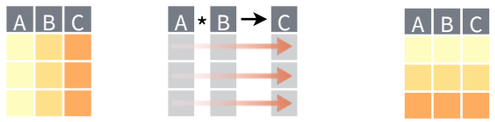
Tidy Data with tidyr : : **CHEAT SHEET**

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

Aggregate **cases** with **row-wise** operations.

Tibbles

AN ENHANCED DATA FRAME

The **tibble** package provides a class for storing tabular data, the tibble. They inherit the data frame class, but improve three 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 a 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. Useful for data where data or variables are stored in column names, rows contain multiple observations, and for summarizing or aggregating data.

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 pivoting column values into rows. 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

→

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	NA	NA

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

The inverse of pivot_longer, "widen" data by pivoting row data into columns. Pivot a names_from column and values_from column into their own new columns.

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

Split Cells

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

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

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"))

table5

country	century	year
Afghan	19	99
Afghan	20	00
Brazil	19	99
Brazil	20	00

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	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	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

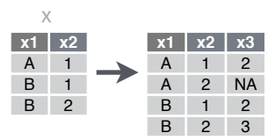
separate_rows(data, …, sep = "[^[:alnum:].]+", convert = FALSE) Separate each cell in a column into several rows.

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

Expand Tables

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

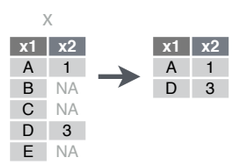
expand(data, …**)** Create a new tibble with all possible combinations of the values of the variables listed in … Also **expand_grid()** and **crossing()**.
expand(mtcars, cyl, gear, carb)



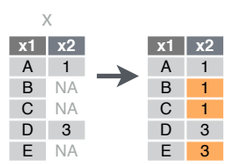
complete(data, …, fill = list()**)** Add missing possible combinations of values of variables listed in …
complete(mtcars, cyl, gear, carb)

Handle Missing Values

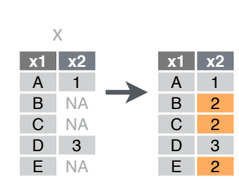
Drop or replace explicit missing values (NA).



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



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



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

RStudio

Nested Data

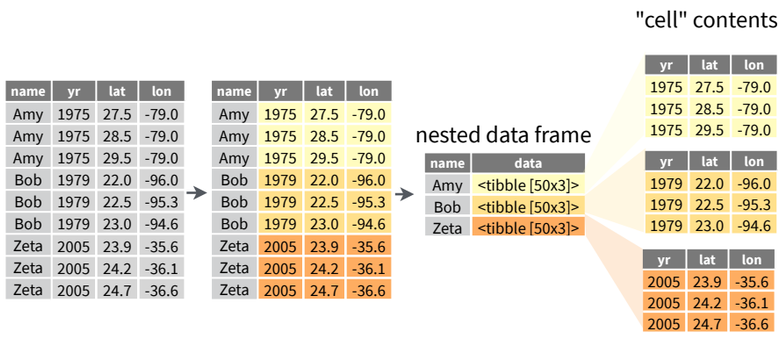
A **nested data frame** stores individual tables as a list-column of data frames within a larger organizing data frame. List-columns can also be lists of other atomic data types.
Use a nested data frame to:

- Preserve relationships between observations and subsets of data.
- Manipulate many sub-tables at once with **purrr** funcitons 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:hu_diameter))**



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

Create tibbles with list-columns

tibble::**tribble(**…**)** Makes list-column 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 tibble with list-cols
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

Output list-columns from other functions

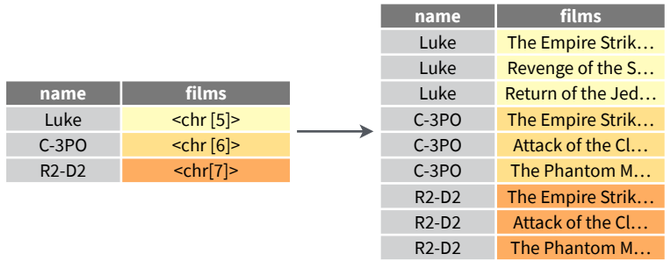
dplyr::**mutate(), transmute(),** and **summarise()** can output list-columns if they return a list.

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

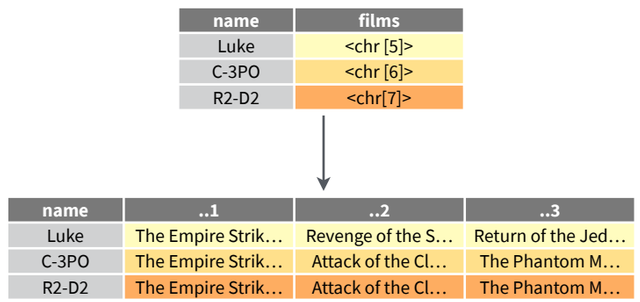
RESHAPE NESTED DATA

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)

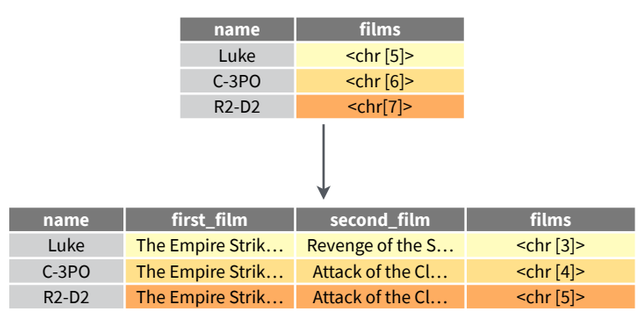


unnest_wider(data, col**)** Turn each element of a list-column into a column.
starwars %>% select(name, films) %>% unnest_wider(films)



hoist(.data, .col, ..., .remove = TRUE) Selectively pull list components 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

dplyr::**rowwise(**.data, …**)** Group data into individual rows. Use with dpylr functions apply functions to list-columns. See **c_across(**.cols**)** for column selection.

rowwise() can also be used to apply functions to list-columns.

1. Group data by row using **rowwise()**
2. Use dplyr::**mutate()** or dplyr::**transmute()** to compute a new variable that is the result of a function applied to each list in the specified list-columns. Use dplyr::**summarise()** to compute summary functions for each element of the list-column.

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

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

group row-wise · fun(columns) · new col name · dim() returns a vector, so col type must be list

Apply a function and **create a regular column**

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

nrow() returns an int, col type is int

Apply a function and **multiple list-columns**

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

Apply **summary functions**

```
seq_df <- tibble(x = c(3, 4, 5), y = list(1:3, 1:4, 1:5))
seq_df %>%
    rowwise() %>%
    summarise(avg = mean(y))
```

See **purrr** for more list functions.