

# Data tidying with tidyrr :: 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"))`

**tribble(...)** Construct by rows.

```
tribble(~x, ~y,
  1, "a",
  2, "b",
  3, "c")
```

Both make this tibble

```
A tibble: 3 x 2
  <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)`

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

tables

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	172M
B	2000	80K	174M

tables3

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

## 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)`

x

x1	x2	x3
A	1	3
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).

x

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)`

x

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)`

x

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

**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

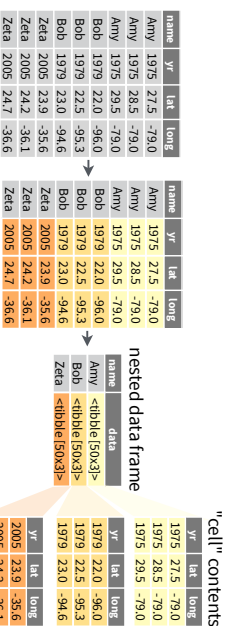
**nest(data, ...)** Moves groups of cells into a list-column of a data frame. Use alone or with **dplyr::group\_by()**:

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))
```



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)
```

**tibble::tibble(...)** Saves list input as list-columns.

```
tibble(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.

## OUTPUT LIST-COLUMNS FROM OTHER FUNCTIONS

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

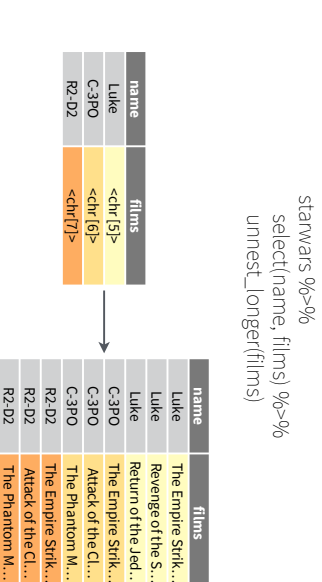
```
group_by(cyl) %>%
  summarise(q = list(quantile(mpg)))
```

## RESHAPE NESTED DATA

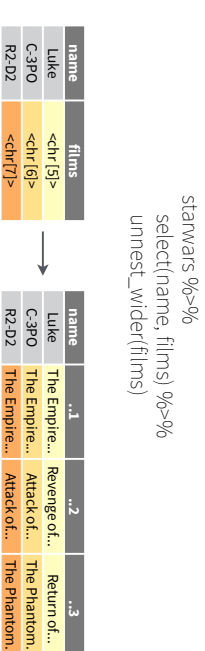
**unnest(data, cols, ..., keep\_empty = FALSE)** Flatten nested columns back to regular columns. The inverse of **nest()**.

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

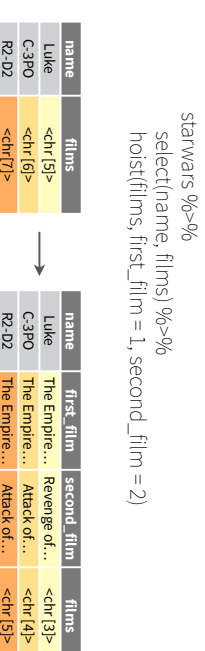
**unnest\_longer(data, col, values\_to = NULL, indices\_to = NULL)**  
Turn each element of a list-column into a row.



**unnest\_wider(data, col)** Turn each element of a list-column into a regular column.



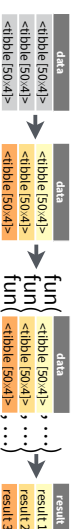
**hoist(data, col, ..., remove = TRUE)** Selectively pull list components out into their own top-level columns. Uses **purrr::pluck()** syntax for selecting from lists.



## 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**.



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



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



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



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

