
  left_join(
    kpg_sf,
    bn_census2021,
    by = join_by(id, kampong, mukim, district)
  )
```

Great. Let's take a look at the population column. It would be very interesting to see where most of the 440,704 people of Brunei live!

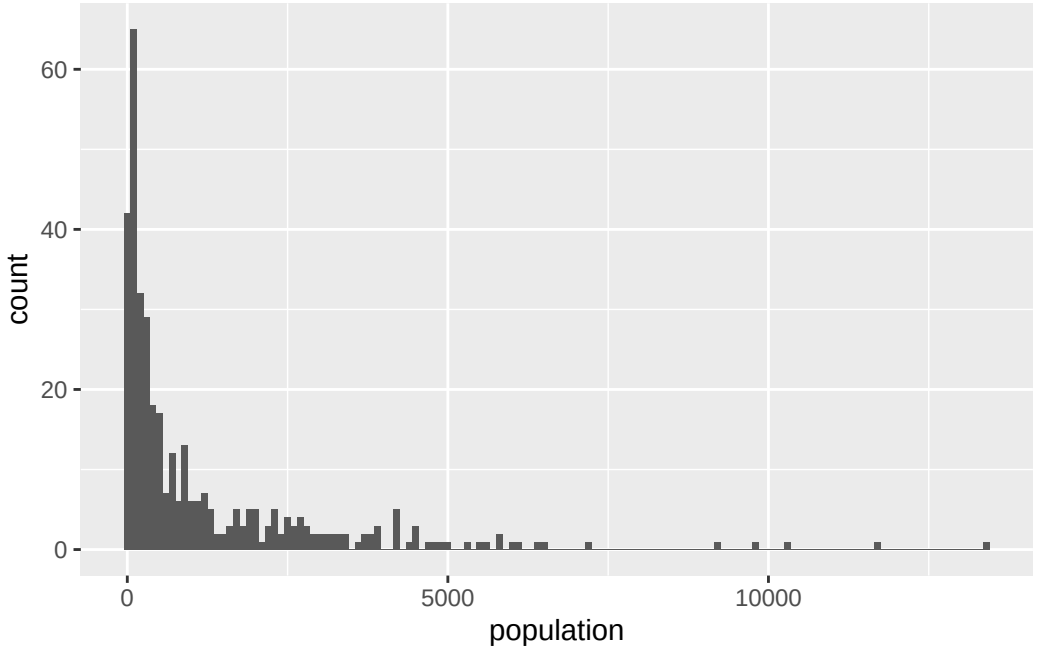
```
ggplot(bn_pop_sf) +
  geom_sf(aes(fill = population)) +
  scale_fill_viridis_c(na.value = NA)
```



As expected, there are "hotspots" of population in the Brunei-Muara district, and to a lesser extent in the Belait district. We can make this graph a bit better by binning the population values. It seems to be dominated by a lot of these low value colours. Let's take a look at this further by inspecting a histogram.

```
ggplot(bn_pop_sf) +
  geom_histogram(aes(population), binwidth = 100)
```

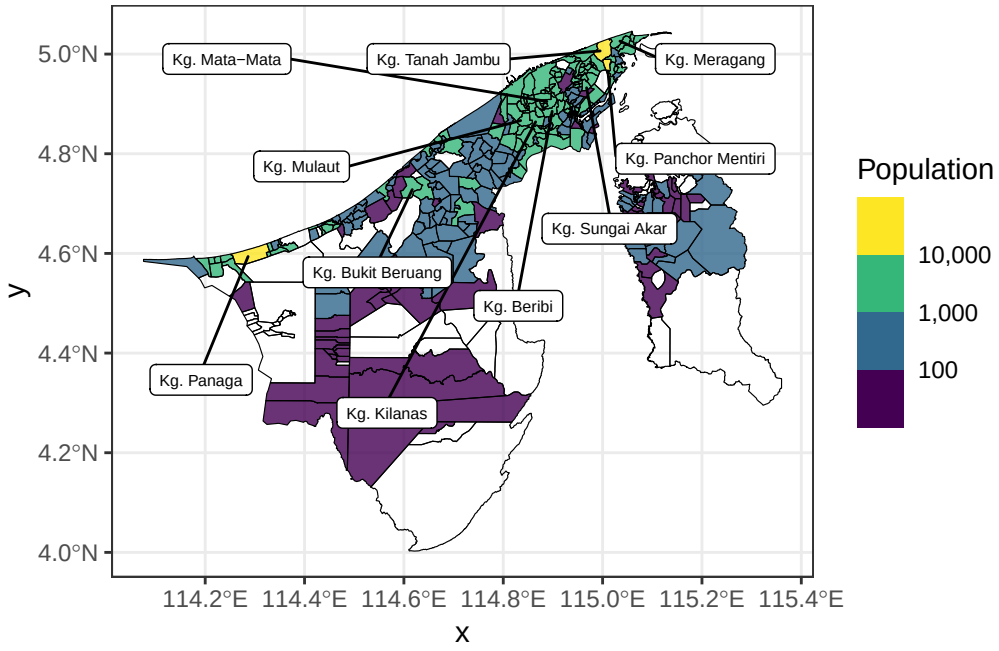
```
Warning: Removed 75 rows containing non-finite outside the scale range
(`stat_bin()`).
```



So maybe we can bin the population into 4 categories: $<$ 100, 101-1000, 1001-10000, and 10000+. For this we directly use the `scale_fill_viridis_b()` and adjust the breaks. Otherwise we would have to `cut()` the population column and then use `scale_fill_manual()`. We also added the names of the top 10 most populous kampongs to the map using `ggrepel::geom_label_repel()`.

```
kpg_labels_sf <-
  bn_pop_sf |>
  arrange(desc(population)) |>
  slice_head(n = 10)

bn_pop_sf |>
  # filter(population > 50) |>
  ggplot() +
  geom_sf(aes(fill = population), col = NA, alpha = 0.8) +
  geom_sf(data = kpg_sf, fill = NA, col = "black") +
  ggrepel::geom_label_repel(
    data = kpg_labels_sf,
    aes(label = kampong, geometry = geometry),
    stat = "sf_coordinates",
    inherit.aes = FALSE,
    box.padding = 1,
    size = 2,
    max.overlaps = Inf
```

```
  ) +
  scale_fill_viridis_b(
    name = "Population",
    na.value = NA,
    labels = scales::comma,
    breaks = c(0, 100, 1000, 10000, 20000)
    # limits = c(0, 12000)
  ) +
  theme_bw()
```



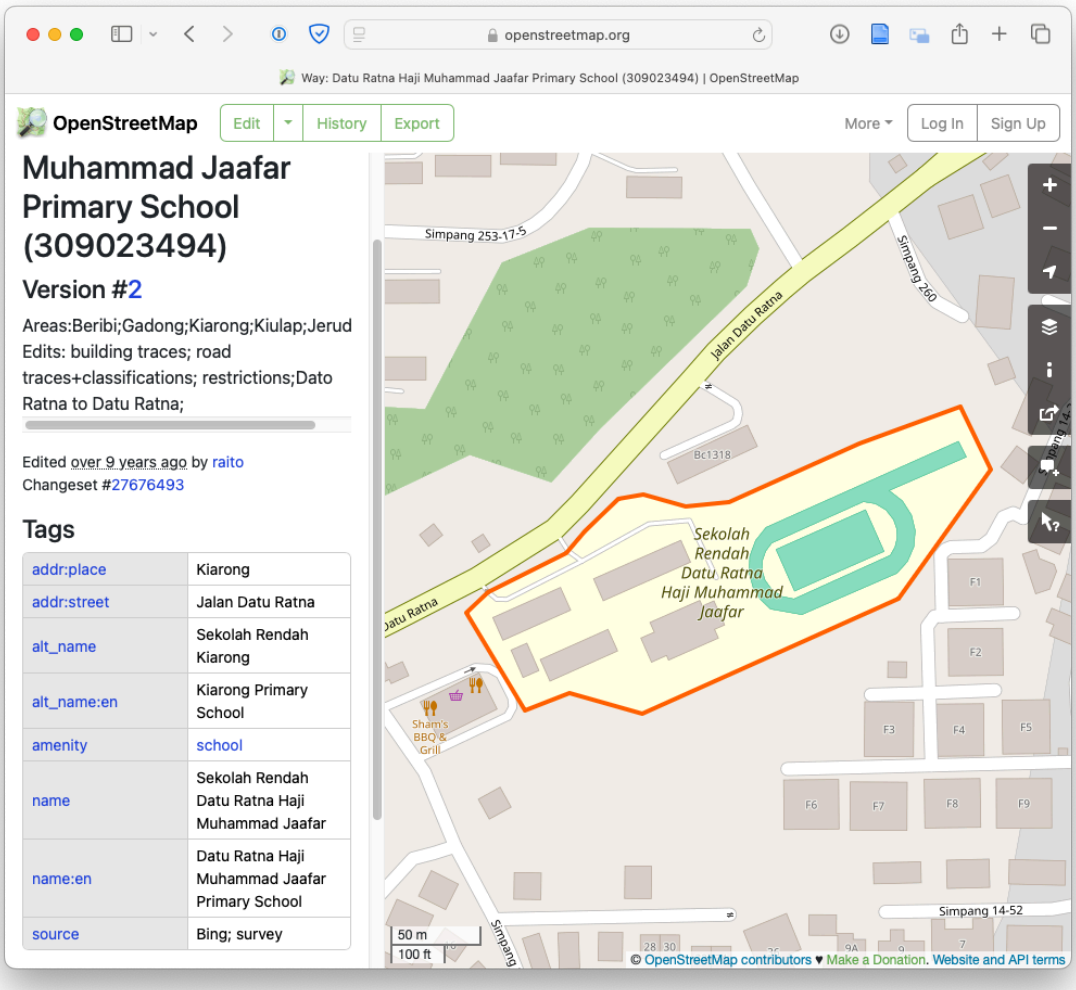
2.5 OpenStreetMap data

> What we'll learn
>
> - How to scrape OpenStreetMap data using the `{osmdata}` package.

The `{osmdata}` package is a very useful tool for scraping OpenStreetMap data. It allows you to download data from OpenStreetMap and convert it into an `sf` object. The package is built on top of the `osmdata` API, which is a wrapper around the Overpass API. The Overpass API is a read-only API that allows you to query OpenStreetMap data. Conveniently, we do not need an API key.

2.5.1 EXAMPLE: How to get all the schools in Brunei

When we go to https://www.openstreetmap.org/ website, we can search for some key terms. For example, if we search for "Sekolah Rendah Kiarong", we see the following:



Highlighted in red is the polygon that represents the school. Furthermore, we have some information in the "Tags" section such as:

- `addr:place` = Kiarong
- `addr:street` = Jalan Datu Ratna
- `alt_name` = Sekolah Rendah Kiarong
- `alt_name:en` = Kiarong Primary School
- `amenity` = school

- etc.

The `{osmdata}` package allows us to query this information. To replicate this 'GUI' experience using code, we do the following:

```
q <-
  opq("brunei") |>
  add_osm_feature(
    key = "name",
    value = "Sekolah Rendah Datu Ratna Haji Muhammad Jaafar"
  ) |>
  osmdata_sf()
print(q)
```

```
Object of class 'osmdata' with:
                 $bbox : 4.002508,113.017925,6.546584,115.3635623
        $overpass_call : The call submitted to the overpass API
                 $meta : metadata including timestamp and version numbers
           $osm_points : 'sf' Simple Features Collection with 16 points
            $osm_lines : NULL
         $osm_polygons : 'sf' Simple Features Collection with 1 polygons
       $osm_multilines : NULL
    $osm_multipolygons : NULL
```

It has found the school. To extract the information, let's look at the `$osm_polygons` entry:

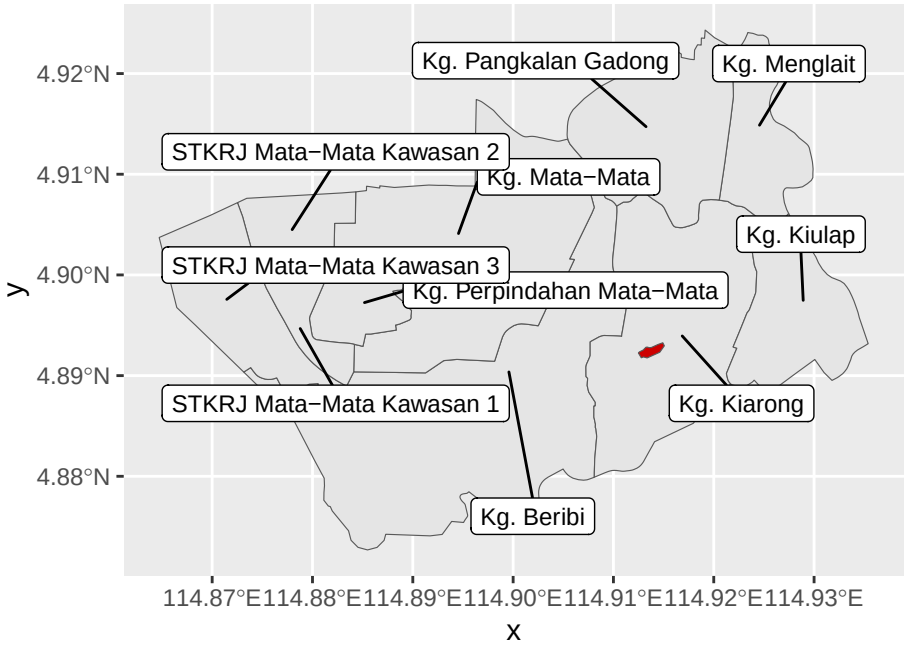
```
glimpse(q$osm_polygons)
```

```
Rows: 1
Columns: 10
$ osm_id        <chr> "309023494"
$ name          <chr> "Sekolah Rendah Datu Ratna Haji Muhammad Jaafar"
$ `addr:place`  <chr> "Kiarong"
$ `addr:street` <chr> "Jalan Datu Ratna"
$ alt_name      <chr> "Sekolah Rendah Kiarong"
$ `alt_name:en` <chr> "Kiarong Primary School"
$ amenity       <chr> "school"
$ `name:en`     <chr> "Datu Ratna Haji Muhammad Jaafar Primary School"
$ source        <chr> "Bing; survey"
$ geometry      <POLYGON [°]> POLYGON ((114.9125 4.892252...
```

Let's plot it!

```
# warning: false
ggplot(filter(kpg_sf, mukim == "Mukim Gadong B")) +
  geom_sf() +
  geom_label_repel(
    aes(label = kampong, geometry = geometry),
    stat = "sf_coordinates",
    inherit.aes = FALSE,
    box.padding = 1,
    size = 3,
    max.overlaps = Inf
  ) +
  geom_sf(data = q$osm_polygons, fill = "red3")
```

```
Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
give correct results for longitude/latitude data
```



We can query based on amenity type as well. For example, to get all the schools in Brunei:

```
# Bounding box for Brunei Muara
bm_sf <- filter(kpg_sf, district == "Brunei Muara")
bm_bbox <- st_bbox(bm_sf)
```

```
q <-
  opq(bm_bbox) |>
  add_osm_feature(
    key = "amenity",
    value = "school"
  ) |>
  osmdata_sf()
print(q)
```

```
Object of class 'osmdata' with:
                 $bbox : 4.72903834429411,114.771346735899,5.04587807206061,115.138720231749
        $overpass_call : The call submitted to the overpass API
                 $meta : metadata including timestamp and version numbers
           $osm_points : 'sf' Simple Features Collection with 1321 points
            $osm_lines : NULL
         $osm_polygons : 'sf' Simple Features Collection with 153 polygons
       $osm_multilines : NULL
    $osm_multipolygons : 'sf' Simple Features Collection with 1 multipolygons
```

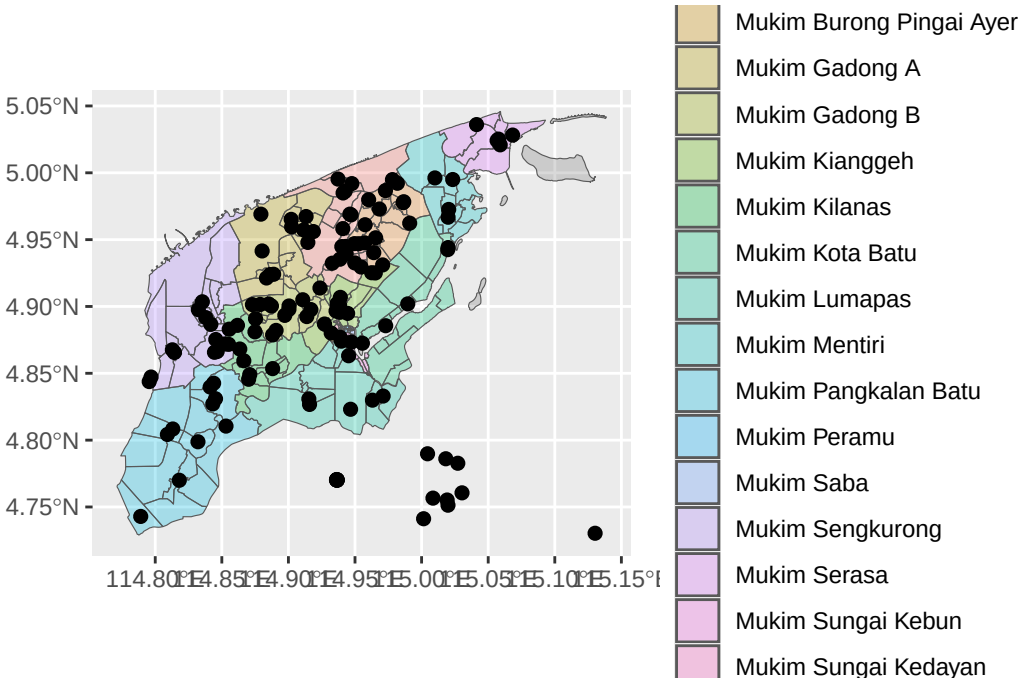
Almost always it is a good idea to look at the polygons, instead of the points. In any case, you can always find the centroid of the polygons if you wanted to plot point data.

```
schools_sf <-
  q$osm_polygons |>
  as_tibble() |>  # these two lines convert to tibble-like object
  st_as_sf() |>
  select(osm_id, name) |>
  drop_na() |>
  st_centroid()  # obtains X,Y coordinates of centroids

print(schools_sf)
```

```
Simple feature collection with 138 features and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 114.7891 ymin: 4.730341 xmax: 115.1303 ymax: 5.036068
Geodetic CRS:  WGS 84
# A tibble: 138 x 3
   osm_id    name                                              geometry
 * <chr>     <chr>                                          <POINT [°]>
```

```
 1 45517438  Sekolah Rendah Haji Tarif                   (114.9321 4.88012)
 2 45768022  Sekolah Menengah Awang Semaun               (114.9389 4.876925)
 3 45820441  Sekolah Rendah Pengiran Anak Puteri Besar (114.9397 4.874045)
 4 45820563  Pehin Dato Jamil Primary School            (114.9473 4.873318)
 5 157197463 Sekolah Ugama Pengiran Muda Abdul Malik ~ (114.8709 4.848966)
 6 157489516 Sekolah Rendah Dato Marsal                 (114.9576 4.961157)
 7 167974917 Chung Hwa Middle School                    (114.9445 4.894822)
 8 167974963 Sekolah Rendah Pusar Ulak                  (114.9358 4.896647)
 9 167974968 St. Andrew's School                        (114.9372 4.896313)
10 260696860 Jerudong International School              (114.8793 4.969056)
# i 128 more rows
```

```
ggplot() +
  geom_sf(data = bm_sf, aes(fill = mukim), alpha = 0.3) +
  geom_sf(data = schools_sf, size = 2)
```



From here…

- Visit the OSM Wiki to see what other amenities you can query.
- Clearly not limited to schools – clinics, shops, movie theatres, …
- Combine with the road data from `{osrm}` to calculate distances between schools and hospitals, for example.

3 Quantitative analysis of textual data

- https://tutorials.quanteda.io/introduction/

3.1 Introduction

There are several R packages used for quantitative text analysis, but we will focus specifically on the `{quanteda}` package. So, first install the package from CRAN:

```
install.packages("quanteda")
```

Since the release of `{quanteda}` version 3.0, `textstat_*`, `textmodel_*` and `textplot_*` functions are available in separate packages. We will use several of these functions in the chapters below and strongly recommend installing these packages.

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

We will use the `{readtext}` package to read in different types of text data in these tutorials.

```
install.packages("readtext")
```

3.2 Quantitative data

Before beginning we need to load the libraries

```
library(tidyverse)
library(quanteda)
library(quanteda.textmodels)
library(quanteda.textstats)
library(quanteda.textplots)
library(readtext)
```

And these ones lates for the modelling section:

```
library(seededlda)
library(LSX)
library(lubridate)
library(ggdendro)
```

3.2.1 Pre-formatted files

If your text data is stored in a pre-formatted file where one column contains the text and additional columns might store document-level variables (e.g. year, author, or language), you can import this into R using `read_csv()`.

```
path_data <- system.file("extdata/", package = "readtext")
dat_inaug <- read_csv(paste0(path_data, "/csv/inaugCorpus.csv"))
```

```
Rows: 5 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (3): texts, President, FirstName
dbl (1): Year

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(dat_inaug)
```

```
Rows: 5
Columns: 4
$ texts     <chr> "Fellow-Citizens of the Senate and of the House of Represent~
$ Year      <dbl> 1789, 1793, 1797, 1801, 1805
$ President <chr> "Washington", "Washington", "Adams", "Jefferson", "Jefferson"
$ FirstName <chr> "George", "George", "John", "Thomas", "Thomas"
```

The data set is about the inaugural speeches of the US presidents. So as we can see the data set is arranged in tabular form, with 5 rows and 4 columns. The columns are `texts`, `Year`, `President`, and `FirstName`.

Alternatively, you can use the `{readtext}` package to import character (comma- or tab-separated) values. `{readtext}` reads files containing text, along with any associated document-level variables. As an example, consider the following tsv file:

```
tsv_file <- paste0(path_data, "/tsv/dailsample.tsv")
cat(readLines(tsv_file, n = 4), sep = "\n")  # first 3 lines
```

```
speechID    memberID    partyID constID title   date    member_name party_name  const_name  s
1   977 22  158 1. CEANN COMHAIRLE I gCOIR AN LAE.  1919-01-21  Count George Noble, Count Plu
2   1603    22  103 1. CEANN COMHAIRLE I gCOIR AN LAE.  1919-01-21  Mr. Pádraic Ó Máille    S
3   116 22  178 1. CEANN COMHAIRLE I gCOIR AN LAE.  1919-01-21  Mr. Cathal Brugha   Sinn Féi
```

The document itself in raw format is arranged in tabular form, separated by tabs. Each row contains a "document" (in this case, a speech) and the columns contain **document-level** variables. The column that contains the actual speech is named `speech`. To import this using `{readtext}`, you can use the following code:

```
dat_dail <- readtext(tsv_file, text_field = "speech")
glimpse(dat_dail)
```

```
Rows: 33
Columns: 11
$ doc_id      <chr> "dailsample.tsv.1", "dailsample.tsv.2", "dailsample.tsv.3"~
$ text        <chr> "Molaimse don Dáil Cathal Brugha, an Teachta ó Dhéisibh Ph~
$ speechID    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ memberID    <int> 977, 1603, 116, 116, 116, 116, 496, 116, 116, 2095, 116, 1~
$ partyID     <int> 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22~
$ constID     <int> 158, 103, 178, 178, 178, 178, 46, 178, 178, 139, 178, 178,~
$ title       <chr> "1. CEANN COMHAIRLE I gCOIR AN LAE.", "1. CEANN COMHAIRLE ~
$ date        <chr> "1919-01-21", "1919-01-21", "1919-01-21", "1919-01-21", "1~
$ member_name <chr> "Count George Noble, Count Plunkett", "Mr. Pádraic Ó Máill~
$ party_name  <chr> "Sinn Féin", "Sinn Féin", "Sinn Féin", "Sinn Féin", "Sinn ~
$ const_name  <chr> "Roscommon North", "Galway Connemara", "Waterford County",~
```

3.2.2 Multiple text files

A second option to import data is to load multiple text files at once that are stored in the same folder or subfolders. Again, `path_data` is the location of sample files on your computer. Unlike the pre-formatted files, individual text files usually do not contain document-level variables. However, you can create document-level variables using the `{readtext}` package.

The directory `/txt/UDHR` contains text files (".txt") of the Universal Declaration of Human Rights in 13 languages.

```
path_udhr <- paste0(path_data, "/txt/UDHR")
list.files(path_udhr)  # list the files in this folder
```

```
 [1] "UDHR_chinese.txt"    "UDHR_czech.txt"      "UDHR_danish.txt"
 [4] "UDHR_english.txt"    "UDHR_french.txt"     "UDHR_georgian.txt"
 [7] "UDHR_greek.txt"      "UDHR_hungarian.txt"  "UDHR_icelandic.txt"
[10] "UDHR_irish.txt"      "UDHR_japanese.txt"   "UDHR_russian.txt"
[13] "UDHR_vietnamese.txt"
```

Each one of these txt files contains the text of the UDHR in the specific language. For instance, to inspect what each one of these files contain, we do the following:

```
# just first 5 lines
cat(readLines(file.path(path_udhr, "UDHR_chinese.txt"), n = 5), sep = "\n")
```

```
        217A(III)     1948   12   10


        ,                                  ,
```

To import these files, you can use the following code:

```
dat_udhr <- readtext(path_udhr)
glimpse(dat_udhr)
```

```
Rows: 13
Columns: 2
$ doc_id <chr> "UDHR_chinese.txt", "UDHR_czech.txt", "UDHR_danish.txt", "UDHR_~
$ text   <chr> "    \n         217A(III)  ~
```

> **Note**
>
> If you are using Windows, you need might need to specify the encoding of the file by adding `encoding = "utf-8"`. In this case, imported texts might appear like `<U+4E16><U+754C><U+4EBA><U+6743>` but they indicate that Unicode charactes are imported correctly.

Here's another example of multiple text files. The directory `/txt/EU_manifestos` contains text files (".txt") of the European Union manifestos in different languages.

```
path_eu <- paste0(path_data, "/txt/EU_manifestos/")
list.files(path_eu)  # list the files in this folder
```

```
 [1] "EU_euro_2004_de_PSE.txt" "EU_euro_2004_de_V.txt"
 [3] "EU_euro_2004_en_PSE.txt" "EU_euro_2004_en_V.txt"
 [5] "EU_euro_2004_es_PSE.txt" "EU_euro_2004_es_V.txt"
 [7] "EU_euro_2004_fi_V.txt"   "EU_euro_2004_fr_PSE.txt"
 [9] "EU_euro_2004_fr_V.txt"   "EU_euro_2004_gr_V.txt"
[11] "EU_euro_2004_hu_V.txt"   "EU_euro_2004_it_PSE.txt"
[13] "EU_euro_2004_lv_V.txt"   "EU_euro_2004_nl_V.txt"
[15] "EU_euro_2004_pl_V.txt"   "EU_euro_2004_se_V.txt"
[17] "EU_euro_2004_si_V.txt"
```

You can generate document-level variables based on the file names using the `docvarnames` and `docvarsfrom` argument. `dvsep = "_"` specifies the value separator in the filenames. `encoding = "ISO-8859-1"` determines character encodings of the texts. Notice how the document variables are nicely generated from the file names.

```
dat_eu <- readtext(
  file = path_eu,
  docvarsfrom = "filenames",
  docvarnames = c("unit", "context", "year", "language", "party"),
  dvsep = "_",
  encoding = "ISO-8859-1"
)
glimpse(dat_eu)
```

```
Rows: 17
Columns: 7
$ doc_id   <chr> "EU_euro_2004_de_PSE.txt", "EU_euro_2004_de_V.txt", "EU_euro_~
$ text     <chr> "PES · PSE · SPE European Parliament rue Wiertz B 1047 Brusse~
$ unit     <chr> "EU", "EU", "EU", "EU", "EU", "EU", "EU", "EU", "EU", "EU", "~
$ context  <chr> "euro", "euro", "euro", "euro", "euro", "euro", "euro", "euro~
$ year     <int> 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2004, 2~
$ language <chr> "de", "de", "en", "en", "es", "es", "fi", "fr", "fr", "gr", "~
$ party    <chr> "PSE", "V", "PSE", "V", "PSE", "V", "V", "PSE", "V", "V", "V"~
```

3.2.3 JSON

You can also read JSON files (.json) downloaded from the Twititer stream API. twitter.json is located in data directory of this tutorial package.

The JSON file looks something like this

```
{"created_at":"Wed Jun 07 23:30:01 +0000 2017","id":872596537142116352,"id_str":"8725965371
"source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitt
"in_reply_to_status_id_str":"872596176834572288","in_reply_to_user_id":4556760676,"in_reply_
"verified":false,"followers_count":367,"friends_count":304,"listed_count":1,"favourites_coun
"profile_link_color":"1DA1F2","profile_sidebar_border_color":"C0DEED","profile_sidebar_fill_
"profile_image_url_https":"https:\/\/pbs.twimg.com\/profile_images\/870447188400365568\/RiR1
"place":null,"contributors":null,"is_quote_status":false,"retweet_count":0,"favorite_count":
```

It's a little hard to parse, but luckily we just leave it to the `{readtext}` package to do the job for us.

```
dat_twitter <- readtext("../data/twitter.json", source = "twitter")
```

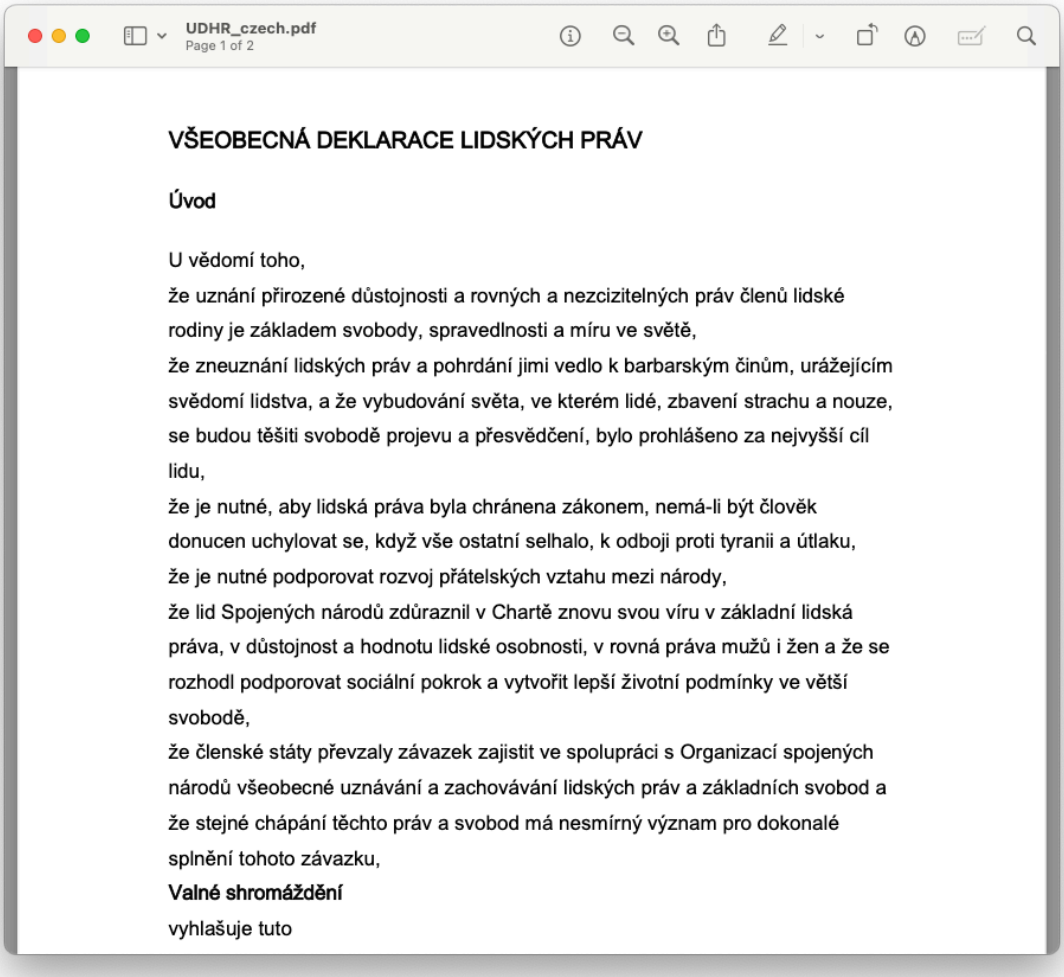
The file comes with several metadata for each tweet, such as the number of retweets and likes, the username, time and time zone.

```
head(names(dat_twitter))
```

```
## [1] "doc_id"         "text"           "retweet_count"  "favorite_count"
## [5] "favorited"      "truncated"
```

3.2.4 PDF

`readtext()` can also convert and read PDF (".pdf") files. The directory `/pdf/UDHR` contains PDF files of the Universal Declaration of Human Rights in 13 languages. Each file looks like this:




```
dat_udhr <- readtext(
  paste0(path_data, "/pdf/UDHR/*.pdf"),
  docvarsfrom = "filenames",
  docvarnames = c("document", "language"),
  sep = "_"
)
print(dat_udhr)
```

```
readtext object consisting of 11 documents and 2 docvars.
# A data frame: 11 x 4
  doc_id           text                          document language
```

```
  <chr>            <chr>                          <chr>    <chr>
1 UDHR_chinese.pdf "\"   \n\n \"..." UDHR     chinese
2 UDHR_czech.pdf   "\"VŠEOBECNÁ \"..."           UDHR     czech
3 UDHR_danish.pdf  "\"Den 10. de\"..."           UDHR     danish
4 UDHR_english.pdf "\"Universal \"..."           UDHR     english
5 UDHR_french.pdf  "\"Déclaratio\"..."           UDHR     french
6 UDHR_greek.pdf   "\"OIKOTMENIK\"..."           UDHR     greek
# i 5 more rows
```

3.2.5 Microsoft Word

Finally, `readtext()` can import Microsoft Word (".doc" and ".docx") files.

```
dat_word <- readtext(paste0(path_data, "/word/*.docx"))
print(dat_udhr)
```

```
readtext object consisting of 11 documents and 2 docvars.
# A data frame: 11 x 4
  doc_id           text                           document language
  <chr>            <chr>                          <chr>    <chr>
1 UDHR_chinese.pdf "\"   \n\n \"..." UDHR     chinese
2 UDHR_czech.pdf   "\"VŠEOBECNÁ \"..."           UDHR     czech
3 UDHR_danish.pdf  "\"Den 10. de\"..."           UDHR     danish
4 UDHR_english.pdf "\"Universal \"..."           UDHR     english
5 UDHR_french.pdf  "\"Déclaratio\"..."           UDHR     french
6 UDHR_greek.pdf   "\"OIKOTMENIK\"..."           UDHR     greek
# i 5 more rows
```

3.3 Workflow

`{quanteda}` has three basic types of objects:

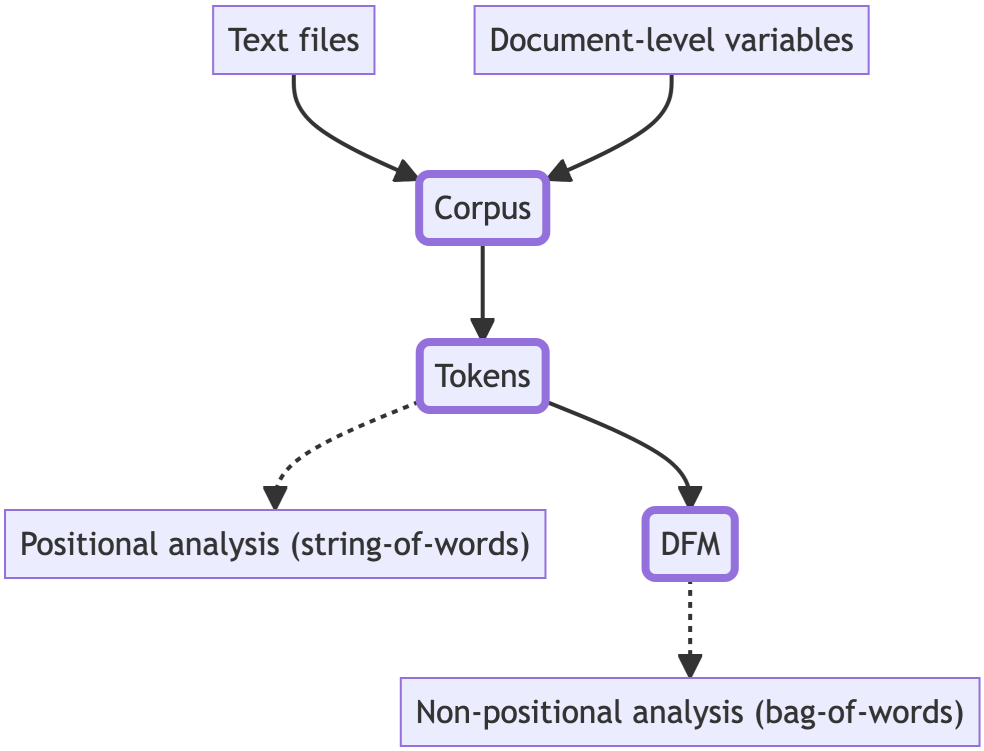
1. Corpus
    - Saves character strings and variables in a data frame
    - Combines texts with document-level variables
2. Tokens
    - Stores tokens in a list of vectors
    - More efficient than character strings, but preserves positions of words

- Positional (string-of-words) analysis is performed using `textstat_collocations()`, `tokens_ngrams()` and `tokens_select()` or `fcm()` with `window` option

3. Document-feature matrix (DFM)

    - Represents frequencies of features in documents in a matrix
    - The most efficient structure, but it does not have information on positions of words
    - Non-positional (bag-of-words) analysis are profrmed using many of the `textstat_*` and `textmodel_*` functions

Text analysis with `{quanteda}` goes through all those three types of objects either explicitly or implicitly.



For example, if character vectors are given to `dfm()`, it internally constructs corpus and tokens objects before creating a DFM.

3.3.1 Corpus

You can create a corpus from various available sources:

1. A character vector consisting of one document per element

2. A data frame consisting of a character vector for documents, and additional vectors for document-level variables

3.3.1.1 Character vector

`data_char_ukimmig2010` is a named character vector and consists of sections of British election manifestos on immigration and asylum.

```
str(data_char_ukimmig2010)
```

```
 Named chr [1:9] "IMMIGRATION: AN UNPARALLELED CRISIS WHICH ONLY THE BNP CAN SOLVE. \n\n- At
 - attr(*, "names")= chr [1:9] "BNP" "Coalition" "Conservative" "Greens" ...
```

```
corp_immig <- corpus(
  data_char_ukimmig2010,
  docvars = data.frame(party = names(data_char_ukimmig2010))
)
print(corp_immig)
```

```
Corpus consisting of 9 documents and 1 docvar.
BNP :
"IMMIGRATION: AN UNPARALLELED CRISIS WHICH ONLY THE BNP CAN S..."

Coalition :
"IMMIGRATION.  The Government believes that immigration has e..."

Conservative :
"Attract the brightest and best to our country. Immigration h..."

Greens :
"Immigration. Migration is a fact of life.  People have alway..."

Labour :
"Crime and immigration The challenge for Britain We will cont..."

LibDem :
"firm but fair immigration system Britain has always been an ..."

[ reached max_ndoc ... 3 more documents ]
```

```
summary(corp_immig)
```

```
Corpus consisting of 9 documents, showing 9 documents:

         Text Types Tokens Sentences        party
          BNP  1125   3280        88          BNP
    Coalition   142    260         4    Coalition
 Conservative   251    499        15 Conservative
       Greens   322    679        21       Greens
       Labour   298    683        29       Labour
       LibDem   251    483        14       LibDem
           PC    77    114         5           PC
          SNP    88    134         4          SNP
         UKIP   346    723        26         UKIP
```

3.3.1.2 Data frame

Using `read_csv()`, load an example file from `path_data` as a data frame called `dat_inaug`. Note that your file does not need to be formatted as `.csv`. You can build a `{quanteda}` corpus from any file format that R can import as a data frame (see, for instance, the **rio** package for importing various files as data frames into R).

```
# set path
path_data <- system.file("extdata/", package = "readtext")

# import csv file
dat_inaug <- read.csv(paste0(path_data, "/csv/inaugCorpus.csv"))
names(dat_inaug)
```

```
[1] "texts"     "Year"      "President" "FirstName"
```

Construct a corpus from the "texts" column in `dat_inaug`.

```
corp_inaug <- corpus(dat_inaug, text_field = "texts")
print(corp_inaug)
```

```
Corpus consisting of 5 documents and 3 docvars.
text1 :
"Fellow-Citizens of the Senate and of the House of Representa..."
```

```
text2 :
"Fellow citizens, I am again called upon by the voice of my c..."

text3 :
"When it was first perceived, in early times, that no middle ..."

text4 :
"Friends and Fellow Citizens: Called upon to undertake the du..."

text5 :
"Proceeding, fellow citizens, to that qualification which the..."
```

3.3.1.3 Document-level variables

{quanteda}'s objects keep information associated with documents. They are called "document-level variables", or "docvars", and are accessed using `docvars()`.

```
corp <- data_corpus_inaugural
head(docvars(corp))
```

```
  Year  President FirstName                 Party
1 1789 Washington    George                  none
2 1793 Washington    George                  none
3 1797      Adams      John            Federalist
4 1801  Jefferson    Thomas Democratic-Republican
5 1805  Jefferson    Thomas Democratic-Republican
6 1809    Madison     James Democratic-Republican
```

If you want to extract individual elements of document variables, you can specify `field`. Or you could just subset it as you normally would a data.frame.

```
docvars(corp, field = "Year")
```

```
 [1] 1789 1793 1797 1801 1805 1809 1813 1817 1821 1825 1829 1833 1837 1841 1845
[16] 1849 1853 1857 1861 1865 1869 1873 1877 1881 1885 1889 1893 1897 1901 1905
[31] 1909 1913 1917 1921 1925 1929 1933 1937 1941 1945 1949 1953 1957 1961 1965
[46] 1969 1973 1977 1981 1985 1989 1993 1997 2001 2005 2009 2013 2017 2021
```

```
corp$Year
```

```
 [1] 1789 1793 1797 1801 1805 1809 1813 1817 1821 1825 1829 1833 1837 1841 1845
[16] 1849 1853 1857 1861 1865 1869 1873 1877 1881 1885 1889 1893 1897 1901 1905
[31] 1909 1913 1917 1921 1925 1929 1933 1937 1941 1945 1949 1953 1957 1961 1965
[46] 1969 1973 1977 1981 1985 1989 1993 1997 2001 2005 2009 2013 2017 2021
```

So that means assignments to *change* document-level variables will work as usual in R. For example, you can change the `Year` variable to a factor (if you wished). And since the output of a `docvars()` function is a data.frame, you could subset or filter as you would a data.frame.

```
docvars(corp) |>
  filter(Year >= 1990)
```

```
  Year President FirstName      Party
1 1993   Clinton      Bill Democratic
2 1997   Clinton      Bill Democratic
3 2001      Bush George W. Republican
4 2005      Bush George W. Republican
5 2009     Obama    Barack Democratic
6 2013     Obama    Barack Democratic
7 2017     Trump Donald J. Republican
8 2021     Biden Joseph R. Democratic
```

```
# {quanteda} also provides corpus_subset() function, but since we learnt about
# dplyr, we can use it here.
```

Another useful feature is the ability to change the unit of texts. For example, the UK Immigration 2010 data set is a corpus of 9 documents, where each document is a speech by the political party.

```
corp <- corpus(data_char_ukimmig2010)
print(corp)
```

```
Corpus consisting of 9 documents.
BNP :
"IMMIGRATION: AN UNPARALLELED CRISIS WHICH ONLY THE BNP CAN S..."

Coalition :
```

```
"IMMIGRATION.  The Government believes that immigration has e..."

Conservative :
"Attract the brightest and best to our country. Immigration h..."

Greens :
"Immigration. Migration is a fact of life.  People have alway..."

Labour :
"Crime and immigration The challenge for Britain We will cont..."

LibDem :
"firm but fair immigration system Britain has always been an ..."

[ reached max_ndoc ... 3 more documents ]
```

We can use `corpus_reshape()` to change the unit of texts. For example, we can change the unit of texts to sentences using the command below.

```
corp_sent <- corpus_reshape(corp, to = "sentences")
print(corp_sent)
```

```
Corpus consisting of 206 documents.
BNP.1 :
"IMMIGRATION: AN UNPARALLELED CRISIS WHICH ONLY THE BNP CAN S..."

BNP.2 :
"The Scale of the Crisis  Britain's existence is in grave per..."

BNP.3 :
"In the absence of urgent action, we, the indigenous British ..."

BNP.4 :
"We, alone of all the political parties, have a decades-long ..."

BNP.5 :
"British People Set to be a Minority within 30 - 50 Years: Th..."

BNP.6 :
"Figures released by the ONS in January 2009 revealed that th..."

[ reached max_ndoc ... 200 more documents ]
```

The following code restores it back to the document level.

```
corp_doc <- corpus_reshape(corp_sent, to = "documents")
print(corp_doc)
```

```
Corpus consisting of 9 documents.
BNP :
"IMMIGRATION: AN UNPARALLELED CRISIS WHICH ONLY THE BNP CAN S..."

Coalition :
"IMMIGRATION.  The Government believes that immigration has e..."

Conservative :
"Attract the brightest and best to our country.  Immigration ..."

Greens :
"Immigration.  Migration is a fact of life.  People have alwa..."

Labour :
"Crime and immigration  The challenge for Britain  We will co..."

LibDem :
"firm but fair immigration system  Britain has always been an..."

[ reached max_ndoc ... 3 more documents ]
```

3.3.2 Tokens

`tokens()` segments texts in a corpus into tokens (words or sentences) by word boundaries. By default, tokens() only removes separators (typically white spaces), but you can also remove punctuation and numbers.

```
toks <- tokens(corp_immig)
print(toks)
```

```
Tokens consisting of 9 documents and 1 docvar.
BNP :
 [1] "IMMIGRATION"  ":"            "AN"           "UNPARALLELED" "CRISIS"
 [6] "WHICH"        "ONLY"         "THE"          "BNP"          "CAN"
[11] "SOLVE"        "."
[ ... and 3,268 more ]
```

```
Coalition :
 [1] "IMMIGRATION" "."           "The"         "Government"  "believes"
 [6] "that"        "immigration" "has"         "enriched"    "our"
[11] "culture"     "and"
[ ... and 248 more ]

Conservative :
 [1] "Attract"     "the"         "brightest"   "and"         "best"
 [6] "to"          "our"         "country"     "."           "Immigration"
[11] "has"         "enriched"
[ ... and 487 more ]

Greens :
 [1] "Immigration" "."           "Migration"   "is"          "a"
 [6] "fact"        "of"          "life"        "."           "People"
[11] "have"        "always"
[ ... and 667 more ]

Labour :
 [1] "Crime"       "and"         "immigration" "The"         "challenge"
 [6] "for"         "Britain"     "We"          "will"        "control"
[11] "immigration" "with"
[ ... and 671 more ]

LibDem :
 [1] "firm"        "but"         "fair"        "immigration" "system"
 [6] "Britain"     "has"         "always"      "been"        "an"
[11] "open"        ","
[ ... and 471 more ]

[ reached max_ndoc ... 3 more documents ]
```

```r
toks_nopunct <- tokens(data_char_ukimmig2010, remove_punct = TRUE)
print(toks_nopunct)
```

```
Tokens consisting of 9 documents.
BNP :
 [1] "IMMIGRATION"  "AN"           "UNPARALLELED" "CRISIS"       "WHICH"
 [6] "ONLY"         "THE"          "BNP"          "CAN"          "SOLVE"
[11] "At"           "current"
[ ... and 2,839 more ]
```

```
Coalition :
 [1] "IMMIGRATION"  "The"          "Government"   "believes"     "that"
 [6] "immigration"  "has"          "enriched"     "our"          "culture"
[11] "and"          "strengthened"
[ ... and 219 more ]

Conservative :
 [1] "Attract"     "the"         "brightest"   "and"         "best"
 [6] "to"          "our"         "country"     "Immigration" "has"
[11] "enriched"    "our"
[ ... and 440 more ]

Greens :
 [1] "Immigration" "Migration"   "is"          "a"           "fact"
 [6] "of"          "life"        "People"      "have"        "always"
[11] "moved"       "from"
[ ... and 598 more ]

Labour :
 [1] "Crime"       "and"         "immigration" "The"         "challenge"
 [6] "for"         "Britain"     "We"          "will"        "control"
[11] "immigration" "with"
[ ... and 608 more ]

LibDem :
 [1] "firm"        "but"         "fair"        "immigration" "system"
 [6] "Britain"     "has"         "always"      "been"        "an"
[11] "open"        "welcoming"
[ ... and 423 more ]

[ reached max_ndoc ... 3 more documents ]
```

You can see how keywords are used in the actual contexts in a concordance view produced by `kwic()`.

```
kw <- kwic(toks, pattern =  "immig*")
head(kw, 10)
```

```
Keyword-in-context with 10 matches.
   [BNP, 1]                                          | IMMIGRATION |
  [BNP, 16]                  SOLVE. - At current | immigration |
```

```
 [BNP, 78]                 a halt to all further | immigration |
 [BNP, 85]        the deportation of all illegal | immigrants  |
[BNP, 169]          Britain, regardless of their | immigration |
[BNP, 197] admission that they orchestrated mass | immigration |
[BNP, 272]            grave peril, threatened by | immigration |
[BNP, 374]                  ), legal Third World | immigrants  |
[BNP, 531]        to second and third generation |  immigrant  |
[BNP, 661]                      are added in, the |  immigrant  |

: AN UNPARALLELED CRISIS WHICH
and birth rates, indigenous
, the deportation of all
, a halt to the
status. - The BNP
to change forcibly Britain's demographics
and multiculturalism. In the
made up 14.7 percent (
mothers. Figures released by
birth rate is estimated to
```

> **Note**
>
> 1. If you want to find multi-word expressions, separate words by white space and wrap the character vector by `phrase()`, as follows:
>
>    ```
>    kw_asylum <- kwic(toks, pattern = phrase("asylum seeker*"))
>    ```
>
> 2. Texts do not always appear nicely in your R console, so you can use `View()` to see the keywords-in-context in an interactive HTML table.

You can remove tokens that you are not interested in using `tokens_select()`. Usually we remove function words (grammatical words) that have little or no substantive meaning in pre-processing. `stopwords()` returns a pre-defined list of function words.

```
toks_nostop <- tokens_select(
  toks,
  pattern = stopwords("en"),
  selection = "remove"  # keep or remove
)
print(toks_nostop)
```

```
Tokens consisting of 9 documents and 1 docvar.
```

```
BNP :
 [1] "IMMIGRATION"  ":"            "UNPARALLELED" "CRISIS"       "BNP"
 [6] "CAN"          "SOLVE"        "."            "-"            "current"
[11] "immigration"  "birth"
[ ... and 2,109 more ]

Coalition :
 [1] "IMMIGRATION"  "."            "Government"   "believes"     "immigration"
 [6] "enriched"     "culture"      "strengthened" "economy"      ","
[11] "must"         "controlled"
[ ... and 146 more ]

Conservative :
 [1] "Attract"     "brightest"   "best"        "country"     "."
 [6] "Immigration" "enriched"    "nation"      "years"       "want"
[11] "attract"     "brightest"
[ ... and 277 more ]

Greens :
 [1] "Immigration" "."           "Migration"   "fact"        "life"
 [6] "."           "People"      "always"      "moved"       "one"
[11] "country"     "another"
[ ... and 377 more ]

Labour :
 [1] "Crime"            "immigration"      "challenge"        "Britain"
 [5] "control"          "immigration"      "new"              "Australian-style"
 [9] "points-based"     "system"           "-"                "unlike"
[ ... and 391 more ]

LibDem :
 [1] "firm"        "fair"        "immigration" "system"      "Britain"
 [6] "always"      "open"        ","           "welcoming"   "country"
[11] ","           "thousands"
[ ... and 285 more ]

[ reached max_ndoc ... 3 more documents ]
```

> **Note**
>
> The `stopwords()` function returns character vectors of stopwords for different languages, using the ISO-639-1 language codes. For **Malay**, use `stopwords("ms", source = "stopwords-iso")`. For Bruneian specific context, you may need to amend the stopwords yourselves.

You can generate n-grams in any lengths from a tokens using `tokens_ngrams()`. N-grams are a contiguous sequence of n tokens from already tokenized text objects. So for example, in the phrase "natural language processing":

- Unigram (1-gram): "natural", "language", "processing"
- Bigram (2-gram): "natural language", "language processing"
- Trigram (3-gram): "natural language processing"

```
# tokens_ngrams() also supports skip to generate skip-grams.
toks_ngram <- tokens_ngrams(toks_nopunct, n = 3, skip = 0)
head(toks_ngram[[1]], 20)  # the first political party's trigrams
```

```
 [1] "IMMIGRATION_AN_UNPARALLELED" "AN_UNPARALLELED_CRISIS"
 [3] "UNPARALLELED_CRISIS_WHICH"   "CRISIS_WHICH_ONLY"
 [5] "WHICH_ONLY_THE"              "ONLY_THE_BNP"
 [7] "THE_BNP_CAN"                 "BNP_CAN_SOLVE"
 [9] "CAN_SOLVE_At"                "SOLVE_At_current"
[11] "At_current_immigration"      "current_immigration_and"
[13] "immigration_and_birth"       "and_birth_rates"
[15] "birth_rates_indigenous"      "rates_indigenous_British"
[17] "indigenous_British_people"   "British_people_are"
[19] "people_are_set"              "are_set_to"
```

3.3.3 Document feature matrix

`dfm()` constructs a document-feature matrix (DFM) from a tokens object.

```
toks_inaug <- tokens(data_corpus_inaugural, remove_punct = TRUE)
dfmat_inaug <- dfm(toks_inaug)
print(dfmat_inaug)
```

```
Document-feature matrix of: 59 documents, 9,419 features (91.89% sparse) and 4 docvars.
                 features
docs              fellow-citizens  of the senate and house representatives
```

```
  1789-Washington               1  71 116      1  48     2               2
  1793-Washington               0  11  13      0   2     0               0
  1797-Adams                    3 140 163      1 130     0               2
  1801-Jefferson                2 104 130      0  81     0               0
  1805-Jefferson                0 101 143      0  93     0               0
  1809-Madison                  1  69 104      0  43     0               0
                 features
docs              among vicissitudes incident
  1789-Washington     1             1        1
  1793-Washington     0             0        0
  1797-Adams          4             0        0
  1801-Jefferson      1             0        0
  1805-Jefferson      7             0        0
  1809-Madison        0             0        0
[ reached max_ndoc ... 53 more documents, reached max_nfeat ... 9,409 more features ]
```

Some useful functions to operate on DFMs are:

1. `ndoc()`: returns the number of documents
2. `nfeat()`: returns the number of features
3. `docnames()`: returns the document names
4. `featnames()`: returns the feature (column) names
5. `topfeatures()`: returns the most frequent features
6. `docvars()`: returns the document-level variables

DFMs sometimes behaves like normal matrices too, so you can use `rowSums()` and `colSums()` to calculate marginals.

Most commonly perhaps, is you want to select some columns (i.e. features) from the DFM that satisfy a pattern. For instance,

```
dfm_select(dfmat_inaug, pattern = "freedom")
```

```
Document-feature matrix of: 59 documents, 1 feature (38.98% sparse) and 4 docvars.
                 features
docs              freedom
  1789-Washington       0
  1793-Washington       0
  1797-Adams            0
  1801-Jefferson        4
  1805-Jefferson        2
  1809-Madison          1
[ reached max_ndoc ... 53 more documents ]
```

```
dfm_keep(dfmat_inaug, min_nchar = 5)
```

```
Document-feature matrix of: 59 documents, 8,560 features (93.04% sparse) and 4 docvars.
                 features
docs              fellow-citizens senate house representatives among
  1789-Washington               1      1     2               2     1
  1793-Washington               0      0     0               0     0
  1797-Adams                    3      1     0               2     4
  1801-Jefferson                2      0     0               0     1
  1805-Jefferson                0      0     0               0     7
  1809-Madison                  1      0     0               0     0
                 features
docs              vicissitudes incident event could filled
  1789-Washington            1        1     2     3      1
  1793-Washington            0        0     0     0      0
  1797-Adams                 0        0     0     1      0
  1801-Jefferson             0        0     0     0      0
  1805-Jefferson             0        0     0     2      0
  1809-Madison               0        0     0     1      1
[ reached max_ndoc ... 53 more documents, reached max_nfeat ... 8,550 more features ]
```

There is also `dfm_trim()` to remove features that are too frequent or too rare, based on frequencies.

```
# Trim DFM containing features that occur less than 10 times in the corpus
dfm_trim(dfmat_inaug, min_termfreq = 10)
```

```
Document-feature matrix of: 59 documents, 1,524 features (68.93% sparse) and 4 docvars.
                 features
docs              fellow-citizens  of the senate and house representatives
  1789-Washington               1  71 116      1  48     2               2
  1793-Washington               0  11  13      0   2     0               0
  1797-Adams                    3 140 163      1 130     0               2
  1801-Jefferson                2 104 130      0  81     0               0
  1805-Jefferson                0 101 143      0  93     0               0
  1809-Madison                  1  69 104      0  43     0               0
                 features
docs              among to life
  1789-Washington     1 48    1
  1793-Washington     0  5    0
  1797-Adams          4 72    2
```

```
  1801-Jefferson      1 61    1
  1805-Jefferson      7 83    2
  1809-Madison        0 61    1
[ reached max_ndoc ... 53 more documents, reached max_nfeat ... 1,514 more features ]
```

```
# Trim DFM containing features that occur in more than 10% of the documents
dfm_trim(dfmat_inaug, max_docfreq = 0.1, docfreq_type = "prop")
```

```
Document-feature matrix of: 59 documents, 7,388 features (96.87% sparse) and 4 docvars.
                 features
docs              vicissitudes filled anxieties notification transmitted 14th
  1789-Washington            1      1         1            1           1    1
  1793-Washington            0      0         0            0           0    0
  1797-Adams                 0      0         0            0           0    0
  1801-Jefferson             0      0         0            0           0    0
  1805-Jefferson             0      0         0            0           0    0
  1809-Madison               0      1         0            0           0    0
                 features
docs              month summoned veneration fondest
  1789-Washington     1        1          1       1
  1793-Washington     0        0          0       0
  1797-Adams          0        0          2       0
  1801-Jefferson      0        0          0       0
  1805-Jefferson      0        0          0       0
  1809-Madison        0        0          0       0
[ reached max_ndoc ... 53 more documents, reached max_nfeat ... 7,378 more features ]
```

3.4 Statistical analysis

Note: If you have not installed `{quanteda.corpora}`, do so by running

```
remotes::install_github("quanteda/quanteda.corpora")
```

3.4.1 Simple frequency analysis

Unlike `topfeatures()`, `textstat_frequency()` shows both term and document frequencies. You can also use the function to find the most frequent features within groups. Using the `download()` function from `{quanteda.corpora}`, you can retrieve a text corpus of tweets.

```
corp_tweets <- quanteda.corpora::download(url = "https://www.dropbox.com/s/846skn1i5elbnd2/da
```

We can analyse the most frequent hashtags by applying `tokens_keep(pattern = "#*")` before creating a DFM.

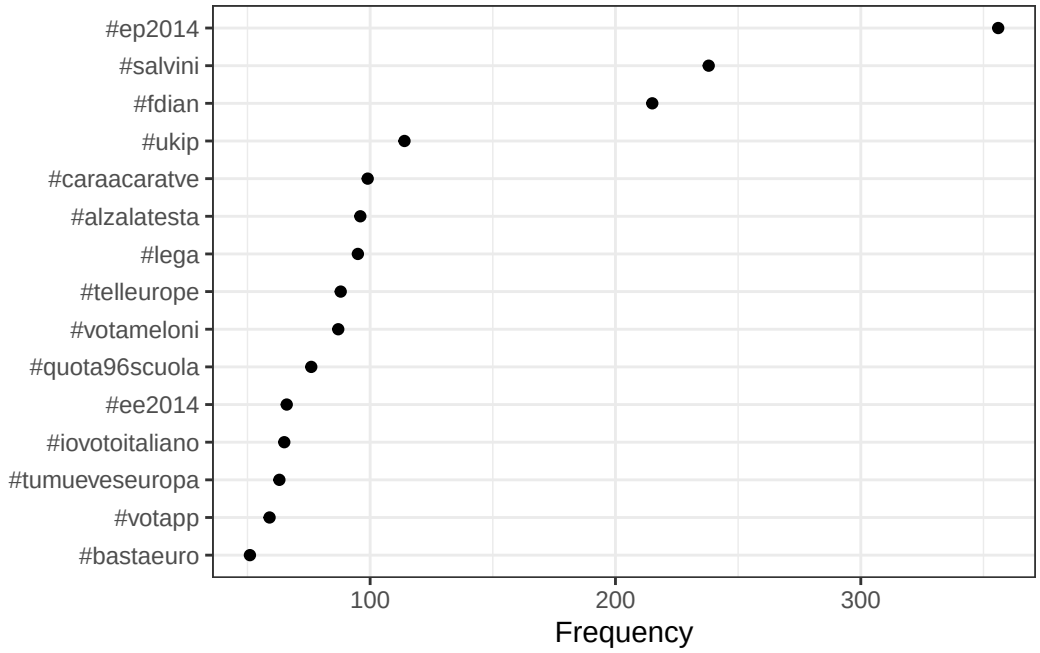
```
toks_tweets <-
  tokens(corp_tweets, remove_punct = TRUE) |>
  tokens_keep(pattern = "#*")
dfmat_tweets <- dfm(toks_tweets)

tstat_freq <- textstat_frequency(dfmat_tweets, n = 5, groups = lang)
head(tstat_freq, 20)
```

```
             feature frequency rank docfreq     group
1           #twitter         1    1       1    Basque
2     #canviemeuropa         1    1       1    Basque
3              #prest         1    1       1    Basque
4           #psifizo         1    1       1    Basque
5     #ekloges2014gr         1    1       1    Basque
6            #ep2014         1    1       1 Bulgarian
7         #yourvoice         1    1       1 Bulgarian
8      #eudebate2014         1    1       1 Bulgarian
9              #             1    1       1 Bulgarian
10           #ep2014         1    1       1  Croatian
11 #savedonbaspeople         1    1       1  Croatian
12   #vitoriagasteiz         1    1       1  Croatian
13           #ep14dk        32    1      32    Danish
14            #dkpol        18    2      18    Danish
15            #eupol         7    3       7    Danish
16        #vindtilep         7    3       7    Danish
17    #patentdomstol         4    5       4    Danish
18           #ep2014        35    1      35     Dutch
19              #vvd        11    2      11     Dutch
20               #eu         9    3       7     Dutch
```

You can also plot the Twitter hashtag frequencies easily using `ggplot()`.

```
dfmat_tweets |>
  textstat_frequency(n = 15) |>
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
```

```
  coord_flip() +
  labs(x = NULL, y = "Frequency") +
  theme_bw()
```

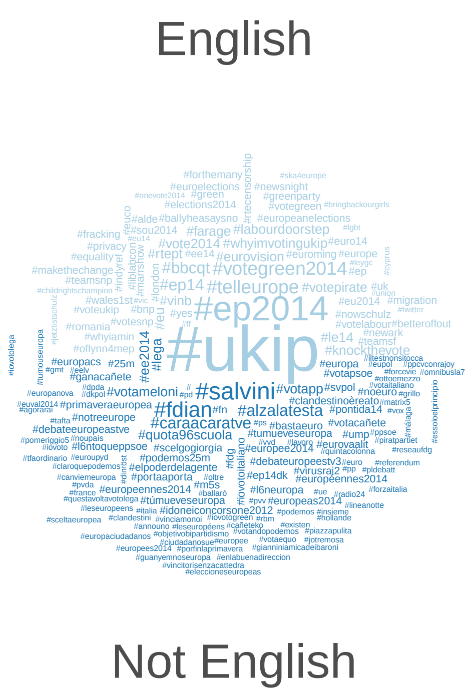


Alternatively, you can create a word cloud of the 100 most common hashtags.

```
set.seed(132)
textplot_wordcloud(dfmat_tweets, max_words = 100)
```

Finally, it is possible to compare different groups within one Wordcloud. We must first create a dummy variable that indicates whether a tweet was posted in English or a different language. Afterwards, we can compare the most frequent hashtags of English and non-English tweets.

```r
# create document-level variable indicating whether tweet was in English or
# other language
corp_tweets$dummy_english <-
  factor(ifelse(corp_tweets$lang == "English", "English", "Not English"))

# tokenize texts
toks_tweets <- tokens(corp_tweets)

# create a grouped dfm and compare groups
dfmat_corp_language <-
  dfm(toks_tweets) |>
  dfm_keep(pattern = "#*") |>
  dfm_group(groups = dummy_english)

# create wordcloud
set.seed(132) # set seed for reproducibility
textplot_wordcloud(dfmat_corp_language, comparison = TRUE, max_words = 200)
```



3.4.2 Lexical diversity

Lexical diversity is a measure of how varied the vocabulary in a text or speech is. It indicates the richness of language use by comparing the number of unique words (types) to the total number of words (tokens) in the text. It is useful, for instance, for analysing speakers' or writers' linguistic skills, or the complexity of ideas expressed in documents.

A common metric for lexical diversity is the Type-Token Ratio (TTR), calculated as:

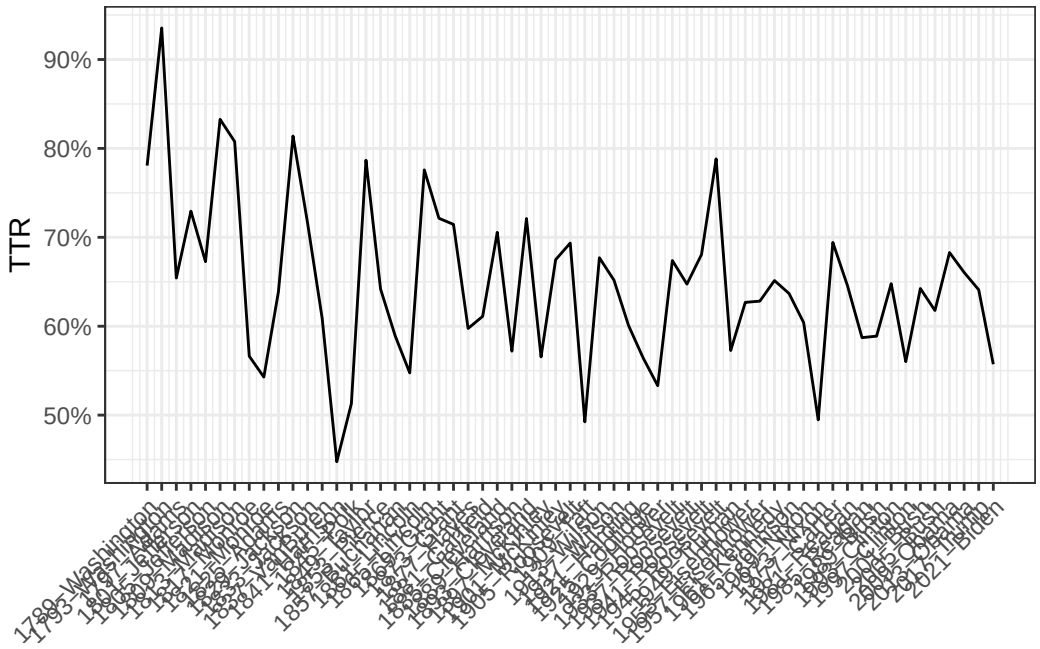
$$TTR = \frac{N_{\text{types}}}{N_{\text{tokens}}}$$

```
toks_inaug <- tokens(data_corpus_inaugural)
dfmat_inaug <-
  dfm(toks_inaug) |>
  dfm_remove(pattern = stopwords("en"))  # similar to dfm_select()

tstat_lexdiv <- textstat_lexdiv(dfmat_inaug)
tail(tstat_lexdiv, 5)
```

```
     document       TTR
55  2005-Bush 0.6176753
56 2009-Obama 0.6828645
57 2013-Obama 0.6605238
58 2017-Trump 0.6409537
59 2021-Biden 0.5572316
```

We can prepare a plot using `ggplot()` as follows:

```
plot_df <-
  tstat_lexdiv |>
  mutate(id = row_number())

ggplot(plot_df, aes(id, TTR)) +
  geom_line() +
  scale_x_continuous(
    breaks = plot_df$id,
    labels = plot_df$document,
    name = NULL
  ) +
  scale_y_continuous(labels = scales::percent) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



3.4.3 Document/Feature similarity

Document/feature similarity is a measure of how alike two documents or sets of features are based on their content. It quantifies the degree to which documents share similar terms, topics, or characteristics.

`textstat_dist()` calculates similarities of documents or features for various measures. The output is compatible with R's `dist()`, so hierarchical clustering can be performed without any transformation.

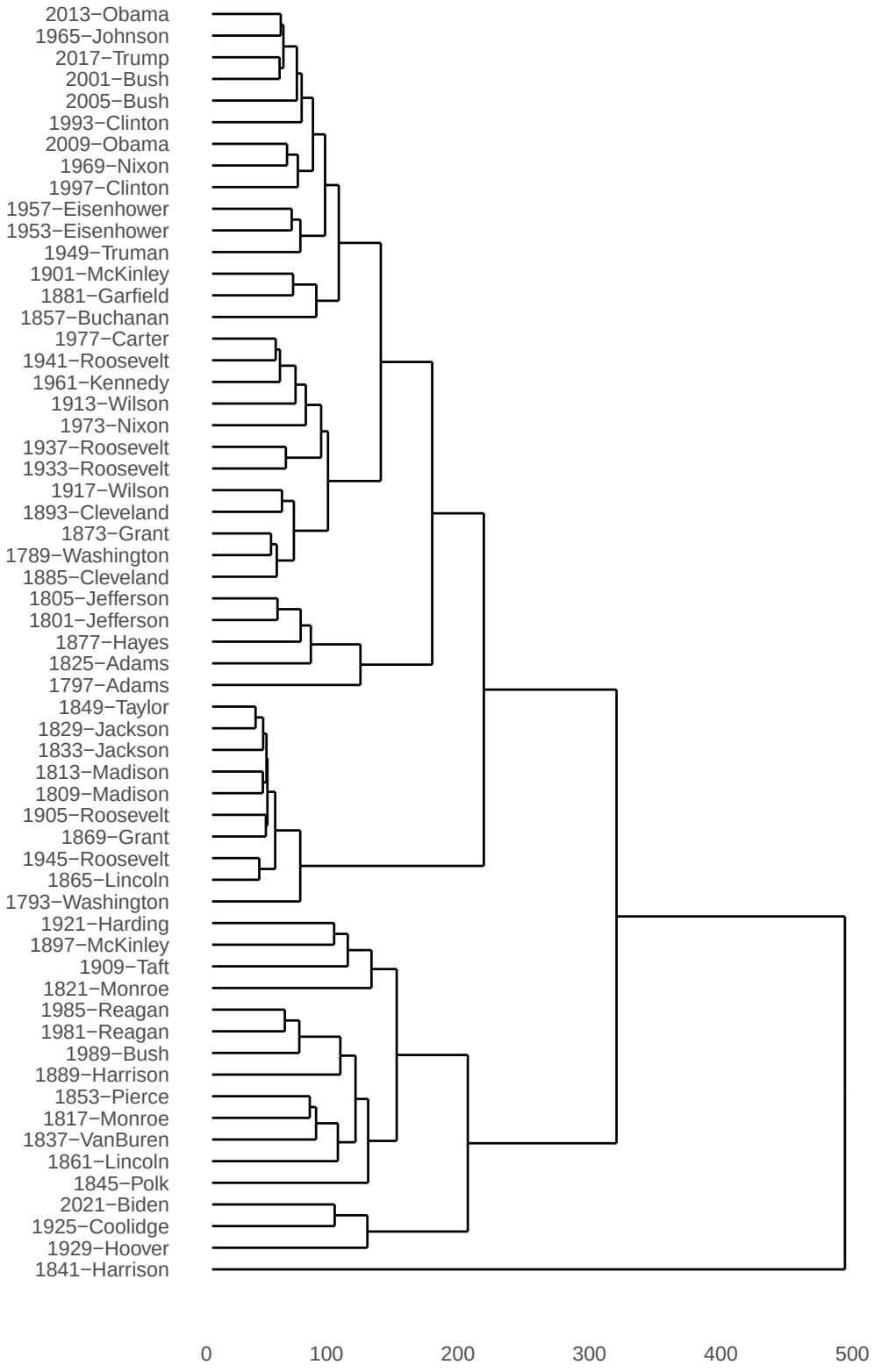
```
toks_inaug <- tokens(data_corpus_inaugural)
dfmat_inaug <-
  dfm(toks_inaug) |>
  dfm_remove(pattern = stopwords("en"))  # similar to dfm_select()

# Calculate document similarity
dist_mat <- textstat_dist(dfmat_inaug)  # using Euclidean distance
dist_mat[1:3, 1:3]
```

```
3 x 3 Matrix of class "dspMatrix"
                 1789-Washington 1793-Washington 1797-Adams
1789-Washington          0.00000         76.13803   141.4072
1793-Washington         76.13803          0.00000   206.6954
```

```
1797-Adams              141.40721       206.69543      0.0000
```

To plot this using `ggplot()`, we rely on the `{ggdendro}` package.

```r
clust <- hclust(as.dist(dist_mat))  # hierarchical clustering

library(ggdendro)
dendr <- dendro_data(clust)
ggdendrogram(dendr, rotate = TRUE)
```


2013−Obama
1965−Johnson
2017−Trump
2001−Bush
2005−Bush
1993−Clinton
2009−Obama
1969−Nixon
1997−Clinton
1957−Eisenhower
1953−Eisenhower
1949−Truman
1901−McKinley
1881−Garfield
1857−Buchanan
1977−Carter
1941−Roosevelt
1961−Kennedy
1913−Wilson
1973−Nixon
1937−Roosevelt
1933−Roosevelt
1917−Wilson
1893−Cleveland
1873−Grant
1789−Washington
1885−Cleveland
1805−Jefferson
1801−Jefferson
1877−Hayes
1825−Adams
1797−Adams
1849−Taylor
1829−Jackson
1833−Jackson
1813−Madison
1809−Madison
1905−Roosevelt
1869−Grant
1945−Roosevelt
1865−Lincoln
1793−Washington
1921−Harding
1897−McKinley
1909−Taft
1821−Monroe
1985−Reagan
1981−Reagan
1989−Bush
1889−Harrison
1853−Pierce
1817−Monroe
1837−VanBuren
1861−Lincoln
1845−Polk
2021−Biden
1925−Coolidge
1929−Hoover
1841−Harrison
0
100
200
300
400
500

3.4.4 Feature co-occurence matrix

A feature co-occurrence matrix (FCM) is a square matrix that counts the number of times two features co-occur in the same context, such as within the same document, sentence, or window of text. This is a special object in `{quanteda}`, but behaves similarly to a DFM. As an example, consider the following:

```
tibble(
  doc_id = 1:2,
  text = c("I love Mathematics.", "Mathematics is awesome.")
) |>
  corpus() |>
  tokens(remove_punct = TRUE) |>
  fcm(context = "document")  # by default
```

```
Feature co-occurrence matrix of: 5 by 5 features.
             features
features      I love Mathematics is awesome
  I           0    1           1  0       0
  love        0    0           1  0       0
  Mathematics 0    0           0  1       1
  is          0    0           0  0       1
  awesome     0    0           0  0       0
```

Let's download the `data_corpus_guardian` corpus from the `{quanteda.corpora}` package.

```
corp_news <- quanteda.corpora::download("data_corpus_guardian")
```

When a corpus is large, you have to select features of a DFM before constructing a FCM. In the example below, we clean up as follows:

1. Remove all stopwords and punctuation characters.
2. Remove certain patterns that usually describe the publication time and date of articles.
3. Keep only terms that occur at least 100 times in the document-feature matrix.

```
toks_news <- tokens(corp_news, remove_punct = TRUE)
dfmat_news <-
  dfm(toks_news) |>
  dfm_remove(pattern = c(stopwords("en"), "*-time", "updated-*", "gmt", "bst", "|")) |>
  dfm_trim(min_termfreq = 100)

topfeatures(dfmat_news)
```

```
      said     people        one        new       also         us        can
     28413      11169       9884       8024       7901       7091       6972
government       year       last
      6821       6570       6335
```

```
nfeat(dfmat_news)
```

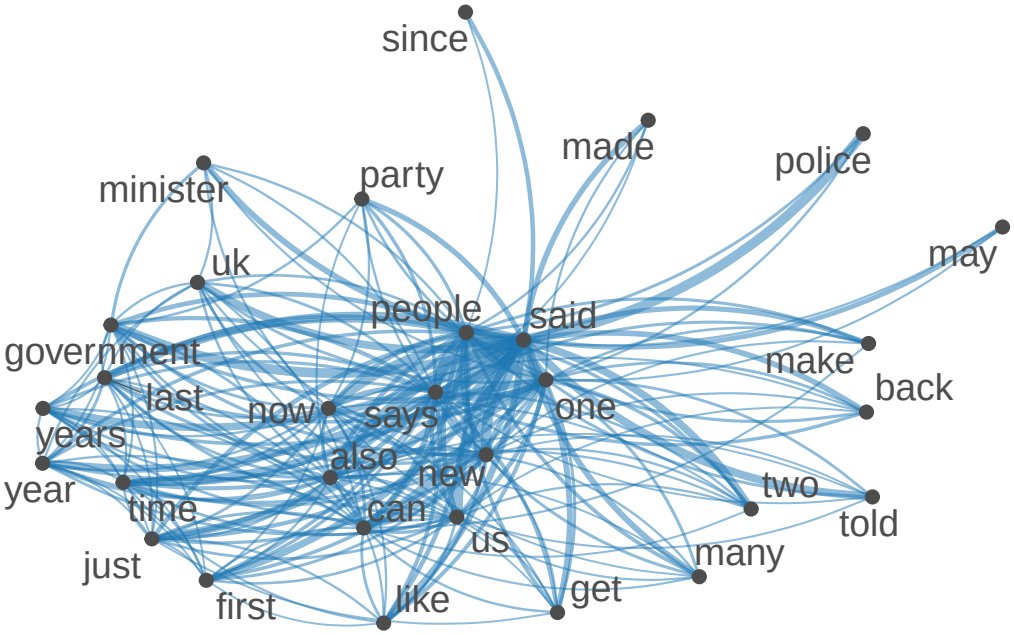
```
[1] 4211
```

To construct an FCM from a DFM (or a tokens object), use `fcm()`. You can visualise the FCM using a `textplot_network()` graph as follows:

```
fcmat_news <- fcm(dfmat_news, context = "document")
feat <- names(topfeatures(dfmat_news, 30))  # Top 30 features
fcmat_news_select <- fcm_select(fcmat_news, pattern = feat, selection = "keep")
dim(fcmat_news_select)
```

```
[1] 30 30
```

```
set.seed(123)
quanteda.textplots::textplot_network(fcmat_news_select)
```



3.5 Scaling and classification

In this section we apply mainly unsupervised learning models to textual data. Scaling and classification aim to uncover hidden structures, relationships, and patterns within textual data by placing texts or words on latent scales (scaling) and grouping them into meaningful categories or themes (classification). This process transforms complex, high-dimensional text into more interpretable and actionable insights.

3.5.1 Wordfish

Wordfish is a Poisson scaling model of one-dimensional document positions (Slapin and Proksch 2008). This model is used primarily for scaling political texts to position documents (like speeches or manifestos) on a latent dimension, often reflecting ideological or policy positions. The main objective is to identify the relative positioning of texts on a scale (e.g., left-right political spectrum) based on word frequencies.

Let y_{ij} be the count of word j in document i. Then assume

$$y_{ij} \sim \text{Poi}(\lambda_{ij}) \tag{3.1}$$

$$\log(\lambda_{ij}) = \psi_j + \beta_j \theta_i \tag{3.2}$$

In this example, we will show how to apply Wordfish to the Irish budget speeches from 2010. First, we will create a document-feature matrix. Afterwards, we will run Wordfish.

```
toks_irish <- tokens(data_corpus_irishbudget2010, remove_punct = TRUE)
dfmat_irish <- dfm(toks_irish)

# Run Wordfish model
tmod_wf <- textmodel_wordfish(dfmat_irish, dir = c(6, 5))
summary(tmod_wf)
```

```
Call:
textmodel_wordfish.dfm(x = dfmat_irish, dir = c(6, 5))

Estimated Document Positions:
                            theta      se
Lenihan, Brian (FF)       1.79403 0.02007
Bruton, Richard (FG)     -0.62160 0.02823
Burton, Joan (LAB)       -1.13503 0.01568
```

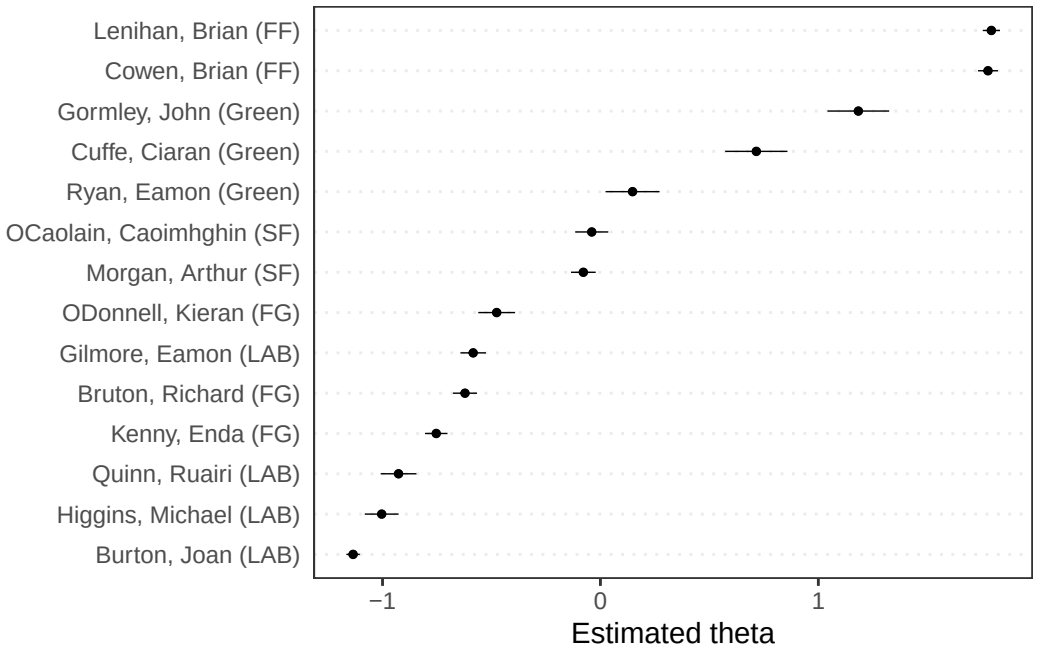
```
Morgan, Arthur (SF)       -0.07841 0.02896
Cowen, Brian (FF)          1.77846 0.02330
Kenny, Enda (FG)          -0.75350 0.02635
ODonnell, Kieran (FG)     -0.47615 0.04309
Gilmore, Eamon (LAB)      -0.58406 0.02994
Higgins, Michael (LAB)    -1.00383 0.03964
Quinn, Ruairi (LAB)       -0.92648 0.04183
Gormley, John (Green)      1.18361 0.07224
Ryan, Eamon (Green)        0.14738 0.06321
Cuffe, Ciaran (Green)      0.71541 0.07291
OCaolain, Caoimhghin (SF) -0.03982 0.03878

Estimated Feature Scores:
        when      i presented    the supplementary  budget     to   this  house
beta -0.1594 0.3177    0.3603 0.1933         1.077 0.03537 0.3077 0.2473 0.1399
psi   1.6241 2.7239   -1.7958 5.3308        -1.134 2.70993 4.5190 3.4603 1.0396
       last   april    said     we   could   work    our   way through  period
beta 0.2419 -0.1565 -0.8339 0.4156 -0.6138 0.5221 0.6892 0.275  0.6115  0.4985
psi  0.9853 -0.5725 -0.4514 3.5125  1.0858 1.1151 2.5278 1.419  1.1603 -0.1779
        of severe economic distress   today   can  report    that
beta 0.2777  1.229   0.4237    1.799 0.09141 0.304  0.6196 0.01506
psi  4.4656 -2.013   1.5714   -4.456 0.83875 1.564 -0.2466 3.83785
     notwithstanding difficulties   past
beta           1.799        1.175 0.4746
psi           -4.456       -1.357 0.9321
```

The R output shows the results of the Wordfish model applied to Irish political texts, estimating the ideological positions of various politicians. Each politician is assigned a "theta" value, representing their placement on a latent scale; positive values indicate one end of the spectrum, while negative values indicate the opposite.

For example, Brian Lenihan (FF) has a high positive theta, suggesting a strong position on one side, while Joan Burton (LAB) has a negative theta, placing her on the other side. The model also provides feature scores for words (beta values), indicating their importance in distinguishing between these positions. Words with higher absolute beta values, such as "supplementary," are key in differentiating the ideological content of the texts, while psi values reflect word frequency variance, contributing to the model's differentiation of document positions.

We can plot the results of a fitted scaling model using `textplot_scale1d()`.

```
textplot_scale1d(tmod_wf)
```

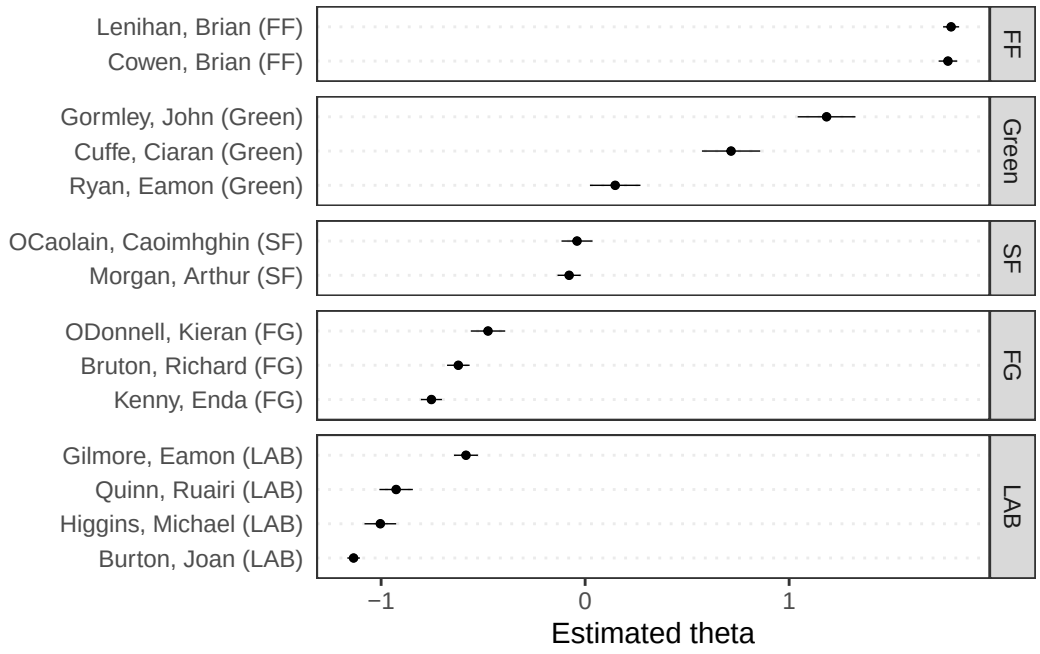


Note

The value of 0 for theta in the Wordfish model is not a true zero in an absolute sense. Instead, it serves as a relative reference point on the latent scale. In Wordfish, theta values are relative, meaning they indicate positions along a spectrum where the direction (positive or negative) is determined by the model's scaling based on the data and specified parameters.

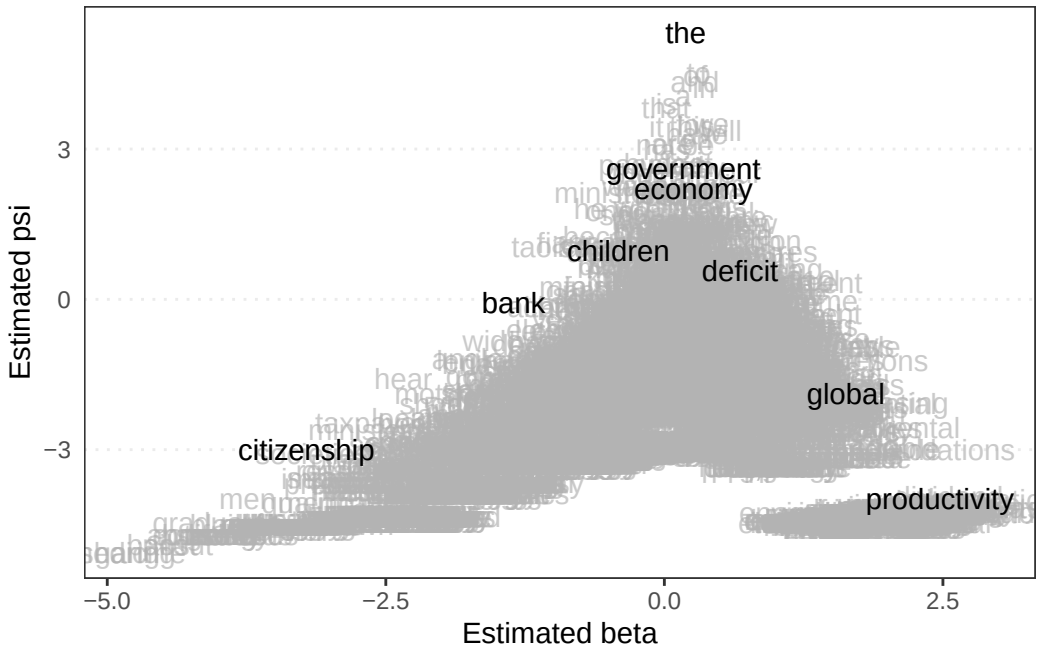
The function also allows you to plot scores by a grouping variable, in this case the party affiliation of the speakers.

```
textplot_scale1d(tmod_wf, groups = dfmat_irish$party)
```



Finally, we can plot the estimated word positions and highlight certain features.

```
textplot_scale1d(
  tmod_wf,
  margin = "features",
  highlighted = c("government", "global", "children",
                  "bank", "economy", "the", "citizenship",
                  "productivity", "deficit")
)
```



Beta (x-axis) Reflects how strongly a word is associated with the latent dimension (e.g., ideological position). Words with high absolute beta values are more influential in distinguishing between different positions; positive beta values indicate words more associated with one end of the scale, while negative values indicate the opposite.

Psi (y-axis) Represents the variance in word frequency. Higher psi values suggest that the word occurs with varying frequency across documents, while lower values indicate more consistent usage.

Therefore, words in the upper right (high beta, high psi) are influential and variably used, indicating key terms that may strongly differentiate between document positions. Words in the lower left (low beta, low psi) are less influential and consistently used, likely serving as common or neutral terms.

The plot also helps identify which words are driving the distinctions in the latent scale and how their usage varies across documents.

3.5.2 Topic models

Topic models are statistical models used to identify the underlying themes or topics within a large collection of documents. They analyze word co-occurrences across documents to group words into topics, where each topic is a distribution over words, and each document is a mixture of these topics.

A common topic model is Latent Dirichlet Allocation (LDA), which assumes that each document contains multiple topics in varying proportions. Topic models help uncover hidden semantic structures in text, making them useful for organizing, summarizing, and exploring large text datasets. In R, we use the `{seededlda}` package for LDA.

```
# install.packages("seededlda")
library(seededlda)
```

Back to the Guardian data, `corp_news`. We will select only news articles published in 2016 using `corpus_subset()` function and the `year()` function from the `{lubridate}` package

```
corp_news_2016 <- corpus_subset(corp_news, year(date) == 2016)
ndoc(corp_news_2016)
```

```
[1] 1959
```

Further, after removal of function words and punctuation in `dfm()`, we will only keep the top 20% of the most frequent features (`min_termfreq = 0.8`) that appear in less than 10% of all documents (`max_docfreq = 0.1`) using `dfm_trim()` to focus on common but distinguishing features.

```
# Create tokens
toks_news <-
  tokens(
    corp_news_2016,
    remove_punct = TRUE,
    remove_numbers = TRUE,
    remove_symbol = TRUE
  ) |>
  tokens_remove(
    pattern = c(stopwords("en"), "*-time", "updated-*", "gmt", "bst")
  )

# Create DFM
dfmat_news <-
  dfm(toks_news) %>%
  dfm_trim(
    min_termfreq = 0.8,
    termfreq_type = "quantile",
    max_docfreq = 0.1,
    docfreq_type = "prop"
  )
```

The LDA is fitted using the code below. Note that `k = 10` specifies the number of topics to be discovered. This is an important parameter and you should try a variety of values and validate the outputs of your topic models thoroughly.

```
# Takes a while to fit!
tmod_lda <- seededlda::textmodel_lda(dfmat_news, k = 10)
```

You can extract the most important terms for each topic from the model using `terms()`. Each column (`topic1`, `topic2`, etc.) lists words that frequently co-occur in the dataset, suggesting a common theme within each topic.

```
terms(tmod_lda, 10)
```

```
      topic1     topic2       topic3         topic4       topic5    topic6
 [1,] "syria"    "labor"      "johnson"      "funding"    "son"     "officers"
 [2,] "refugees" "australia"  "brussels"     "housing"    "church"  "violence"
 [3,] "isis"     "australian" "talks"        "nhs"        "black"   "doctors"
 [4,] "military" "corbyn"     "boris"        "income"     "love"    "prison"
 [5,] "syrian"   "turnbull"   "benefits"     "education"  "father"  "victims"
 [6,] "un"       "budget"     "summit"       "scheme"     "felt"    "sexual"
 [7,] "islamic"  "leadership" "negotiations" "fund"       "parents" "abuse"
 [8,] "forces"   "shadow"     "ireland"      "green"      "story"   "hospital"
 [9,] "turkey"   "senate"     "migrants"     "homes"      "visit"   "criminal"
[10,] "muslim"   "coalition"  "greece"       "businesses" "read"    "crime"
      topic7      topic8    topic9          topic10
 [1,] "oil"       "clinton" "climate"       "sales"
 [2,] "markets"   "sanders" "water"         "apple"
 [3,] "prices"    "cruz"    "energy"        "customers"
 [4,] "banks"     "hillary" "food"          "users"
 [5,] "investors" "obama"   "gas"           "google"
 [6,] "shares"    "trump's" "drug"          "technology"
 [7,] "trading"   "bernie"  "drugs"         "games"
 [8,] "china"     "ted"     "environmental" "game"
 [9,] "rates"     "rubio"   "air"           "iphone"
[10,] "quarter"   "senator" "emissions"     "app"
```

As an example, Topic 1 (“syria”, “refugees”, “isis”), is likely related to international conflicts, specifically around Syria and refugee crises. Topic 4 (“funding”, “housing”, “nhs”) is likely related to public services and social welfare issues, such as healthcare and housing. Each topic provides a distinct theme, derived from the words that frequently appear together in the corpus, helping to summarize and understand the main themes in the text.

You can then obtain the most likely topics using topics() and save them as a document-level variable.

```
# assign topic as a new document-level variable
dfmat_news$topic <- topics(tmod_lda)

# cross-table of the topic frequency
table(dfmat_news$topic)
```

```
 topic1  topic2  topic3  topic4  topic5  topic6  topic7  topic8  topic9 topic10
    203     222      85     211     249     236     180     186     196     184
```

In the seeded LDA, you can pre-define topics in LDA using a dictionary of "seeded" words. For more information, see the `{seededlda}` package documentation.

3.5.3 Latent semantic scaling

Latent Semantic Scaling (LSS) is a method used to place words or documents on a latent scale that represents an underlying dimension, such as sentiment, ideology, or any other semantic axis. The key idea is to use the co-occurrence patterns of words across documents to identify and position items along this hidden dimension.

LSS is performed using the `{LSX}` package. In this example, we will apply LSS to the corpus of Sputnik articles about Ukraine. First, we prepare the data set.

```
# Read the RDS file directly from the URL
corp <- readRDS(url("https://www.dropbox.com/s/abme18nlrwxgmz8/data_corpus_sputnik2022.rds?dl

toks <-
  corp |>
  corpus_reshape("sentences") |>  # this is a must!
  tokens(
    remove_punct = TRUE,
    remove_symbols = TRUE,
    remove_numbers = TRUE,
    remove_url = TRUE
  )

dfmt <-
  dfm(toks) |>
  dfm_remove(pattern = stopwords("en"))
```

Now to run an LSS model, run the following command:

```
lss <- textmodel_lss(
  dfmt,
  seeds = as.seedwords(data_dictionary_sentiment),
  k = 300,
  # cache = TRUE,
  include_data = TRUE,
  group_data = TRUE
)
```

Taking the DFM and the seed words as the only inputs, `textmodel_lss()` computes the *polarity* scores of all the words in the corpus based on their semantic similarity to the seed words. You usually do not need to change the value of `k` (300 by default).

Let's look at the output of the LSS model:

```
summary(lss)
```

```
Call:
textmodel_lss(x = dfmt, seeds = as.seedwords(data_dictionary_sentiment),
    k = 300, include_data = TRUE, group_data = TRUE)

Seeds:
       good        nice   excellent    positive   fortunate     correct
          1           1           1           1           1           1
   superior         bad       nasty        poor    negative unfortunate
          1          -1          -1          -1          -1          -1
      wrong    inferior
         -1          -1

Beta:
(showing first 30 elements)
          excellent            positive              gander                good
             0.2102              0.1918              0.1883              0.1829
         diplomatic           staffer's              ex-kgb             correct
             0.1802              0.1773              0.1771              0.1749
              mend  inter-parliamntary         workmanship      russo-american
             0.1743              0.1719              0.1715              0.1713
             nicety           china-u.s               soufi good-neighborliness
             0.1712              0.1674              0.1664              0.1644
```

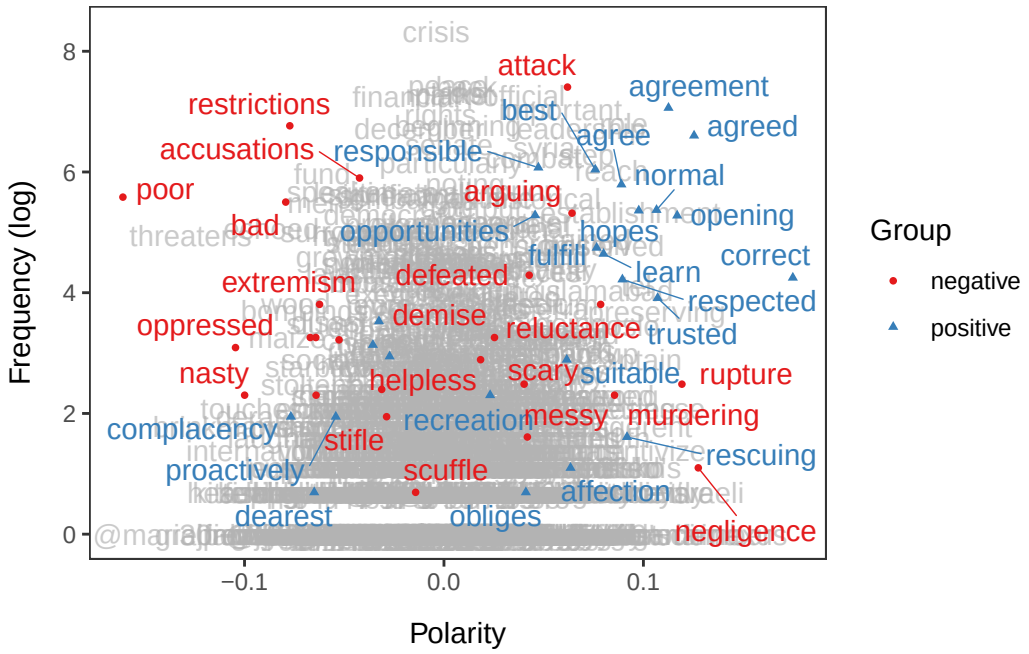
```
      china-canada             fidgety           relations           downtrend
            0.1616              0.1611              0.1584              0.1578
           cordial       canada-china            superior      understanding
            0.1573              0.1569              0.1569              0.1561
  good-neighbourly          brennan's              mutual         reaffirmed
            0.1550              0.1544              0.1538              0.1534
             blida         abdelkader
            0.1533              0.1533

Data Dimension:
[1]  8063 59711
```

Polarity scores in Latent Semantic Scaling (LSS) quantify how words or documents relate to a specific dimension (e.g., sentiment) based on predefined seed words. Seed words represent the extremes of this dimension (e.g., "good" vs. "bad"). LSS analyzes how other words co-occur with these seed words to assign a score.

We can visualize the polarity of words using `textplot_terms()`. If you pass a dictionary to be highlighted, words are plotted in different colors. `data_dictionary_LSD2015` is a widely-used sentiment dictionary. If `highlighted = NULL`, words are selected randomly to highlight.

```
textplot_terms(lss, highlighted = data_dictionary_LSD2015[1:2])
```

```
Warning: ggrepel: 8 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```



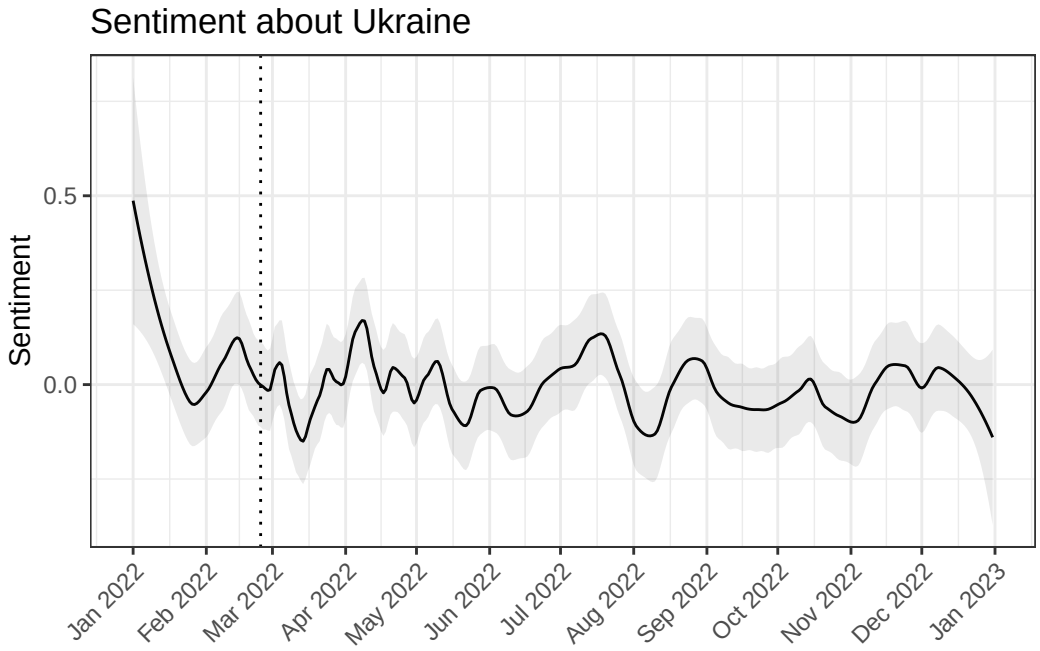
Based on the fitted model, we can now predict polarity scores of documents using `predict()`. It is best to work with the document-level data frame, which we will then add a new column for the predicted polarity scores.

```
dat <- docvars(lss$data)
dat$lss <- predict(lss)
glimpse(dat)
```

```
Rows: 8,063
Columns: 5
$ head <chr> "Biden: US Desires Diplomacy But 'Ready No Matter What Happens' I~
$ url  <chr> "https://sputniknews.com/20220131/biden-us-desires-diplomacy-but-~
$ time <dttm> 2022-02-01 03:25:22, 2022-02-01 01:58:19, 2022-02-01 01:47:56, 2~
$ date <date> 2022-01-31, 2022-01-31, 2022-01-31, 2022-01-31, 2022-01-31, 2022~
$ lss  <dbl> 0.10093989, 0.99263241, -0.76609160, 0.15991318, 0.25420329, -1.5~
```

Basically what we have is a data frame, where each row represents a single document (here, a news item from Sputnik with a timestamp). Each document also has a predicted polarity score based on the LSS model. We can visualise this easily using `ggplot()`. But first, we need to smooth the scores using `smooth_lss()` (otherwise it is too rough to interpret).

```r
dat_smooth <- smooth_lss(dat, lss_var = "lss", date_var = "date")

ggplot(dat_smooth, aes(x = date, y = fit)) +
  geom_line() +
  geom_ribbon(
    aes(ymin = fit - se.fit * 1.96,
        ymax = fit + se.fit * 1.96),
    alpha = 0.1
  ) +
  geom_vline(xintercept = as.Date("2022-02-24"), linetype = "dotted") +
  scale_x_date(date_breaks = "months", date_labels = "%b %Y", name = NULL) +
  labs(title = "Sentiment about Ukraine", x = "Date", y = "Sentiment") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



The plot shows that the sentiment of the articles about Ukraine became more negative in March but more positive in April. Zero on the Y-axis is the overall mean of the score; the dotted vertical line indicate the beginning of the war.

DEPS. 2022. "The Population and Housing Census Report (BPP) 2021: Demographic, Household and Housing Characteristics." Department of Economic Planning and Statistics, Ministry of Finance and Economy, Brunei Darussalam.

Jaafar, Salwana Md, and Rahayu Sukmaria Sukri. 2023. "Data on the Physicochemical Characteristics and Texture Classification of Soil in Bornean Tropical Heath Forests Affected

by Exotic Acacia Mangium." *Data in Brief* 51 (December). https://doi.org/10.1016/j.dib.2023.109670.

Pebesma, Edzer, and Roger Bivand. 2023. *Spatial Data Science: With Applications in R*. 1st ed. New York: Chapman and Hall/CRC. https://doi.org/10.1201/9780429459016.

Slapin, Jonathan B., and Sven-Oliver Proksch. 2008. "A Scaling Model for Estimating Time-Series Party Positions from Texts." *American Journal of Political Science* 52 (3): 705–22. https://doi.org/10.1111/j.1540-5907.2008.00338.x.