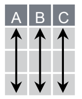
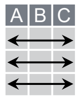
Data tidying with tidyr : : CHEATSHEET

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

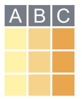


Each **variable** is in its own **column**

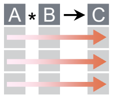
&



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 × 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

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)

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

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 = "")

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

separate_wider_delim(data, cols, delim, ..., names = NULL, names_sep = NULL, names_repair = "check unique", too_few, too_many, cols_remove = TRUE**)** Separate each cell in a column into several columns. Also **separate_wider_regex()** and **separate_wider_position().**

separate_wider_delim(table3, rate, delim = "/", into = c("cases", "pop"))

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

separate_longer_delim(data, cols, delim, .., width, keep_eampty**)** Separate each cell in a column into several rows.

separate_longer_delim(table3, rate, delim = "/")

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

→

x1	x2
A	1
A	2
B	1
B	2

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
B	1	4
B	2	3

→

x1	x2	x3
A	1	3
A	2	NA
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

→

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))

CC BY SA Posit Software, PBC • info@posit.co • posit.co • Learn more at **tidyr.tidyverse.org** • HTML cheatsheets at **pos.it/cheatsheets** • tidyr 1.3.1 • tibble 3.2.1 • Updated: 2024–05

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

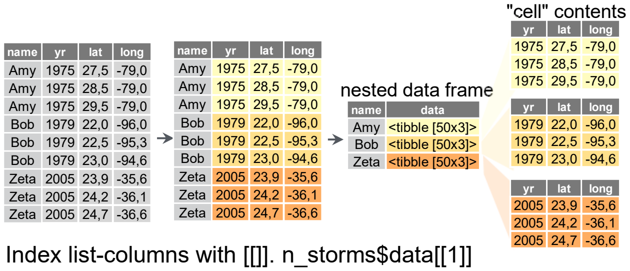
nest(data, …) Moves groups of cells into a list-column of a data frame. Use alone or with dplyr::**group_by()**:

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	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

→ nested data frame

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

"cell" contents

yr	lat	long
1975	27,5	-79,0
1975	28,5	-79,0
1975	29,5	-79,0

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]]

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::**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.
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.

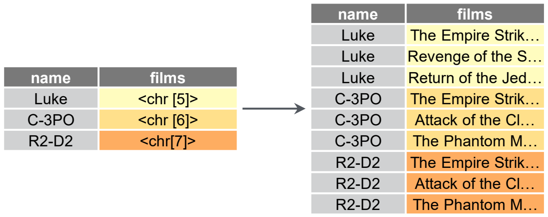
```
mtcars |>
  group_by(cyl) |>
  summarise(q = list(quantile(mpg)))
```

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)
```



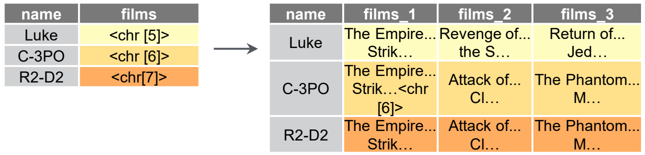
name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>

→

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, names_sep =
"_")
```



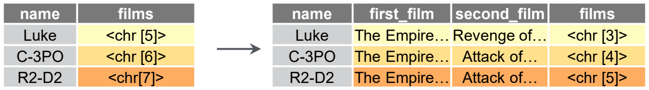
name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>

→

name	films_1	films_2	films_3
Luke	The Empire... Strik…	Revenge of... the S…	Return of... Jed…
C-3PO	The Empire... Strik…<chr [6]>	Attack of... Cl…	The Phantom... M…
R2-D2	The Empire... Strik…	Attack of... Cl…	The Phantom... M…

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]>

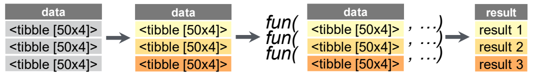
→

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]>

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.**



data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>

→

data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>

→

	data	
fun(	<tibble [50x4]>	, …)
fun(	<tibble [50x4]>	, …)
fun(	<tibble [50x4]>	, …)

→

result
result 1
result 2
result 3

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.