

TIDY VERSE PART 2

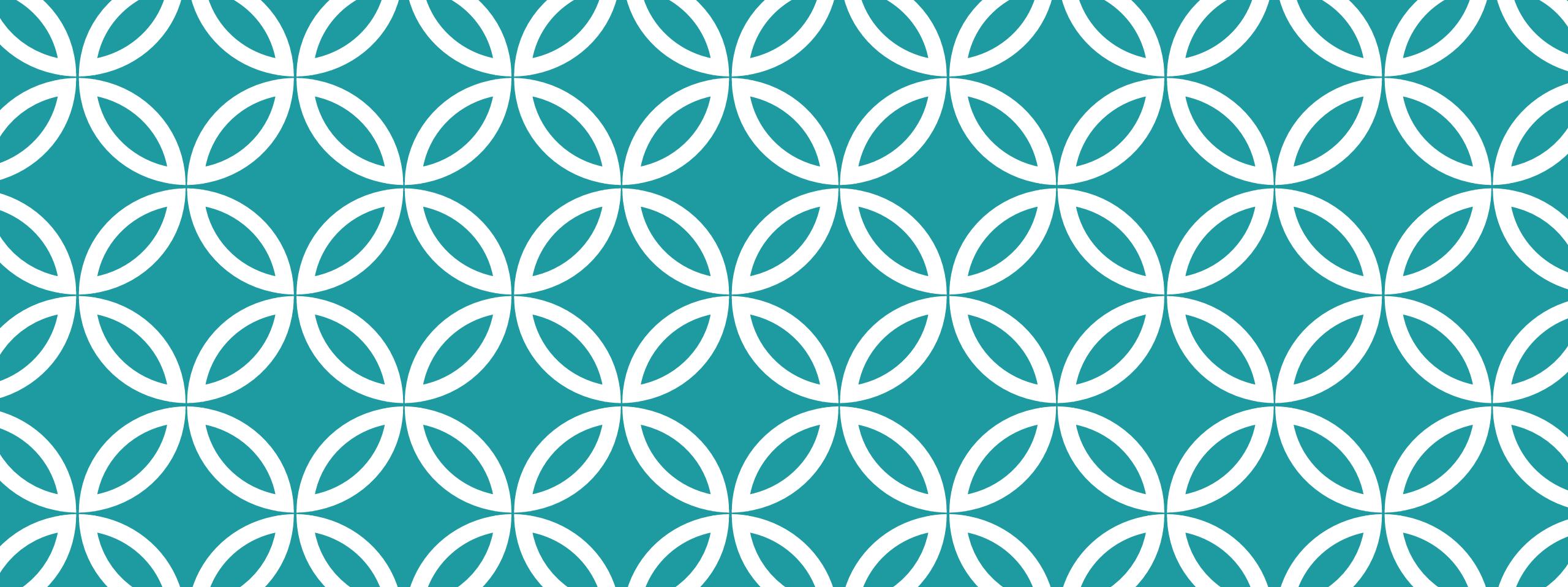
dr Maria Kubara, WNE UW

TIDYVERSE - MORE

Tidyverse is a huge environment of packages and functions which can help you with data modification in R. They are not necessarily needed to use R fluently (you can do all those things with base R commands) but learning tidyverse can make your coding more convenient and your code more readable.

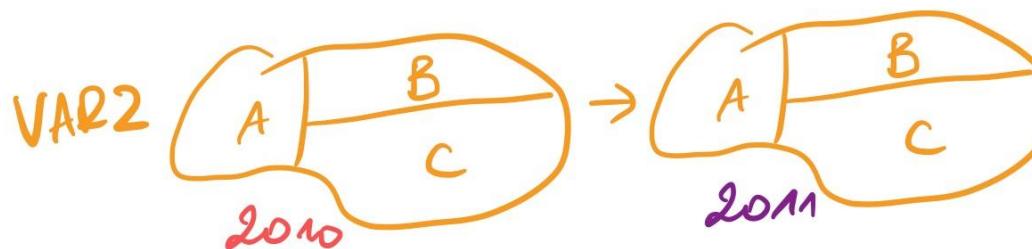
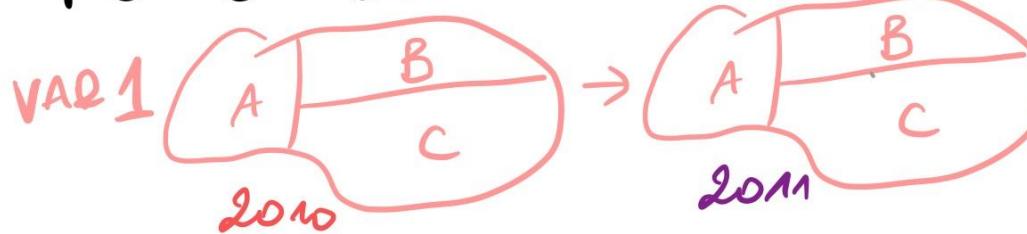
There are two key concepts on these slides → pivoting panel data and unnesting data. You should learn these functions, as they are likely to give you the most value-added for R coding. The remaining functions can be nice additions to your coding skills. Sometimes more convenient than standard base R coding.

**Remember – base codes work for most data objects.
(Most) tidyverse functions work on tabular data only
(`data.frames` or, better, `tibbles`) .**



RESHAPING PANEL DATA - TIDYR

Panel data



Wide

Place	2010 VAR1	2010 VAR2	2011 VAR1	2011 VAR2
A	x	x	x	x
B	x	x	x	x

Long

Place	Year	VAR1	VAR2
A	2010	x	x
A	2011	x	x
B	2010	x	x
B	2011	x	x

(Super) Long

Place	Year	Var	Count
A	2010	1	x
A	2010	2	x
A	2011	1	x
A	2011	2	x
B	2010	1	x
B	2010	2	x
B	2011	1	x
B	2011	2	x

Wide

Place 2010 2011

A

x x

B

x x

Long

Place Year VAR1

A 2010 x

A 2011 x

B 2010 x

B 2011 x

pivot_longer()

pivot_longer (DATA, cols = 2:3, names_to = "Year", values_to = "VAR1")

Common issue, transformation from wide to long is often needed. Long data are easier to be processed. However, panel data is usually reported in wide form. If you have more variables to be transformed, try to manage them one by one.

(Super) Long

Place	Year	VAR	Count
A	2010	VAR1	X
A	2010	VAR2	X
A	2011	VAR1	X
A	2011	VAR2	X
B	2010	VAR1	X
B	2010	VAR2	X
B	2011	VAR1	X
B	2011	VAR2	X

Long

Place	Year	VAR1	VAR2
A	2010	X	X
A	2011	X	X
B	2010	X	X
B	2011	X	X

pivot_wider()

pivot_wider(DATA, names_from = "VAR", values_from = "Count")

Data in „super long” form is too fragmentated. We can make it a bit „wider” with the usage of pivot_wider() function.

Reshape Data

- Pivot data to reorganize values into a new layout.

<https://github.com/rstudio/cheatsheets/blob/main/tidyr.pdf>

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K



country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name",
values_to = "value", values_drop_na = FALSE)

"Lengthen" data by collapsing several columns
into two. Column names move to a new
names_to column and values to a new values_to
column.

```
pivot_longer(table4a, cols = 2:3, names_to = "year",  
values_to = "cases")
```

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T



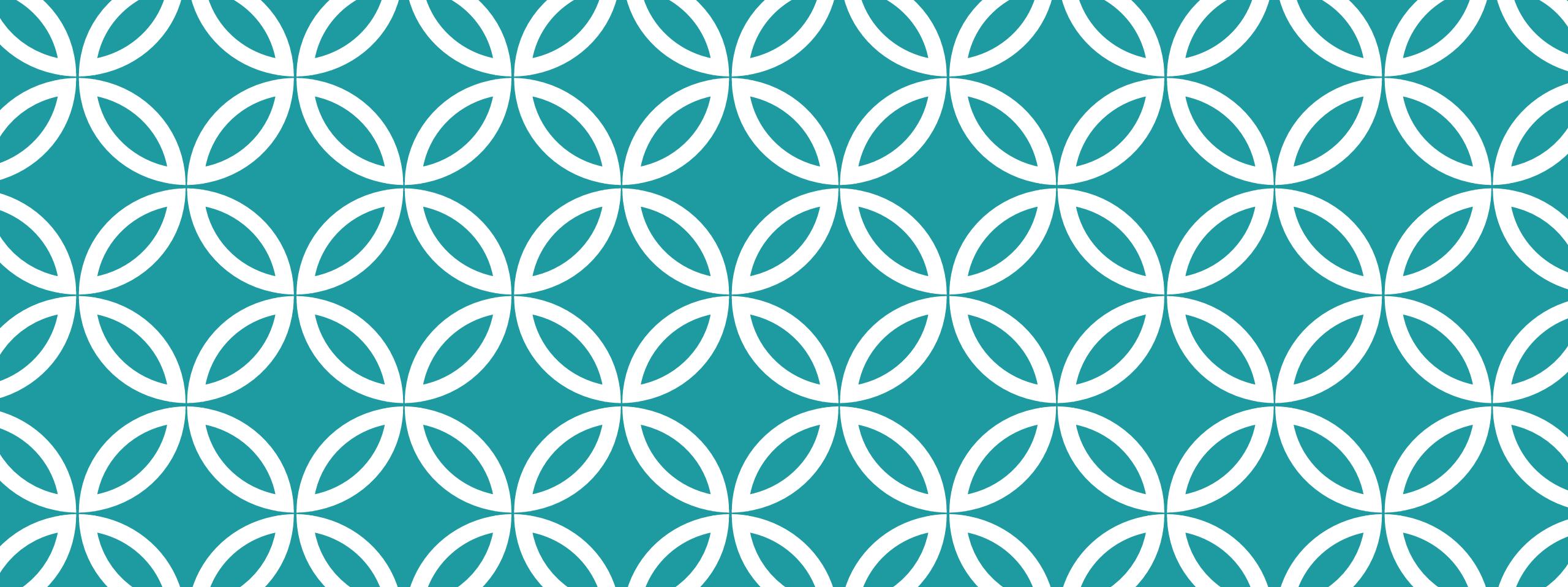
country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

pivot_wider(data, names_from = "name",
values_from = "value")

The inverse of pivot_longer(). "Widen" data by
expanding two columns into several. One column
provides the new column names, the other the
values.

```
pivot_wider(table2, names_from = type,  
values_from = count)
```

Look here to get more examples: <https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html>



NESTED VALUES

LIST-COLUMN DATA

Because of this characteristic, if you extract a single column from a tibble it will be still a **tibble** (not a vector!!!).

Therefore, you need to be careful when operating with single columns from tibbles – e.g. statistical functions that expect a vector input won't work on this kind of data unless you convert it to a vector.

Tibbles can store more information in their columns than a standard data.frame.

Columns in a tibble can store lists inside. These can be lists of vectors or lists of varying data types (even with tibbles inside → good use case for data grouping).

name	yr	lat	long
Amy	1975	27.5	-79.0
Amy	1975	28.5	-79.0
Amy	1975	29.5	-79.0
Bob	1979	22.0	-96.0
Bob	1979	22.5	-95.3
Bob	1979	23.0	-94.6
Zeta	2005	23.9	-35.6
Zeta	2005	24.2	-36.1
Zeta	2005	24.7	-36.6

name	yr	lat	long
Amy	1975	27.5	-79.0
Amy	1975	28.5	-79.0
Amy	1975	29.5	-79.0
Bob	1979	22.0	-96.0
Bob	1979	22.5	-95.3
Bob	1979	23.0	-94.6
Zeta	2005	23.9	-35.6
Zeta	2005	24.2	-36.1
Zeta	2005	24.7	-36.6

"cell" contents

yr	lat	long
1975	27.5	-79.0
1975	28.5	-79.0
1975	29.5	-79.0

nested data frame

name	data
Amy	<tibble [50x3]>
Bob	<tibble [50x3]>
Zeta	<tibble [50x3]>

yr	lat	long
1979	22.0	-96.0
1979	22.5	-95.3
1979	23.0	-94.6

yr	lat	long
2005	23.9	-35.6
2005	24.2	-36.1
2005	24.7	-36.6

Index list-columns with `[[[]]]`. `n_storms$data[[1]]`

USUALLY, YOU'LL NEED TO UNNEST YOUR DATA

unnest_longer(data, col, values_to = NULL, indices_to = NULL)
Turn each element of a list-column into a row.

```
starwars %>%  
  select(name, films) %>%  
  unnest_longer(films)
```

name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>



name	films
Luke	The Empire Strik...
Luke	Revenge of the S...
Luke	Return of the Jed...
C-3PO	The Empire Strik...
C-3PO	Attack of the Cl...
C-3PO	The Phantom M...
R2-D2	The Empire Strik...
R2-D2	Attack of the Cl...
R2-D2	The Phantom M...

unnest_wider(data, col) Turn each element of a list-column into a regular column.

```
starwars %>%  
  select(name, films) %>%  
  unnest_wider(films)
```

name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>

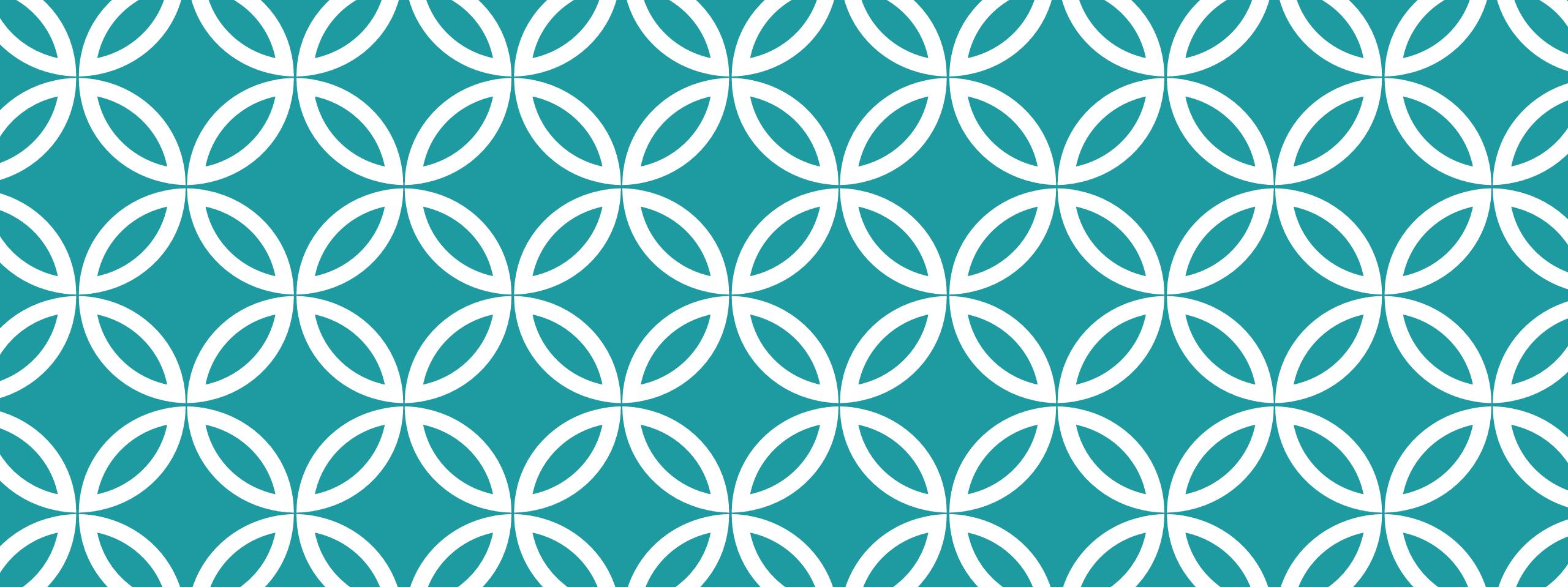
name	.1	.2	.3
Luke	The Empire...	Revenge of...	Return of...
C-3PO	The Empire...	Attack of...	The Phantom...
R2-D2	The Empire...	Attack of...	The Phantom...

hoist(.data, .col, ..., .remove = TRUE) Selectively pull list components out into their own top-level columns. Uses purrr::pluck() syntax for selecting from lists.

```
starwars %>%  
  select(name, films) %>%  
  hoist(films, first_film = 1, second_film = 2)
```

name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>

name	first_film	second_film	films
Luke	The Empire...	Revenge of...	<chr [3]>
C-3PO	The Empire...	Attack of...	<chr [4]>
R2-D2	The Empire...	Attack of...	<chr [5]>



HANDLE MISSING VALUES

TIDYR PROVIDES HANDY FUNCTIONS FOR MISSING DATA MANAGEMENTS

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(data, ...) Drop rows containing NA's in ... columns.
`drop_na(x, x2)`

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(data, ..., .direction = "down") Fill in NA's in ... columns using the next or previous value.
`fill(x, x2)`

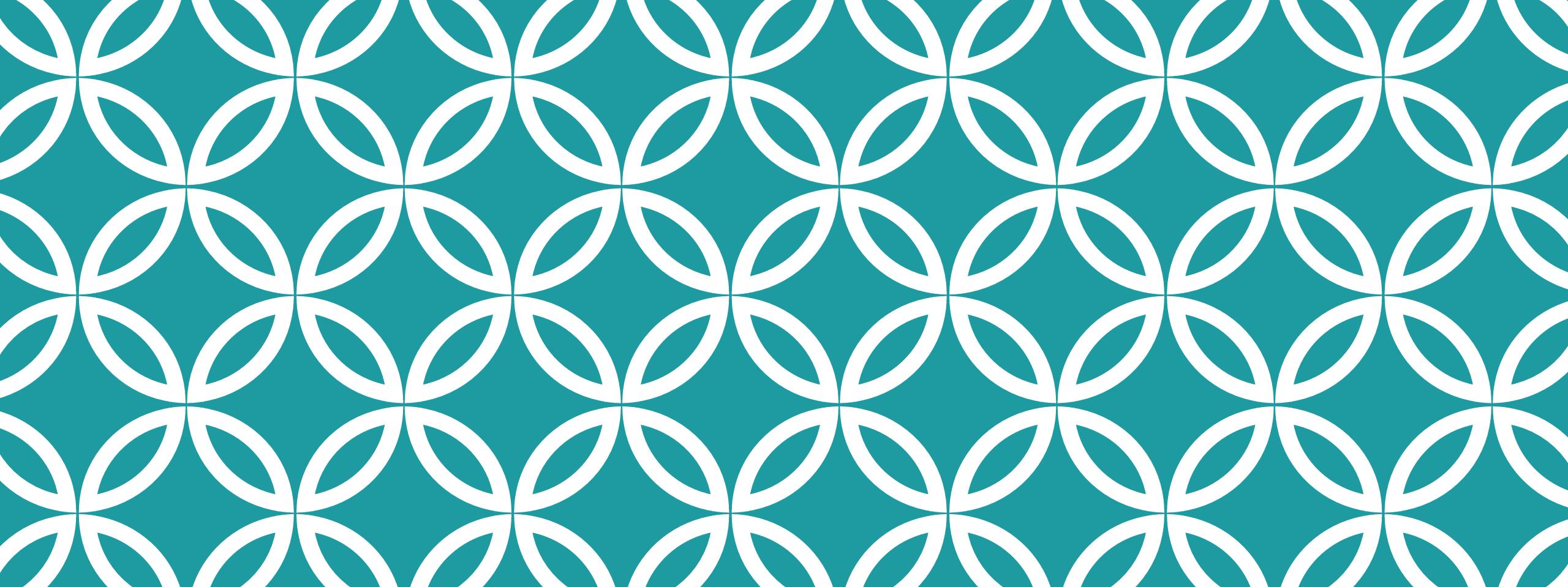
x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(data, replace) Specify a value to replace NA in selected columns.
`replace_na(x, list(x2 = 2))`



MORE DATA MODIFICATIONS

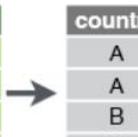
**IF YOU NEED TO
MERGE OR SPLIT
INFORMATION
BETWEEN
COLUMNS TRY
USING ONE OF
THESE FUNCTIONS**

Split Cells

- Use these functions to split or combine cells into individual, isolated values.

table5

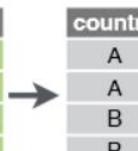
country	century	year
A	19	99
A	20	00
B	19	99
B	20	00



country	year
A	1999
A	2000
B	1999
B	2000

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M



country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M



country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M

unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE) Collapse cells across several columns into a single column.

```
unite(table5, century, year, col = "year", sep = "")
```

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...) Separate each cell in a column into several columns. Also **extract()**.

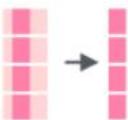
```
separate(table3, rate, sep = "/",  
        into = c("cases", "pop"))
```

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE) Separate each cell in a column into several rows.

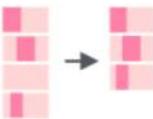
```
separate_rows(table3, rate, sep = "/")
```

IF YOU NEED TO WORK SPECIFICALLY WITH TEXT DATA TRY STRINGR FUNCTIONS

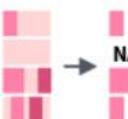
Subset Strings



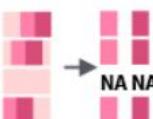
str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector.
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`



str_subset(string, pattern, negate = FALSE) Return only the strings that contain a pattern match. `str_subset(fruit, "p")`



str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also `str_extract_all()` to return every pattern match. `str_extract(fruit, "[aeiou]")`

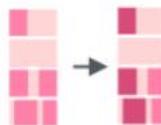


str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also `str_match_all()`.
`str_match(sentences, "(a|the) ([^ +])")`

Mutate Strings



str_sub() <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.
`str_sub(fruit, 1, 3) <- "str"`



str_replace(string, pattern, replacement) Replace the first matched pattern in each string. Also `str_remove()`.
`str_replace(fruit, "p", "-")`



str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. Also `str_remove_all()`.
`str_replace_all(fruit, "p", "-")`

A STRING
↓
a string

a string
↓
A STRING

a string
↓
A String

str_to_lower(string, locale = "en")¹ Convert strings to lower case.
`str_to_lower(sentences)`

str_to_upper(string, locale = "en")¹ Convert strings to upper case.
`str_to_upper(sentences)`

str_to_title(string, locale = "en")¹ Convert strings to title case. Also `str_to_sentence()`.
`str_to_title(sentences)`

WHEN STRUGGLING WITH FACTORS TRY FORCATS

Change the order of levels

a	1 = a
c	2 = b
b	3 = c

a	1 = b
c	2 = c
b	3 = a

fct_relevel(f, ..., after = 0L)
Manually reorder factor levels.
`fct_relevel(f, c("b", "c", "a"))`

c	1 = a
c	2 = c
a	

c	1 = c
c	2 = a
a	

fct_infreq(f, ordered = NA)
Reorder levels by the frequency
in which they appear in the
data (highest frequency first).
Also **fct_inseq()**.
`f3 <- factor(c("c", "c", "a"))`
`fct_infreq(f3)`

b	1 = a
a	2 = b

b	1 = b
a	2 = a

fct_inorder(f, ordered = NA)
Reorder levels by order in which
they appear in the data.
`fct_inorder(f2)`

Add or drop levels

a	1 = a
b	2 = b
x	

a	1 = a
b	2 = b
x	

fct_drop(f, only) Drop unused levels.
`f5 <- factor(c("a", "b"), c("a", "b", "x"))`
`f6 <- fct_drop(f5)`

a	1 = a
b	2 = b
x	

a	1 = a
b	2 = b
x	

fct_expand(f, ...) Add levels to a factor.
`fct_expand(f6, "x")`

a	1 = a
b	2 = b
NA	

a	1 = a
b	2 = b
x	

fct_explicit_na(f, na_level = "(Missing)")
Assigns a level to NAs to ensure they
appear in plots, etc.
`fct_explicit_na(factor(c("a", "b", NA)))`

Combine Factors

a	1 = a
c	2 = c

b	1 = a
a	2 = b
a	

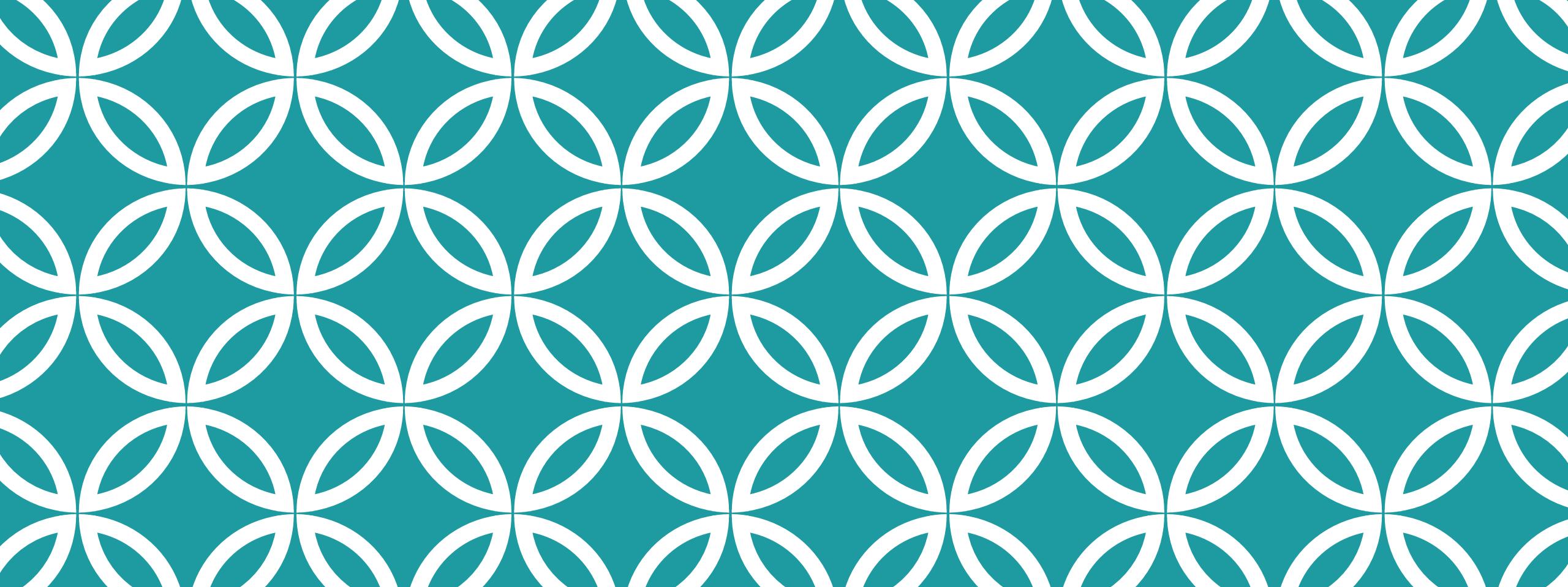
fct_c(...) Combine factors
with different levels.
Also **fct_cross()**.

`f1 <- factor(c("a", "c"))`
`f2 <- factor(c("b", "a"))`
`fct_c(f1, f2)`

a	1 = a
b	2 = b
a	1 = a

a	1 = a
b	2 = b
c	3 = c

fct_unify(fs, levels = lvls_union(fs)) Standardize
levels across a list of factors.
`fct_unify(list(f2, f1))`



**USE CHEAT SHEETS AND VIGNETTES
TO LEARN MORE**

Data tidying with `tidyr` :: CHEAT SHEET



Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:



Each variable is in its own column

&



&

Each observation, or case, is in its own row



Access variables as vectors



Preserve cases in vectorized operations

Tibbles



AN ENHANCED DATA FRAME

Tibbles are a table format provided by the `tibble` package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with `[`, a vector with `[[` and `$`.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

`options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf)` Control default display settings.

`View()` or `glimpse()` View the entire data set.

CONSTRUCT A TIBBLE

`tibble(...)` Construct by columns.

`tibble(x = 1:3, y = c("a", "b", "c"))`

Both make this tibble

`tribble(...)` Construct by rows.

`tribble(~x, ~y,`

1, "a",

2, "b",

3, "c")

```
A tibble: 3 x 2
#> #>   x     y
#> #>   <int> <chr>
#> #> 1     1     a
#> #> 2     2     b
#> #> 3     3     c
```

`as_tibble(x, ...)` Convert a data frame to a tibble.

`enframe(x, name = "name", value = "value")`

Convert a named vector to a tibble. Also `deframe()`.

`is_tibble(x)` Test whether x is a tibble.

Reshape Data

- Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K



country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T



country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

`pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE)`

"Lengthen" data by collapsing several columns into two. Column names move to a new `names_to` column and values to a new `values_to` column.

`pivot_longer(table4a, cols = 2:3, names_to = "year", values_to = "cases")`

`pivot_wider(data, names_from = "name", values_from = "value")`

The inverse of `pivot_longer()`. "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

`pivot_wider(table2, names_from = type, values_from = count)`

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

x	x1	x2	x3
A	1	3	
B	1	4	
B	2	3	

`expand(data, ...)` Create a new tibble with all possible combinations of the values of the variables listed in ...
Drop other variables.
`expand(mtcars, cyl, gear, carb)`

x1	x2	x3
A	1	3
B	1	4
B	2	3

`complete(data, ..., fill = list())` Add missing possible combinations of values of variables listed in ... Fill remaining variables with NA.
`complete(mtcars, cyl, gear, carb)`

Handle Missing Values

Drop or replace explicit missing values (NA).

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

`drop_na(data, ...)` Drop rows containing NA's in ... columns.
`drop_na(x, x2)`

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

`fill(data, ..., .direction = "down")` Fill in NA's in ... columns using the next or previous value.
`fill(x, x2)`

x1	x2
A	1
B	1
C	2
D	3
E	3

`replace_na(data, replace)` Specify a value to replace NA in selected columns.
`replace_na(x, list(x2 = 2))`

Nested Data

A **nested data frame** stores individual tables as a list-column of data frames within a larger organizing data frame. List-columns can also be lists of vectors or lists of varying data types.

Use a nested data frame to:

- Preserve relationships between observations and subsets of data. Preserve the type of the variables being nested (factors and datetimes aren't coerced to character).
- Manipulate many sub-tables at once with **purrr** functions like `map()`, `map2()`, or `pmap()` or with **dplyr** `rowwise()` grouping.



CREATE NESTED DATA

`nest(data, ...)` Moves groups of cells into a list-column of a data frame. Use alone or with `dplyr::group_by()`:

1. Group the data frame with `group_by()` and use `nest()` to move the groups into a list-column.

```
n_storms <- storms %>%
  group_by(name) %>%
  nest()
```

2. Use `nest(new_col = c(x, y))` to specify the columns to group using `dplyr::select()` syntax.

```
n_storms <- storms %>%
  nest(data = c(year:long))
```

name	yr	lat	long
Amy	1975	27.5	-79.0
Amy	1975	28.5	-79.0
Amy	1975	29.5	-79.0
Bob	1979	22.0	-96.0
Bob	1979	22.5	-95.3
Bob	1979	23.0	-94.6
Zeta	2005	23.9	-35.6
Zeta	2005	24.2	-36.1
Zeta	2005	24.7	-36.6

name	data
Amy	<tibble [5x3]>
Bob	<tibble [5x3]>
Zeta	<tibble [5x3]>

yr	lat	long
2005	23.9	-35.6
2005	24.2	-36.1
2005	24.7	-36.6

Index list-columns with `[[[]]]`. `n_storms$data[[1]]`

CREATE TIBBLES WITH LIST-COLUMNS

`tibble::tribble(...)` Makes list-columns when needed.

```
tribble(~max, ~seq,
       3, 1:3,
       4, 1:4,
       5, 1:5)
```

max	seq
3	<int [3]>
4	<int [4]>
5	<int [5]>

`tibble::tribble(...)` Saves list input as list-columns.

```
tribble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))
```

`tibble::enframe(x, name="name", value="value")`

Converts multi-level list to a tibble with list-cols.
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

OUTPUT LIST-COLUMNS FROM OTHER FUNCTIONS

`dplyr::mutate()`, `transmute()`, and `summarise()` will output list-columns if they return a list.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(q = list(quantile(mpg)))
```

RESHAPE NESTED DATA

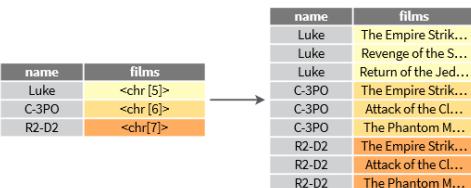
`unnest(data, cols, ..., keep_empty = FALSE)` Flatten nested columns back to regular columns. The inverse of `nest()`.

```
n_storms %>% unnest(data)
```

`unnest_longer(data, col, values_to = NULL, indices_to = NULL)`

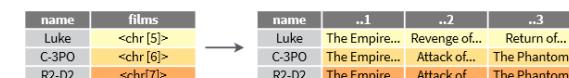
Turn each element of a list-column into a row.

```
starwars %>%
  select(name, films) %>%
  unnest_longer(films)
```



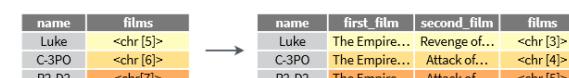
`unnest_wider(data, col)` Turn each element of a list-column into a regular column.

```
starwars %>%
  select(name, films) %>%
  unnest_wider(films)
```



`hoist(.data, .col, ..., .remove = TRUE)` Selectively pull list components out into their own top-level columns. Uses `purrr::pluck()` syntax for selecting from lists.

```
starwars %>%
  select(name, films) %>%
  hoist(films, first_film = 1, second_film = 2)
```



TRANSFORM NESTED DATA

A vectorized function takes a vector, transforms each element in parallel, and returns a vector of the same length. By themselves vectorized functions cannot work with lists, such as list-columns.

`dplyr::rowwise(.data, ...)` Group data so that each row is one group, and within the groups, elements of list-columns appear directly (accessed with `[[]]`, not as lists of length one. When you use `rowwise()`, `dplyr` functions will seem to apply functions to list-columns in a vectorized fashion.



Apply a function to a list-column and **create a new list-column**.

```
n_storms %>%
  rowwise() %>%
  mutate(n = list(dim(data)))
```

`dim()` returns two values per row
wrap with list to tell `mutate` to create a list-column

Apply a function to a list-column and **create a regular column**.

```
n_storms %>%
  rowwise() %>%
  mutate(n = nrow(data))
```

`nrow()` returns one integer per row

Collapse **multiple list-columns** into a single list-column.

```
starwars %>%
  rowwise() %>%
  mutate(transport = list(append(vehicles, starships)))
```

`append()` returns a list for each row, so col type must be list

Apply a function to **multiple list-columns**.

```
starwars %>%
  rowwise() %>%
  mutate(n_transports = length(c(vehicles, starships)))
```

`length()` returns one integer per row

See **purrr** package for more list functions.



String manipulation with stringr :: CHEAT SHEET

The `stringr` package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

<https://github.com/rstudio/cheatsheets/blob/main/strings.pdf>

Detect Matches

 → TRUE TRUE FALSE TRUE	str_detect(string, pattern, negate = FALSE) Detect the presence of a pattern match in a string. Also <code>str_like()</code> . <code>str_detect(fruit, "a")</code>
 → TRUE FALSE TRUE	str_starts(string, pattern, negate = FALSE) Detect the presence of a pattern match at the beginning of a string. Also <code>str_ends()</code> . <code>str_starts(fruit, "a")</code>
 → 2 4	str_which(string, pattern, negate = FALSE) Find the indexes of strings that contain a pattern match. <code>str_which(fruit, "a")</code>
 → 4 7 NA NA 3 4	str_locate(string, pattern) Locate the positions of pattern matches in a string. Also <code>str_locate_all()</code> . <code>str_locate(fruit, "a")</code>
 → 3 1 2	str_count(string, pattern) Count the number of matches in a string. <code>str_count(fruit, "a")</code>

Subset Strings

 → 1	str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. <code>str_sub(fruit, 1, 3); str_sub(fruit, -2)</code>
 → 1 2	str_subset(string, pattern, negate = FALSE) Return only the strings that contain a pattern match. <code>str_subset(fruit, "p")</code>
 → NA	str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also <code>str_extract_all()</code> to return every pattern match. <code>str_extract(fruit, "[aeiou]")</code>
 → NA NA	str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also <code>str_match_all()</code> . <code>str_match(sentences, "(a the) ([^+])")</code>

Manage Lengths

 → 6 2 3	str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). <code>str_length(fruit)</code>
 → P	str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. <code>str_pad(fruit, 17)</code>
 → P	str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. <code>str_trunc(sentences, 6)</code>
 → P	str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. <code>str_trim(str_pad(fruit, 17))</code>
 → P	str_squish(string) Trim whitespace from each end and collapse multiple spaces into single spaces. <code>str_squish(str_pad(fruit, 17, "both"))</code>

Mutate Strings

 ↓ a string	str_sub() <- value. Replace substrings by identifying the substrings with <code>str_sub()</code> and assigning into the results. <code>str_sub(fruit, 1, 3) <- "str"</code>
 ↓ a string	str_replace(string, pattern, replacement) Replace the first matched pattern in each string. Also <code>str_remove()</code> . <code>str_replace(fruit, "p", "-")</code>
 ↓ a string	str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. Also <code>str_remove_all()</code> . <code>str_replace_all(fruit, "p", "-")</code>
 ↓ a string	str_to_lower(string, locale = "en")¹ Convert strings to lower case. <code>str_to_lower(sentences)</code>
 ↓ a string	str_to_upper(string, locale = "en")¹ Convert strings to upper case. <code>str_to_upper(sentences)</code>
 ↓ A String	str_to_title(string, locale = "en")¹ Convert strings to title case. Also <code>str_to_sentence()</code> . <code>str_to_title(sentences)</code>

Join and Split

 → LETTERS	str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. <code>str_c(letters, LETTERS)</code>
 → fruit	str_flatten(string, collapse = "") Combines into a single string, separated by collapse. <code>str_flatten(fruit, "")</code>
 → sentences	str_dup(string, times) Repeat strings times times. Also <code>str_unique()</code> to remove duplicates. <code>str_dup(fruit, times = 2)</code>
 → H	str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also <code>str_split()</code> to return a list of substrings and <code>str_split_n()</code> to return the nth substring. <code>str_split_fixed(sentences, "", n=3)</code>
 → Pi	str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. <code>str_glue("Pi is {pi}")</code>
 → mtcars	str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. <code>str_glue_data(mtcars, "rownames(mtcars) has {hp} hp")</code>

Order Strings

 → 1 3 2	str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Return the vector of indexes that sorts a character vector. <code>fruit[str_order(fruit)]</code>
 → P	str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Sort a character vector. <code>str_sort(fruit)</code>

Helpers

 → apple banana pear	str_conv(string, encoding) Override the encoding of a string. <code>str_conv(fruit, "ISO-8859-1")</code>
 → sentences	str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches. Also <code>str_view()</code> to see only the first match. <code>str_view_all(sentences, "[aeiou]")</code>
 → TRUE FALSE TRUE	str_equal(x, y, locale = "en", ignore_case = FALSE, ...)¹ Determine if two strings are equivalent. <code>str_equal(c("a", "b"), c("a", "c"))</code>
 → This is a long sentence. ↓ This is a long sentence.	str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. <code>str_wrap(sentences, 20)</code>

¹ See bit.ly/ISO639-1 for a complete list of locales.

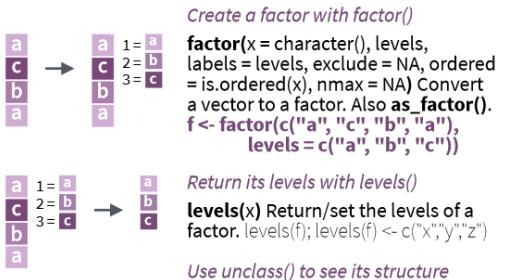
Factors withforcats :: CHEAT SHEET



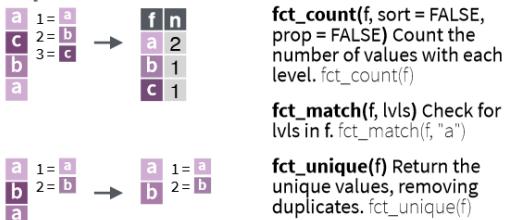
The `forcats` package provides tools for working with factors, which are R's data structure for categorical data.

Factors

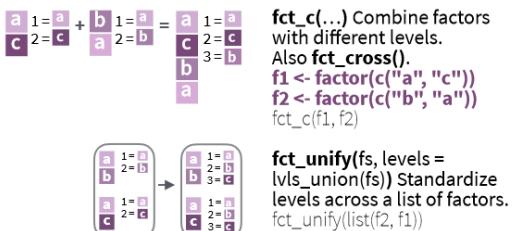
R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between integers and categorical values. When you view a factor, R displays not the integers, but the levels associated with them.



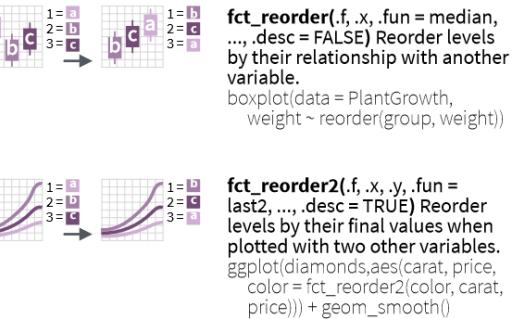
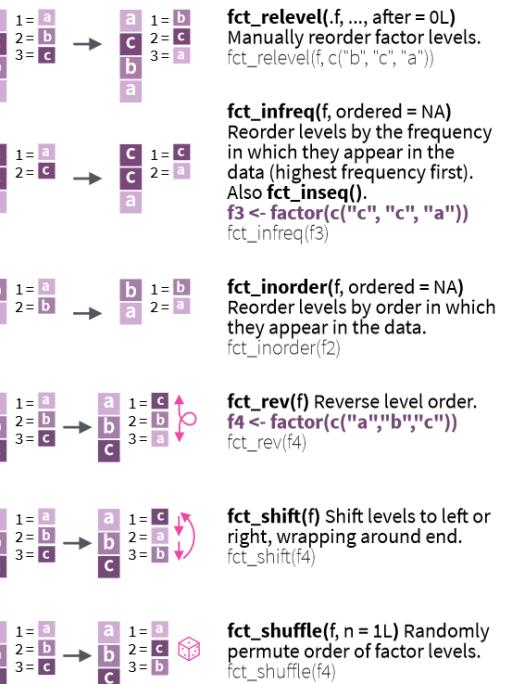
Inspect Factors



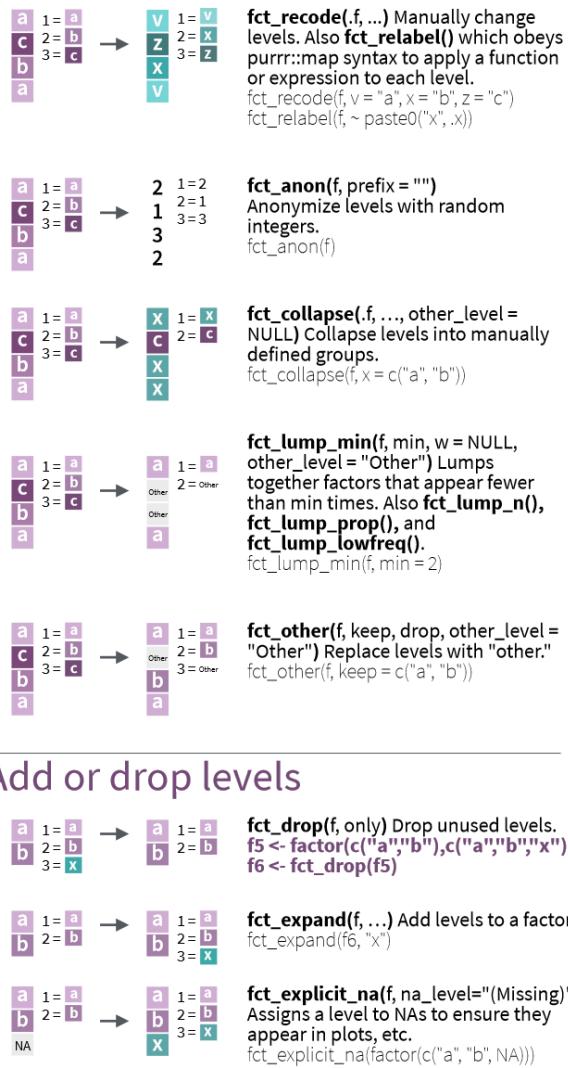
Combine Factors



Change the order of levels



Change the value of levels



Add or drop levels