A brief introduction to regular expressions

Data Wrangling: Session 5

Kieran Healy
Statistical Horizons, October 2023

Load the packages, as always

```
library(here)      # manage file paths
library(socviz)    # data and some useful functions
```

```
library(tidyverse) # your friend and mine
library(gapminder) # gapminder data
library(stringr)
```

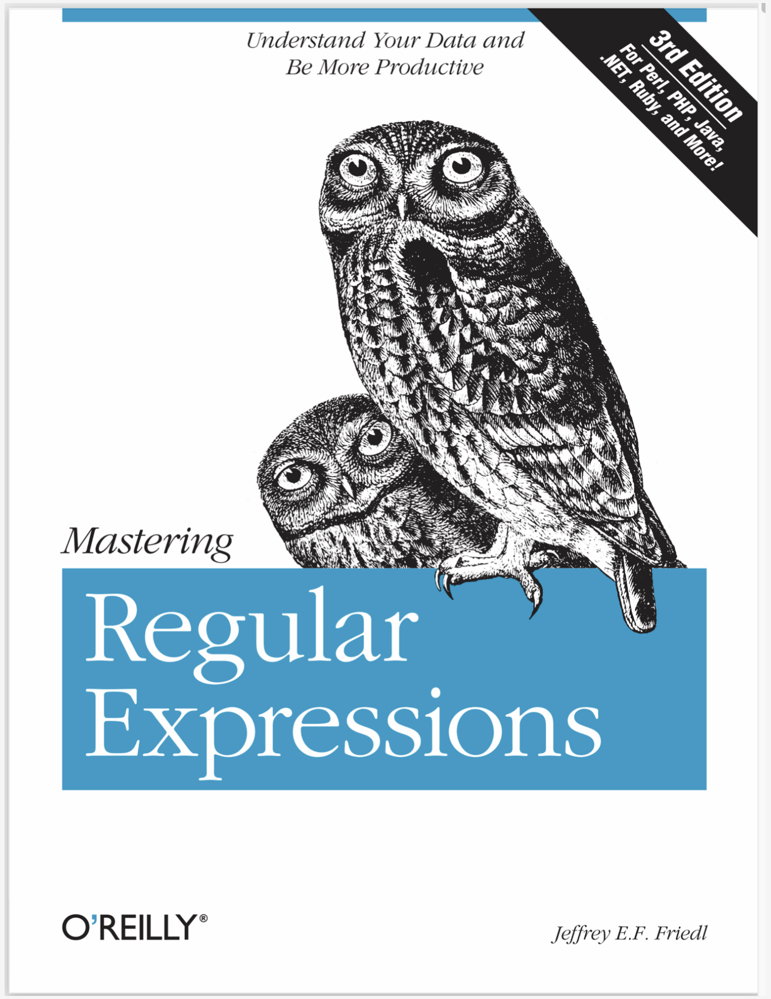
Regular Expressions

Or, waiter, there appears to be a language inside my language

stringr is your gateway to regexps

```r
library(stringr) # It's loaded by default with library(tidyverse)
```

regexps are their own whole world


Understand Your Data and Be More Productive
3rd Edition
For Perl, PHP, Java, .NET, Ruby, and More!
Mastering
Regular Expressions
O'REILLY
Jeffrey E.F. Friedl

This book is a thing of beauty.

Searching for patterns

A regular expression is a way of searching for a piece of text, or *pattern*, inside some larger body of text, called a *string*.

Searching for patterns

A regular expression is a way of searching for a piece of text, or *pattern*, inside some larger body of text, called a *string*.

The simplest sort of search is like the "Find" functionality in a Word Processor, where the pattern is a literal letter, number, punctuation mark, word or series of words and the text is a document that gets searched one line at a time. The next step up is "Find and Replace".

Searching for patterns

A regular expression is a way of searching for a piece of text, or *pattern*, inside some larger body of text, called a *string*.

The simplest sort of search is like the "Find" functionality in a Word Processor, where the pattern is a literal letter, number, punctuation mark, word or series of words and the text is a document that gets searched one line at a time. The next step up is "Find and Replace".

Every pattern-searching function in `stringr` has the same basic form:

```
str_view(<STRING>, <PATTERN>, [...], html=TRUE) # where [...] means "maybe some options"
```

Searching for patterns

A regular expression is a way of searching for a piece of text, or *pattern*, inside some larger body of text, called a *string*.

The simplest sort of search is like the "Find" functionality in a Word Processor, where the pattern is a literal letter, number, punctuation mark, word or series of words and the text is a document that gets searched one line at a time. The next step up is "Find and Replace".

Every pattern-searching function in `stringr` has the same basic form:

```
str_view(<STRING>, <PATTERN>, [...], html=TRUE) # where [...] means "maybe some options"
```

Functions that *replace* as well as *detect* strings all have this form:

```
str_replace(<STRING>, <PATTERN>, <REPLACEMENT>)
```

Searching for patterns

A regular expression is a way of searching for a piece of text, or *pattern*, inside some larger body of text, called a *string*.

The simplest sort of search is like the "Find" functionality in a Word Processor, where the pattern is a literal letter, number, punctuation mark, word or series of words and the text is a document that gets searched one line at a time. The next step up is "Find and Replace".

Every pattern-searching function in `stringr` has the same basic form:

```
str_view(<STRING>, <PATTERN>, [...], html=TRUE) # where [...] means "maybe some options"
```

Functions that *replace* as well as *detect* strings all have this form:

```
str_replace(<STRING>, <PATTERN>, <REPLACEMENT>)
```

(If you think about it, `<STRING>`, `<PATTERN>` and `<REPLACEMENT>` above are all kinds of pattern: they are meant to "stand for" all kinds of text, not be taken literally.)

Searching for patterns

Here I'll follow the exposition in Wickham & Grolemund (2017).

```r
x ← c("apple", "banana", "pear")

str_view(x, "an", html=TRUE)
```

```
banana
```

Searching for patterns

Regular expressions get their real power from *wildcards*, i.e. tokens that match more than just literal strings, but also more general and more complex patterns.

Searching for patterns

Regular expressions get their real power from *wildcards*, i.e. tokens that match more than just literal strings, but also more general and more complex patterns.

The most general pattern-matching token is, "Match everything!" This is represented by the period, or `.`]

Searching for patterns

Regular expressions get their real power from *wildcards*, i.e. tokens that match more than just literal strings, but also more general and more complex patterns.

The most general pattern-matching token is, "Match everything!" This is represented by the period, or `.`]

But ... if “`.`” matches any character, how do you specifically match the character “`.`”?

Escaping

You have to "escape" the period to tell the regex you want to match it exactly, rather than interpret it as meaning "match anything".

Escaping

You have to "escape" the period to tell the regex you want to match it exactly, rather than interpret it as meaning "match anything".

regexs use the backslash, \, to signal "escape the next character".

Escaping

You have to "escape" the period to tell the regex you want to match it exactly, rather than interpret it as meaning "match anything".

regexs use the backslash, `\`, to signal "escape the next character".

To match a ".", you need the regex "`\.`"

Hang on, I see a further problem

We use strings to represent regular expressions. `\` is also used as an escape symbol in strings. So to create the regular expression `\.` we need the string `"\\."`

```r
# To create the regular expression, we need \\
dot <- "\\."

# But the expression itself only contains one:
writeLines(dot)
```

```
## \.
```

```r
# And this tells R to look for an explicit .
str_view(c("abc", "a.c", "bef"), "a\\.c", html=TRUE)
```

```
a.c
```

But ... then how do you match a literal \?

```r
x <- "a\\b"
writeLines(x)
```

```
## a\b
```

```r
#> a\b

str_view(x, "\\\\", html=TRUE) # you need four!
```

```
a\b
```

But … then how do you match a literal \\?

This is the price we pay for having to express searches for patterns using a language containing these same characters, which we may also want to search for.

I *promise* this will pay off

Use `^` to match the start of a string.

Use `$` to match the end of a string.

I *promise* this will pay off

Use **^** to match the start of a string.

Use **$** to match the end of a string.

```r
x <- c("apple", "banana", "pear")
str_view(x, "^a", html=TRUE)
```

```
apple
```

I *promise* this will pay off

Use `^` to match the start of a string.

Use `$` to match the end of a string.

```r
x <- c("apple", "banana", "pear")
str_view(x, "^a", html=TRUE)
```

```
apple
```

```r
str_view(x, "a$", html=TRUE)
```

```
banana
```

Matching start and end

To force a regular expression to only match a complete string, anchor it with both `^` and `$`

Matching start and end

To force a regular expression to only match a complete string, anchor it with both ^ and $

```r
x <- c("apple pie", "apple", "apple cake")
str_view(x, "apple", html=TRUE)
```

```
apple pie
apple
apple cake
```

Matching start and end

To force a regular expression to only match a complete string, anchor it with both ^ and $

```r
x <- c("apple pie", "apple", "apple cake")
str_view(x, "apple", html=TRUE)
```

```
apple pie
apple
apple cake
```

```r
str_view(x, "^apple$", html=TRUE)
```

```
apple
```

Matching character classes

`\d` matches any digit.

`\s` matches any whitespace (e.g. space, tab, newline).

`[abc]` matches a, b, or c.

`[^abc]` matches anything except a, b, or c.

Matching the *special* characters

Look for a literal character that normally has special meaning in a regex

```r
str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c", html=TRUE)
```

```
a.c
```

Matching the *special* characters

Look for a literal character that normally has special meaning in a regex

```r
str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c", html=TRUE)
```

```
a.c
```

```r
str_view(c("abc", "a.c", "a*c", "a c"), ".[*]c", html=TRUE)
```

```
a*c
```

Alternation

Use parentheses to make the precedence of `|` clear:

```
str_view(c("groy", "grey", "griy", "gray"), "gr(e|a)y", html=TRUE)
```

```
grey
gray
```

Repeated patterns

? is 0 or 1

+ is 1 or more

***** is 0 or more

```r
x <- "1888 is the longest year in Roman numerals: MDCCCLXXXVIII"
str_view(x, "CC?", html=TRUE)
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Repeated patterns

? is 0 or 1

+ is 1 or more

***** is 0 or more

```r
str_view(x, "CC+", html=TRUE)
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Repeated patterns

? is 0 or 1

+ is 1 or more

__*__ is 0 or more

```r
x <- "1888 is the longest year in Roman numerals: MDCCCLXXXVIII"
str_view(x, 'C[LX]+', html=TRUE)
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Exact numbers of repetitions

`{n}` is exactly n

`{n,}` is n or more

`{,m}` is at most m

`{n,m}` is between n and m

```
str_view(x, "C{2}", html=TRUE)
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Exact numbers of repetitions

`{n}` is exactly n

`{n,}` is n or more

`{,m}` is at most m

`{n,m}` is between n and m

```
str_view(x, "C{2,}", html=TRUE)
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

Exact numbers of repetitions

`{n}` is exactly n

`{n,}` is n or more

`{,m}` is at most m

`{n,m}` is between n and m

By default these are *greedy* matches. You can make them “lazy”, matching the shortest string possible by putting a **?** after them. **This is often very useful!**

```
str_view(x, 'C[LX]+?', html=TRUE)
```

```
1888 is the longest year in Roman numerals: MDCCCLXXXVIII
```

And finally ... backreferences

```r
fruit # built into stringr
```

```
##  [1] "apple"             "apricot"           "avocado"
##  [4] "banana"            "bell pepper"       "bilberry"
##  [7] "blackberry"        "blackcurrant"      "blood orange"
## [10] "blueberry"         "boysenberry"       "breadfruit"
## [13] "canary melon"      "cantaloupe"        "cherimoya"
## [16] "cherry"            "chili pepper"      "clementine"
## [19] "cloudberry"        "coconut"           "cranberry"
## [22] "cucumber"          "currant"           "damson"
## [25] "date"              "dragonfruit"       "durian"
## [28] "eggplant"          "elderberry"        "feijoa"
## [31] "fig"               "goji berry"        "gooseberry"
## [34] "grape"             "grapefruit"        "guava"
## [37] "honeydew"          "huckleberry"       "jackfruit"
## [40] "jambul"            "jujube"            "kiwi fruit"
## [43] "kumquat"           "lemon"             "lime"
## [46] "loquat"            "lychee"            "mandarine"
## [49] "mango"             "mulberry"          "nectarine"
## [52] "nut"               "olive"             "orange"
## [55] "pamelo"            "papaya"            "passionfruit"
## [58] "peach"             "pear"              "persimmon"
## [61] "physalis"          "pineapple"         "plum"
## [64] "pomegranate"       "pomelo"            "purple mangosteen"
## [67] "quince"            "raisin"            "rambutan"
## [70] "raspberry"         "redcurrant"        "rock melon"
## [73] "salal berry"       "satsuma"           "star fruit"
## [76] "strawberry"        "tamarillo"         "tangerine"
## [79] "ugli fruit"        "watermelon"
```

Grouping and backreferences

Find all fruits that have a repeated pair of letters:

```
str_view(fruit, "(..)\\1", match = TRUE, html=TRUE)
```

```
banana
coconut
cucumber
jujube
papaya
salal berry
```

Grouping and backreferences

Backreferences and grouping will be very useful for string *replacements*.

OK that was a lot

Learning and testing regexps

Practice with a tester like https://regexr.com

Or an app like Patterns

The regex engine or "flavor" used by `stringr` is Perl- or PCRE-like.

Example: Politics and Placenames

```r
library(ukelection2019)
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019
```

```
## # A tibble: 3,320 × 13
##    cid       constituency electorate party_name cand
##    <chr>     <chr>             <int> <chr>      <chr
##  1 W07000… Aberavon          50747 Labour     Step
##  2 W07000… Aberavon          50747 Conservat… Char
##  3 W07000… Aberavon          50747 The Brexi… Glen
##  4 W07000… Aberavon          50747 Plaid Cym… Nige
##  5 W07000… Aberavon          50747 Liberal D… Shei
##  6 W07000… Aberavon          50747 Independe… Capt
##  7 W07000… Aberavon          50747 Green      Gior
##  8 W07000… Aberconwy         44699 Conservat… Robi
##  9 W07000… Aberconwy         44699 Labour     Emil
## 10 W07000… Aberconwy         44699 Plaid Cym… Lisa
## # ℹ 3,310 more rows
## # ℹ 6 more variables: vote_share_change <dbl>, tot
## #   vrank <int>, turnout <dbl>, fname <chr>, lname
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 ▷
  group_by(constituency)
```

```
## # A tibble: 3,320 × 13
## # Groups:   constituency [650]
##    cid      constituency electorate party_name cand
##    <chr>    <chr>             <int> <chr>      <chr
##  1 W07000… Aberavon          50747 Labour     Step
##  2 W07000… Aberavon          50747 Conservat… Char
##  3 W07000… Aberavon          50747 The Brexi… Glen
##  4 W07000… Aberavon          50747 Plaid Cym… Nige
##  5 W07000… Aberavon          50747 Liberal D… Shei
##  6 W07000… Aberavon          50747 Independe… Capt
##  7 W07000… Aberavon          50747 Green      Gior
##  8 W07000… Aberconwy         44699 Conservat… Robi
##  9 W07000… Aberconwy         44699 Labour     Emil
## 10 W07000… Aberconwy         44699 Plaid Cym… Lisa
## # ℹ 3,310 more rows
## # ℹ 6 more variables: vote_share_change <dbl>, tot
## #   vrank <int>, turnout <dbl>, fname <chr>, lname
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 ▷
  group_by(constituency) ▷
  slice_max(votes)
```

```
## # A tibble: 650 × 13
## # Groups:   constituency [650]
##    cid      constituency electorate party_name cand
##    <chr>    <chr>             <int> <chr>      <chr
##  1 W07000… Aberavon          50747 Labour     Step
##  2 W07000… Aberconwy         44699 Conservat… Robi
##  3 S14000… Aberdeen No…      62489 Scottish … Kirs
##  4 S14000… Aberdeen So…      65719 Scottish … Step
##  5 S14000… Aberdeenshi…      72640 Conservat… Andr
##  6 S14000… Airdrie & S…      64008 Scottish … Neil
##  7 E14000… Aldershot         72617 Conservat… Leo
##  8 E14000… Aldridge-Br…      60138 Conservat… Wend
##  9 E14000… Altrincham …      73096 Conservat… Grah
## 10 W07000… Alyn & Dees…      62783 Labour     Mark
## # ℹ 640 more rows
## # ℹ 6 more variables: vote_share_change <dbl>, tot
## #   vrank <int>, turnout <dbl>, fname <chr>, lname
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 ▷
  group_by(constituency) ▷
  slice_max(votes) ▷
  ungroup()
```

```
## # A tibble: 650 × 13
##    cid      constituency electorate party_name cand
##    <chr>    <chr>             <int> <chr>      <chr
##  1 W07000… Aberavon           50747 Labour     Step
##  2 W07000… Aberconwy          44699 Conservat… Robi
##  3 S14000… Aberdeen No…       62489 Scottish … Kirs
##  4 S14000… Aberdeen So…       65719 Scottish … Step
##  5 S14000… Aberdeenshi…       72640 Conservat… Andr
##  6 S14000… Airdrie & S…       64008 Scottish … Neil
##  7 E14000… Aldershot          72617 Conservat… Leo
##  8 E14000… Aldridge-Br…       60138 Conservat… Wend
##  9 E14000… Altrincham …       73096 Conservat… Grah
## 10 W07000… Alyn & Dees…       62783 Labour     Mark
## # ℹ 640 more rows
## # ℹ 6 more variables: vote_share_change <dbl>, tot
## #   vrank <int>, turnout <dbl>, fname <chr>, lname
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 ▷
  group_by(constituency) ▷
  slice_max(votes) ▷
  ungroup() ▷
  select(constituency, party_name)
```

```
## # A tibble: 650 × 2
##    constituency                      party_name
##    <chr>                             <chr>
##  1 Aberavon                          Labour
##  2 Aberconwy                         Conservative
##  3 Aberdeen North                    Scottish Nationa
##  4 Aberdeen South                    Scottish Nationa
##  5 Aberdeenshire West & Kincardine Conservative
##  6 Airdrie & Shotts                  Scottish Nationa
##  7 Aldershot                         Conservative
##  8 Aldridge-Brownhills               Conservative
##  9 Altrincham & Sale West            Conservative
## 10 Alyn & Deeside                    Labour
## # ℹ 640 more rows
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 ▷
  group_by(constituency) ▷
  slice_max(votes) ▷
  ungroup() ▷
  select(constituency, party_name) ▷
  mutate(shire = str_detect(constituency, "shire"),
         field = str_detect(constituency, "field"),
         dale = str_detect(constituency, "dale"),
         pool = str_detect(constituency, "pool"),
         ton = str_detect(constituency, "(ton$)|(ton )"),
         wood = str_detect(constituency, "(wood$)|(wood )"),
         saint = str_detect(constituency, "(St )|(Saint)"),
         port = str_detect(constituency, "(Port)|(port)"),
         ford = str_detect(constituency, "(ford$)|(ford )"),
         by = str_detect(constituency, "(by$)|(by )"),
         boro = str_detect(constituency, "(boro$)|(boro )|(borough$)|(borou
         ley = str_detect(constituency, "(ley$)|(ley )|(leigh$)|(leigh )"))
```

```
## # A tibble: 650 × 14
##    constituency party_name shire field dale  pool
##    <chr>        <chr>      <lgl> <lgl> <lgl> <lgl>
##  1 Aberavon     Labour     FALSE FALSE FALSE FALSE
##  2 Aberconwy    Conservat… FALSE FALSE FALSE FALSE
##  3 Aberdeen No… Scottish … FALSE FALSE FALSE FALSE
##  4 Aberdeen So… Scottish … FALSE FALSE FALSE FALSE
##  5 Aberdeenshi… Conservat… TRUE  FALSE FALSE FALSE
##  6 Airdrie & S… Scottish … FALSE FALSE FALSE FALSE
##  7 Aldershot    Conservat… FALSE FALSE FALSE FALSE
##  8 Aldridge-Br… Conservat… FALSE FALSE FALSE FALSE
##  9 Altrincham … Conservat… FALSE FALSE FALSE FALSE
## 10 Alyn & Dees… Labour     FALSE FALSE FALSE FALSE
## # ℹ 640 more rows
## # ℹ 3 more variables: by <lgl>, boro <lgl>, ley <l
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 |>
  group_by(constituency) |>
  slice_max(votes) |>
  ungroup() |>
  select(constituency, party_name) |>
  mutate(shire = str_detect(constituency, "shire"),
         field = str_detect(constituency, "field"),
         dale = str_detect(constituency, "dale"),
         pool = str_detect(constituency, "pool"),
         ton = str_detect(constituency, "(ton$)|(ton )"),
         wood = str_detect(constituency, "(wood$)|(wood )"),
         saint = str_detect(constituency, "(St )|(Saint)"),
         port = str_detect(constituency, "(Port)|(port)"),
         ford = str_detect(constituency, "(ford$)|(ford )"),
         by = str_detect(constituency, "(by$)|(by )"),
         boro = str_detect(constituency, "(boro$)|(boro )|(borough$)|(borou
         ley = str_detect(constituency, "(ley$)|(ley )|(leigh$)|(leigh )"))
  pivot_longer(shire:ley, names_to = "toponym")
```

```
## # A tibble: 7,800 × 4
##    constituency party_name toponym value
##    <chr>        <chr>      <chr>   <lgl>
##  1 Aberavon     Labour     shire   FALSE
##  2 Aberavon     Labour     field   FALSE
##  3 Aberavon     Labour     dale    FALSE
##  4 Aberavon     Labour     pool    FALSE
##  5 Aberavon     Labour     ton     FALSE
##  6 Aberavon     Labour     wood    FALSE
##  7 Aberavon     Labour     saint   FALSE
##  8 Aberavon     Labour     port    FALSE
##  9 Aberavon     Labour     ford    FALSE
## 10 Aberavon     Labour     by      FALSE
## # ℹ 7,790 more rows
```

Example: Politics and Placenames

```r
place_tab ← ukvote2019 ▷
  group_by(constituency) ▷
  slice_max(votes) ▷
  ungroup() ▷
  select(constituency, party_name) ▷
  mutate(shire = str_detect(constituency, "shire"),
         field = str_detect(constituency, "field"),
         dale = str_detect(constituency, "dale"),
         pool = str_detect(constituency, "pool"),
         ton = str_detect(constituency, "(ton$)|(ton )"),
         wood = str_detect(constituency, "(wood$)|(wood )"),
         saint = str_detect(constituency, "(St )|(Saint)"),
         port = str_detect(constituency, "(Port)|(port)"),
         ford = str_detect(constituency, "(ford$)|(ford )"),
         by = str_detect(constituency, "(by$)|(by )"),
         boro = str_detect(constituency, "(boro$)|(boro )|(borough$)|(borough )"),
         ley = str_detect(constituency, "(ley$)|(ley )|(leigh$)|(leigh )")) ▷
  pivot_longer(shire:ley, names_to = "toponym")
```

Example: Politics and Placenames

```r
place_tab
```

```
## # A tibble: 7,800 × 4
##    constituency party_name toponym value
##    <chr>        <chr>      <chr>   <lgl>
##  1 Aberavon     Labour     shire   FALSE
##  2 Aberavon     Labour     field   FALSE
##  3 Aberavon     Labour     dale    FALSE
##  4 Aberavon     Labour     pool    FALSE
##  5 Aberavon     Labour     ton     FALSE
##  6 Aberavon     Labour     wood    FALSE
##  7 Aberavon     Labour     saint   FALSE
##  8 Aberavon     Labour     port    FALSE
##  9 Aberavon     Labour     ford    FALSE
## 10 Aberavon     Labour     by      FALSE
## # ℹ 7,790 more rows
```

Example: Politics and Placenames

```
place_tab ▷
  group_by(party_name, toponym)
```

```
## # A tibble: 7,800 × 4
## # Groups:   party_name, toponym [120]
##    constituency party_name toponym value
##    <chr>        <chr>      <chr>   <lgl>
##  1 Aberavon     Labour     shire   FALSE
##  2 Aberavon     Labour     field   FALSE
##  3 Aberavon     Labour     dale    FALSE
##  4 Aberavon     Labour     pool    FALSE
##  5 Aberavon     Labour     ton     FALSE
##  6 Aberavon     Labour     wood    FALSE
##  7 Aberavon     Labour     saint   FALSE
##  8 Aberavon     Labour     port    FALSE
##  9 Aberavon     Labour     ford    FALSE
## 10 Aberavon     Labour     by      FALSE
## # ℹ 7,790 more rows
```

Example: Politics and Placenames

```r
place_tab ▷
  group_by(party_name, toponym) ▷
  filter(party_name %in% c("Conservative", "Labour"))
```

```
## # A tibble: 6,816 × 4
## # Groups:   party_name, toponym [24]
##    constituency party_name toponym value
##    <chr>        <chr>      <chr>   <lgl>
##  1 Aberavon     Labour     shire   FALSE
##  2 Aberavon     Labour     field   FALSE
##  3 Aberavon     Labour     dale    FALSE
##  4 Aberavon     Labour     pool    FALSE
##  5 Aberavon     Labour     ton     FALSE
##  6 Aberavon     Labour     wood    FALSE
##  7 Aberavon     Labour     saint   FALSE
##  8 Aberavon     Labour     port    FALSE
##  9 Aberavon     Labour     ford    FALSE
## 10 Aberavon     Labour     by      FALSE
## # ℹ 6,806 more rows
```

Example: Politics and Placenames

```r
place_tab ▷
  group_by(party_name, toponym) ▷
  filter(party_name %in% c("Conservative", "Labour")) ▷
  group_by(toponym, party_name)
```

```
## # A tibble: 6,816 × 4
## # Groups:   toponym, party_name [24]
##    constituency party_name toponym value
##    <chr>        <chr>      <chr>   <lgl>
##  1 Aberavon     Labour     shire   FALSE
##  2 Aberavon     Labour     field   FALSE
##  3 Aberavon     Labour     dale    FALSE
##  4 Aberavon     Labour     pool    FALSE
##  5 Aberavon     Labour     ton     FALSE
##  6 Aberavon     Labour     wood    FALSE
##  7 Aberavon     Labour     saint   FALSE
##  8 Aberavon     Labour     port    FALSE
##  9 Aberavon     Labour     ford    FALSE
## 10 Aberavon     Labour     by      FALSE
## # ℹ 6,806 more rows
```

Example: Politics and Placenames

```r
place_tab ▷
  group_by(party_name, toponym) ▷
  filter(party_name %in% c("Conservative", "Labour")) ▷
  group_by(toponym, party_name) ▷
  summarize(freq = sum(value))
```

```
## # A tibble: 24 × 3
## # Groups:   toponym [12]
##    toponym party_name    freq
##    <chr>   <chr>        <int>
##  1 boro    Conservative     7
##  2 boro    Labour           1
##  3 by      Conservative     6
##  4 by      Labour           2
##  5 dale    Conservative     3
##  6 dale    Labour           1
##  7 field   Conservative    10
##  8 field   Labour          10
##  9 ford    Conservative    17
## 10 ford    Labour          12
## # ℹ 14 more rows
```

Example: Politics and Placenames

```r
place_tab ▷
  group_by(party_name, toponym) ▷
  filter(party_name %in% c("Conservative", "Labour")) ▷
  group_by(toponym, party_name) ▷
  summarize(freq = sum(value)) ▷
  mutate(pct = freq/sum(freq))
```

```
## # A tibble: 24 × 4
## # Groups:   toponym [12]
##    toponym party_name    freq   pct
##    <chr>   <chr>        <int> <dbl>
##  1 boro    Conservative     7 0.875
##  2 boro    Labour           1 0.125
##  3 by      Conservative     6 0.75
##  4 by      Labour           2 0.25
##  5 dale    Conservative     3 0.75
##  6 dale    Labour           1 0.25
##  7 field   Conservative    10 0.5
##  8 field   Labour          10 0.5
##  9 ford    Conservative    17 0.586
## 10 ford    Labour          12 0.414
## # ℹ 14 more rows
```

Example: Politics and Placenames

```r
place_tab |>
  group_by(party_name, toponym) |>
  filter(party_name %in% c("Conservative", "Labour")) |>
  group_by(toponym, party_name) |>
  summarize(freq = sum(value)) |>
  mutate(pct = freq/sum(freq)) |>
  filter(party_name == "Conservative")
```

```
## # A tibble: 12 × 4
## # Groups:   toponym [12]
##    toponym party_name    freq   pct
##    <chr>   <chr>        <int> <dbl>
##  1 boro    Conservative     7 0.875
##  2 by      Conservative     6 0.75
##  3 dale    Conservative     3 0.75
##  4 field   Conservative    10 0.5
##  5 ford    Conservative    17 0.586
##  6 ley     Conservative    26 0.722
##  7 pool    Conservative     2 0.286
##  8 port    Conservative     3 0.333
##  9 saint   Conservative     3 0.5
## 10 shire   Conservative    37 0.974
## 11 ton     Conservative    37 0.507
## 12 wood    Conservative     7 0.636
```

Example: Politics and Placenames

```r
place_tab |>
  group_by(party_name, toponym) |>
  filter(party_name %in% c("Conservative", "Labour")) |>
  group_by(toponym, party_name) |>
  summarize(freq = sum(value)) |>
  mutate(pct = freq/sum(freq)) |>
  filter(party_name == "Conservative") |>
  arrange(desc(pct))
```

```
## # A tibble: 12 × 4
## # Groups:   toponym [12]
##    toponym party_name    freq   pct
##    <chr>   <chr>        <int> <dbl>
##  1 shire   Conservative    37 0.974
##  2 boro    Conservative     7 0.875
##  3 by      Conservative     6 0.75
##  4 dale    Conservative     3 0.75
##  5 ley     Conservative    26 0.722
##  6 wood    Conservative     7 0.636
##  7 ford    Conservative    17 0.586
##  8 ton     Conservative    37 0.507
##  9 field   Conservative    10 0.5
## 10 saint   Conservative     3 0.5
## 11 port    Conservative     3 0.333
## 12 pool    Conservative     2 0.286
```

Example: Politics and Placenames

```r
place_tab ▷
  group_by(party_name, toponym) ▷
  filter(party_name %in% c("Conservative", "Labour")) ▷
  group_by(toponym, party_name) ▷
  summarize(freq = sum(value)) ▷
  mutate(pct = freq/sum(freq)) ▷
  filter(party_name == "Conservative") ▷
  arrange(desc(pct))
```

```
## # A tibble: 12 × 4
## # Groups:   toponym [12]
##    toponym party_name    freq   pct
##    <chr>   <chr>        <int> <dbl>
##  1 shire   Conservative    37 0.974
##  2 boro    Conservative     7 0.875
##  3 by      Conservative     6 0.75
##  4 dale    Conservative     3 0.75
##  5 ley     Conservative    26 0.722
##  6 wood    Conservative     7 0.636
##  7 ford    Conservative    17 0.586
##  8 ton     Conservative    37 0.507
##  9 field   Conservative    10 0.5
## 10 saint   Conservative     3 0.5
## 11 port    Conservative     3 0.333
## 12 pool    Conservative     2 0.286
```