

7.1 tidyr: Tidy Data and pivot_longer()

One variable per column, one observation per row. `pivot_longer()` turns wide columns into rows, and can strip a prefix, set a type, and drop missing cells.

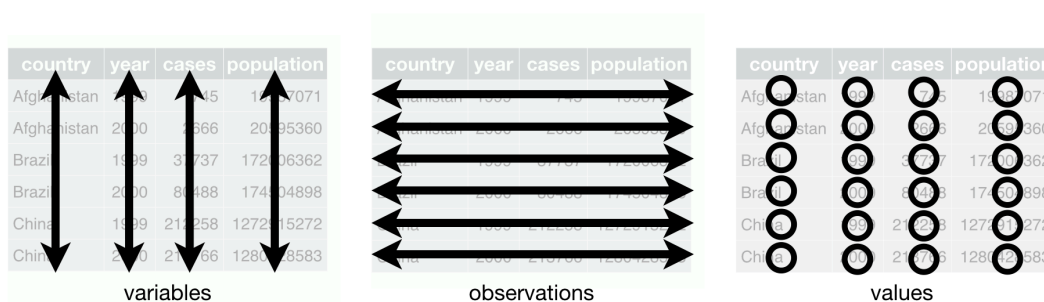
1. Tidy data

A table is **tidy** when:

1. each variable is a column
2. each observation is a row
3. each value is a cell

That is a shape, not a quality judgment. A tidy table can still have missing values, typos, and bad types. A **clean** table has those problems handled. You usually tidy first, then clean.

`tidyr` is the tidyverse package for the shape. It loads with `library(tidyverse)`.



```
library(tidyverse)
```

2. table1 is tidy; table4a is not

`tidyr` ships small WHO tuberculosis tables. `?table1`.

```
table1
table4a
```

`table1` has `country`, `year`, `cases`, and `population`. One year is one row. You can compute a rate without extra reshaping:

```
table1 |>
  mutate(rate = cases / population * 10000)
```

`table4a` stores **years as column names** (1999, 2000) and cases as the cells. Year is a variable, so those names are values in disguise. You cannot `filter(year == 1999)` until year is a column.

`table4b` is the same wide layout for population.

3. `pivot_longer()` makes columns into rows

Three arguments you will type every time:

- `cols` — which columns are values of one variable
- `names_to` — the new column that holds the old names
- `values_to` — the new column that holds the cells

```
table4a |>
  pivot_longer(
    cols = c(`1999`, `2000`),
    names_to = "year",
    values_to = "cases"
  )
```

The result has one row per country-year. `year` is character because it came from names. Section 6 turns that text into an integer inside the pivot.

`cols` uses the same helpers as `select()`. `cols = -country` is the usual shortcut when every other column should lengthen.

```
table4a |>
  pivot_longer(cols = -country, names_to = "year", values_to = "cases")
```

4. Numeric-looking names need backticks

1999 is a number. A bare number in `cols` is a **position**, not a name. Column 1999 does not exist.

This also fails for a second reason: without `c()`, 2000 is a separate argument, not part of `cols`.

```
table4a |>
  pivot_longer(cols = 1999, 2000, names_to = "year", values_to = "cases")
```

Write ``1999`` so R treats it as a name, and wrap the names in `c()`. The same backtick rule applies in `select()` and `mutate()`.

5. A small table with a prefix and missing cells

The next arguments are easier to see when the names share a prefix and some cells are `NA`.

```
sales <- tribble(
  ~region, ~FY2021, ~FY2022, ~FY2023,
  "East",   100L,   120L,   NA,
  "West",   80L,    NA,    95L,
  "North",  NA,     NA,     NA
)
```

`FY2021` is a year with the letters `FY` glued on. Year is the variable. `FY` is a label in the name.

6. `names_prefix` strips text from the start of each name

```
sales |>
  pivot_longer(
    cols = starts_with("FY"),
    names_to = "year",
    values_to = "revenue",
    names_prefix = "FY"
  )
```

`FY2021` becomes `"2021"`. `year` is still **character**. `names_prefix` is a regular expression, and it is matched only at the start of the name.

7. `names_transform` changes the type of the name column

Pass a named list of functions. The name in the list is the new column. The function runs after the prefix is removed.

```
sales |>
  pivot_longer(
    cols = starts_with("FY"),
    names_to = "year",
    values_to = "revenue",
    names_prefix = "FY",
    names_transform = list(year = as.integer)
  )
```

`year` is now an integer. `values_transform` does that job for the cells. `values_transform = list(revenue = as.numeric)` stores revenue as double.

8. `names_ptypes` and `values_ptypes` check the result

A **prototype** (the "ptype") is an empty vector that already has the type, class, and attributes you expect. `pivot_longer()` casts the new column to that template. If the cast fails, you get an error.

This call passes. `year` is character, and `revenue` is integer.

```
sales |>
  pivot_longer(
    cols = starts_with("FY"),
    names_to = "year",
    values_to = "revenue",
    names_prefix = "FY",
    names_ptypes = list(year = character()),
    values_ptypes = list(revenue = integer())
  )
```

This call fails. The year labels are still text, and a prototype of `integer()` cannot accept them.

```
sales |>
  pivot_longer(
    cols = starts_with("FY"),
    names_to = "year",
    values_to = "revenue",
    names_prefix = "FY",
    names_ptypes = list(year = integer())
  )
```

`names_transform` performs the conversion. `names_ptypes` requires the result to match a template.

9. An attribute is metadata on the vector

In R, an **attribute** is extra information attached to a vector, beyond the values themselves. The class is one attribute. Others you will meet:

Object	Attribute	Example
factor	levels	<code>c("low", "medium", "high")</code>
Date	class	<code>"Date"</code>
date-time	tzzone	<code>"America/New_York"</code>
time difference	units	<code>"days"</code>
matrix	dim	<code>c(2, 3)</code>
any vector	names	<code>c("a", "b")</code>

`attributes()` prints them.

```
f <- factor(c("low", "high"), levels = c("low", "medium", "high"))
attributes(f)
```

A prototype has to match the type and those attributes. This one asks for a factor whose levels are 2021, 2022, and 2023, in that order:

```
sales_years <- sales |>
  pivot_longer(
    cols = starts_with("FY"),
    names_to = "year",
    values_to = "revenue",
    names_prefix = "FY",
    names_ptypes = list(year = factor(levels = c("2021", "2022", "2023")))
  )
sales_years
levels(sales_years$year)
```

The cells print as 2021, and the column type is <fct>. levels() is the attribute the prototype required. That check is stricter than "the column should be a factor."

10. values_drop_na drops rows whose cell is NA

```
sales |>
  pivot_longer(
    cols = starts_with("FY"),
    names_to = "year",
    values_to = "revenue",
    names_prefix = "FY",
    values_drop_na = TRUE
  )
```

Four rows remain. North disappears, because every revenue cell for North was NA. East in 2023 and West in 2022 drop out for the same reason. Use this when a missing cell means that combination was never measured. Leave the NA in when the value is unknown but the row should stay.

11. Practice

Put course CSV files in a data/ folder of your project (see the Data section of the course page).

1. Load monkeymem.csv. The Week2 ... Week16 columns are weeks; the cells are accuracy. Pivot them longer so each row is one monkey-week.
2. On table4a, explain why pivot_longer(cols = 1999, 2000, names_to = "year", values_to = "cases") fails. Write the version that works.

3. Pivot `monkeymem.csv` again with `names_prefix = "Week"` and `names_transform` so the week column is an integer.
4. Rebuild `sales` from section 5. Pivot it with `values_drop_na = TRUE`. Which region disappears, and why? Then say why `names_ptypes = list(year = integer())` fails if you leave out `names_transform`.