A brief introduction to **regular expressions**

Data Wrangling: Session 5

Kieran Healy

Statistical Horizons, December 2022

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)
```

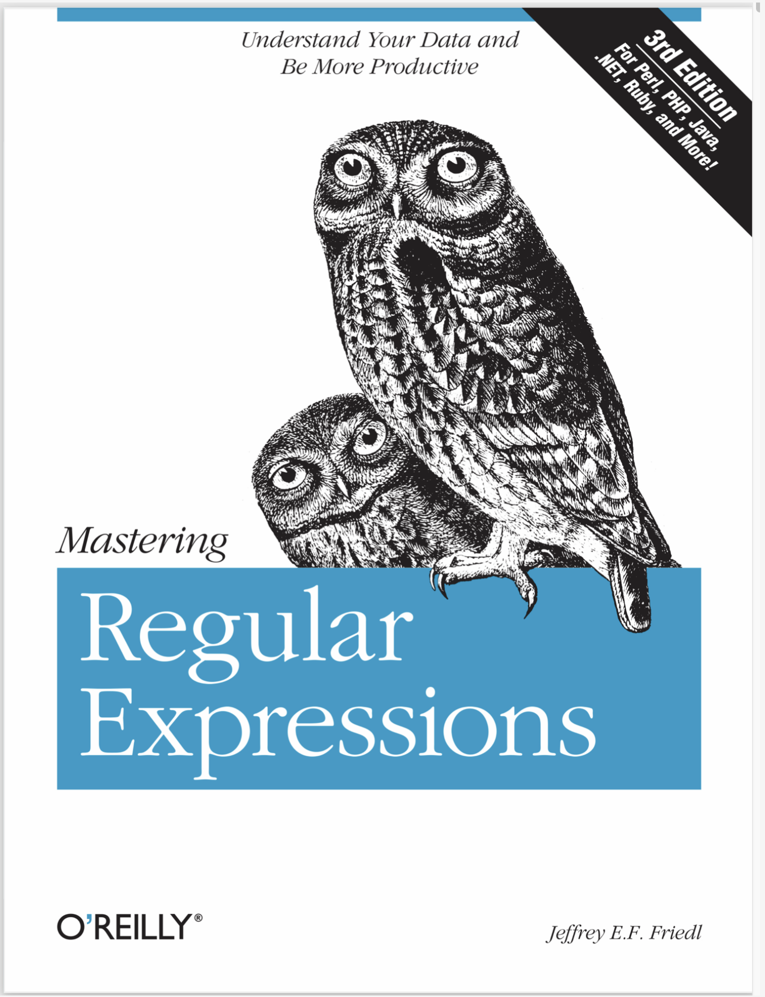
Regular Expressions

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>, [...]) # 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>, [...]) # 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>, [...]) # where [...] means "maybe some options"
```

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

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

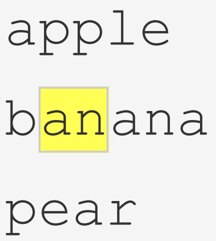
(If you think about it, `<STRING>`, `<PATTERN>` and `<REPLACEMENT>` above are all kinds of pattern: they are meant to "stand for" all kinds of text, not be taken literally.)

Searching for patterns

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

```r
x <- c("apple", "banana", "pear")

str_view(x, "an")
```



```
apple

banana

pear
```

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")
```

```
abc

a.c

bef
```

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

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

```
## a\b
```

```
#> a\b

str_view(x, "\\\\") # 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.

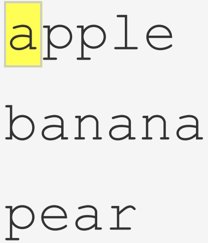
Use `$` to match the end of a string.

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")
```



```
apple
banana
pear
```

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")
```

```
apple

banana

pear
```

```r
str_view(x, "a$")
```

```
apple

banana

pear
```

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")
```

```
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")
```

```
apple pie
apple
apple cake
```

```r
str_view(x, "^apple$")
```

```
apple pie
apple
apple cake
```

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")
```

```
abc
a.c
a*c
a c
```

Matching the *special* characters

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

```
str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c")
```

```
abc

a.c

a*c

a c
```

```
str_view(c("abc", "a.c", "a*c", "a c"), ".[*]c")
```

```
abc

a.c

a*c

a c
```

Alternation

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

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

```
groy
grey
griy
gray
```

Repeated patterns

? is 0 or 1

+ is 1 or more

* is 0 or more

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

```
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+")
```

```
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]+')
```

```
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

```r
str_view(x, "C{2}")
```

```
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,}")
```

```
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]+?')
```

```
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:

```r
str_view(fruit, "(..)\\1", match = 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       const…¹ elect…² party…³ candi…⁴ votes
##    <chr>     <chr>     <int> <chr>   <chr>   <int>
##  1 W07000049 Aberav…   50747 Labour  Stephe… 17008
##  2 W07000049 Aberav…   50747 Conser… Charlo…  6518
##  3 W07000049 Aberav…   50747 The Br… Glenda…  3108
##  4 W07000049 Aberav…   50747 Plaid … Nigel …  2711
##  5 W07000049 Aberav…   50747 Libera… Sheila…  1072
##  6 W07000049 Aberav…   50747 Indepe… Captai…   731
##  7 W07000049 Aberav…   50747 Green   Giorgi…   450
##  8 W07000058 Aberco…   44699 Conser… Robin … 14687
##  9 W07000058 Aberco…   44699 Labour  Emily … 12653
## 10 W07000058 Aberco…   44699 Plaid … Lisa G…  2704
## # … with 3,310 more rows, 3 more variables: turnou
## #   lname <chr>, and abbreviated variable names ¹co
## #   ³party_name, ⁴candidate, ⁵vote_share_percent,
## #   ⁷total_votes_cast
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 |>
  group_by(constituency)
```

```
## # A tibble: 3,320 × 13
## # Groups:   constituency [650]
##    cid       const…¹ elect…² party…³ candi…⁴ votes
##    <chr>     <chr>     <int> <chr>   <chr>   <int>
##  1 W07000049 Aberav…   50747 Labour  Stephe… 17008
##  2 W07000049 Aberav…   50747 Conser… Charlo…  6518
##  3 W07000049 Aberav…   50747 The Br… Glenda…  3108
##  4 W07000049 Aberav…   50747 Plaid … Nigel …  2711
##  5 W07000049 Aberav…   50747 Libera… Sheila…  1072
##  6 W07000049 Aberav…   50747 Indepe… Captai…   731
##  7 W07000049 Aberav…   50747 Green   Giorgi…   450
##  8 W07000058 Aberco…   44699 Conser… Robin … 14687
##  9 W07000058 Aberco…   44699 Labour  Emily … 12653
## 10 W07000058 Aberco…   44699 Plaid … Lisa G…  2704
## # … with 3,310 more rows, 3 more variables: turnou
## #   lname <chr>, and abbreviated variable names ¹c
## #   ³party_name, ⁴candidate, ⁵vote_share_percent,
## #   ⁷total_votes_cast
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 |>
  group_by(constituency) |>
  slice_max(votes)
```

```
## # A tibble: 650 × 13
## # Groups:   constituency [650]
##    cid       const…¹ elect…² party…³ candi…⁴ votes
##    <chr>     <chr>     <int> <chr>   <chr>   <int>
##  1 W07000049 Aberav…   50747 Labour  Stephe… 17008
##  2 W07000058 Aberco…   44699 Conser… Robin … 14687
##  3 S14000001 Aberde…   62489 Scotti… Kirsty… 20205
##  4 S14000002 Aberde…   65719 Scotti… Stephe… 20388
##  5 S14000058 Aberde…   72640 Conser… Andrew… 22752
##  6 S14000003 Airdri…   64008 Scotti… Neil G… 17929
##  7 E14000530 Alders…   72617 Conser… Leo Do… 27980
##  8 E14000531 Aldrid…   60138 Conser… Wendy … 27850
##  9 E14000532 Altrin…   73096 Conser… Graham… 26311
## 10 W07000043 Alyn &…   62783 Labour  Mark T… 18271
## # … with 640 more rows, 3 more variables: turnout <
## #   lname <chr>, and abbreviated variable names ¹c
## #   ³party_name, ⁴candidate, ⁵vote_share_percent,
## #   ⁷total_votes_cast
```

Example: Politics and Placenames

```r
library(ukelection2019)

ukvote2019 |>
  group_by(constituency) |>
  slice_max(votes) |>
  ungroup()
```

```
## # A tibble: 650 × 13
##    cid       const…¹ elect…² party…³ candi…⁴ votes
##    <chr>     <chr>     <int> <chr>   <chr>   <int>
##  1 W07000049 Aberav…   50747 Labour  Stephe… 17008
##  2 W07000058 Aberco…   44699 Conser… Robin … 14687
##  3 S14000001 Aberde…   62489 Scotti… Kirsty… 20205
##  4 S14000002 Aberde…   65719 Scotti… Stephe… 20388
##  5 S14000058 Aberde…   72640 Conser… Andrew… 22752
##  6 S14000003 Airdri…   64008 Scotti… Neil G… 17929
##  7 E14000530 Alders…   72617 Conser… Leo Do… 27980
##  8 E14000531 Aldrid…   60138 Conser… Wendy … 27850
##  9 E14000532 Altrin…   73096 Conser… Graham… 26311
## 10 W07000043 Alyn &…   62783 Labour  Mark T… 18271
## # … with 640 more rows, 3 more variables: turnout <
## #   lname <chr>, and abbreviated variable names ¹co
## #   ³party_name, ⁴candidate, ⁵vote_share_percent,
## #   ⁷total_votes_cast
```

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
## # … with 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
##    constit…¹ party…² shire field dale  pool  ton
##    <chr>     <chr>   <lgl> <lgl> <lgl> <lgl> <lgl>
##  1 Aberavon  Labour  FALSE FALSE FALSE FALSE FALSE
##  2 Aberconwy Conser… FALSE FALSE FALSE FALSE FALSE
##  3 Aberdeen… Scotti… FALSE FALSE FALSE FALSE FALSE
##  4 Aberdeen… Scotti… FALSE FALSE FALSE FALSE FALSE
##  5 Aberdeen… Conser… TRUE  FALSE FALSE FALSE FALSE
##  6 Airdrie … Scotti… FALSE FALSE FALSE FALSE FALSE
##  7 Aldershot Conser… FALSE FALSE FALSE FALSE FALSE
##  8 Aldridge… Conser… FALSE FALSE FALSE FALSE FALSE
##  9 Altrinch… Conser… FALSE FALSE FALSE FALSE FALSE
## 10 Alyn & D… Labour  FALSE FALSE FALSE FALSE FALSE
## # … with 640 more rows, 2 more variables: boro <lg
## #   abbreviated variable names ¹constituency, ²par
```

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
## # … with 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
## # … with 7,790 more rows
```

Example: Politics and Placenames

```r
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
## # … with 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
## # … with 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
## # … with 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
## # … with 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
## # … with 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
```