

6. Grammar of Tables

Self-Paced Workshop #6

Prof. Jack Reilly

Fall 2026

A Grammar?

Like in regular English (i.e., language for humans), we use the word “grammar” in data programming (i.e., language for machines) to indicate a particular *way* of putting things together. In both places it means the same thing: not just *what* goes together, but the customary *way* things go together. We cannot simply mash pieces up in any order that seems logical. Grammar is *opinionated* (everyone has a grammar pedant in their life), and it is about what goes together in a way that reads as natural and persuasive, not only what parses.

The following sentence is perfectly good English:

The quick brown fox jumps over the lazy dog.

Yet something just *feels* off about this one to a native speaker:

The brown quick fox jumps over the lazy dog.

Likewise, it sounds strange to say any of the following:

- “bad big wolf”
- “tock-tick”
- “zag-zig”
- “roll and rock”
- “ends and odds”

Why? Not because any first-order rule of English forbids them, but because of customary rules of presentation: we are used to seeing things in a certain order, and our brains process them more smoothly that way.¹

Grammar for humans is about communication between humans. Grammar for machines is about instructing a machine so that what it produces is sensible and easily digestible *for*

¹If you want to dig deeper: these do follow real, if rarely taught, rules. English adjectives run opinion, size, age, shape, colour, origin, material, purpose, which is why “quick brown” sounds right and “brown quick” does not. Vowels in reduplicated pairs run *i*, then *a*, then *o*: tick-tock, zig-zag, ding-dong, flip-flop.

humans. It forces us to boil a task down to its basic parts and work through them in a logical, coherent order, and the payoff is a clean and clear presentation of our work.

Tables as Visualization

When we think of visualization we usually think of graphics, so much so that we treat the two words as nearly synonymous. But “visualization” means more than that. It means *communicating* what our data say to our audience as clearly and cleanly as possible, using position, order, and spatial relations on the page or screen. A table does exactly that.

In short, **a table is very often your first visualization.**² It takes complex data and distills it to its important parts in a way the reader can see and digest at a glance. You will almost always end up with a graphic as well, but do not overlook the value of a clean table. Tufte’s rule of thumb is that tables usually *outperform* graphics for small data sets of twenty numbers or fewer, and that they are the only right choice when the reader needs exact values.³ So, as with graphical design, we need to attend to good table design.

A Table Has Parts

The first thing most people do with a summary table is paste the raw printout into their write-up. But **a printout is not a table**. A *table* has named parts: a title, a column of row labels, headings over the columns, numbers formatted so they can be read, and notes about where they came from.

The `gt` package (for *grammar of tables*) builds a table from exactly those parts, one at a time, with a pipe between each. That is what “grammar” means here: a small vocabulary of pieces, piped together in the style of the tidyverse, with a rule for putting them together.⁴

When we manipulate a table, there are five things we are deciding, and every rule in the “grammar of tables” concerns exactly one of them:

1. **Structure.** What is a *row*, what is a *column*, and which of them are names rather than numbers. A reader scans down the left edge to find the case they care about and across the top to find the measure, so the labels that identify a row belong at the left and the measurements belong in the middle. Related columns sit next to each other under a shared heading; related rows are grouped.

²For examples and further discussion, see:

- Schwabish, Jonathan A. 2020. “Ten Guidelines for Better Tables.” *Journal of Benefit-Cost Analysis* 11(2): 151–178. doi:10.1017/bca.2020.11.
- Few, Stephen. 2012. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*, 2nd ed. Burlingame, CA: Analytics Press.

³Tufte, Edward R. 2001. *The Visual Display of Quantitative Information*, 2nd ed. Cheshire, CT: Graphics Press, pp. 56 and 178.

⁴Next week’s workshop does the same thing for graphs with `ggplot2`, and it is a much bigger vocabulary; this page is deliberately short so that the idea is clear before the vocabulary grows.

2. **Labels.** What each part is *called*. The title says what the table shows, not what it contains. Column headings are words a reader understands, with units, rather than the abbreviations that were convenient while computing. A good table can be read by someone who never saw the data.
3. **Format.** How each number *looks*: how many decimal places, whether there is a currency symbol or a percent sign, how dates and missing values are shown. Formatting changes only the appearance of a number, never its value, and it is where most false precision is removed.
4. **Annotation.** What the reader needs to know that is *not* a number: where the data came from, how a figure was computed, what an unfamiliar term means. A source line belongs at the bottom of every table; a footnote is attached to the specific cell, heading, or title it explains.
5. **Style.** Alignment, emphasis, rules, and colour. This is the layer people reach for first and should reach for last: the defaults are good, and every mark of emphasis competes with the data for the reader's eye.

Two ideas cut across all five. First, *location*: a table has a fixed anatomy (a header on top, row labels down the left, column headings across the top, a body of values, and a footer underneath), and every one of these decisions is applied to a particular part of that anatomy. That is what makes it a grammar rather than a list of options. Second, *order of operations*: the numbers themselves were settled before the table began. Last week's work, *manipulating data*, decided *what* to show. This week's work, *grammar*, decides *how* it is communicated.

Before you start

This page assumes [Self-Paced Workshop #5](#): the pipe, `filter()`, `group_by()`, `summarise()`, and `pivot_wider()` all appear in the first two steps without explanation. As before, the boxes share one R session — run them top to bottom, and if you see `object not found`, re-run the box above that creates it.

Step 1. Load the packages

Four packages. `dplyr` and `tidyr` build the summary table, `gt` dresses it, and `gapminder` is the data.⁵ The browser fetches them the first time you run the box, so give it a moment.

```
library(dplyr)
library(tidyr)
library(gt)
library(gapminder)
```

Step 2. A summary table to work with

`gapminder` is one row per country per year, 1952 to 2007, with life expectancy, population, and GDP per person:

⁵In your own R, `gt` and `gapminder` each need `install.packages()` once. `dplyr` and `tidyr` come with `library(tidyverse)`.

```
gapminder
```

The table we will build compares continents at the start and end of that span. First, we keep the first and last years, average within each continent and year, and then pivot the two years into side-by-side columns for a nice, clean layout:

```
by_cont <- gapminder |>
  filter(year %in% c(1952, 2007)) |>
  group_by(continent, year) |>
  summarise(life_exp = mean(lifeExp),
            gdp      = mean(gdpPercap),
            .groups  = "drop") |>
  pivot_wider(names_from = year,
              values_from = c(life_exp, gdp))
by_cont
```

Five rows, five columns.⁶ Everything in it is right, and nothing about it is ready to show anyone: the column names are code, the numbers run to six decimal places, and there is no title or source. Let's start fixing those.

Step 3. `gt()`: the table and its anatomy

Hand the data to `gt()` and you get a table:

```
#| output: asis
by_cont |>
  gt()
```

It already looks better than the printout, but all it has is a *body* (the numbers) and *column labels* (the names). Here is the full set of parts `gt` knows about, top to bottom:

Title and subtitle					<- header

stubhead	spanner		spanner		
	column	column	column	column	<- column labels

row label	. . .	body	. . .		
row label					<- stub (left) and body
row label					

footnotes					<- footer
source note					

⁶`pivot_wider()` glued the year onto each value column's name, so you have `life_exp_1952`, `life_exp_2007`, `gdp_1952`, and `gdp_2007`.

Each of the next five steps adds one of those parts. Every one is a function that takes a gt table on the left of the pipe and gives a gt table back, so the order mostly does not matter and you can stop anywhere.

Step 4. The stub: row labels

The continent names are not data to be compared; they are the *names of the rows*. Tell `gt()` which column is the stub, and give the stub a heading:

```
#| output: asis
by_cont |>
  gt(rownames_col = "continent") |>
  tab_stubhead(label = "Continent")
```

The continent column moves to the left, set apart from the numbers.⁷

Step 5. The header: title and subtitle

`tab_header()` puts a title on top. Write the title to say what the table *shows*, not what it *contains*:

```
#| output: asis
by_cont |>
  gt(rownames_col = "continent") |>
  tab_stubhead(label = "Continent") |>
  tab_header(title = "Life expectancy rose everywhere, income did not",
             subtitle = "Continental averages, 1952 and 2007")
```

Step 6. Column labels and spanners

Now the column names. `cols_label()` renames each column for display without touching the data, and `tab_spanner()` draws a heading *across* a group of columns. Because the year is now a spanner-level fact, the column labels themselves can be short:

```
#| output: asis
by_cont |>
  gt(rownames_col = "continent") |>
  tab_stubhead(label = "Continent") |>
  tab_header(title = "Life expectancy rose everywhere, income did not",
             subtitle = "Continental averages, 1952 and 2007") |>
  cols_label(life_exp_1952 = "1952", life_exp_2007 = "2007",
             gdp_1952 = "1952", gdp_2007 = "2007") |>
  tab_spanner(label = "Life expectancy (years)", columns = starts_with("life")) |>
  tab_spanner(label = "GDP per person", columns = starts_with("gdp"))
```

`starts_with()` is the same column-picking helper `select()` uses; `gt` borrows the whole

⁷With survey data the stub is usually a category — party, education, region — and the body is one or more statistics per category, exactly the shape `group_by() |> summarise()` produces.

`select()` vocabulary for its `columns =` arguments, so `c(gdp_1952, gdp_2007)` would have done the same thing.

Try it yourself: change the two spanner labels, and rename the stubhead to “Region”. Nothing in the data changes — only what the reader sees.

```
#| output: asis
by_cont |>
  gt(rownames_col = "continent") |>
  tab_stubhead(label = "Continent") |>
  cols_label(life_exp_1952 = "1952", life_exp_2007 = "2007",
             gdp_1952      = "1952", gdp_2007      = "2007") |>
  tab_spanner(label = "Life expectancy (years)", columns = starts_with("life")) |>
  tab_spanner(label = "GDP per person",          columns = starts_with("gdp"))
```

Step 7. Formatting the body

Six decimal places of life expectancy is false precision, and dollar values want a dollar sign. The `fmt_*`() family formats the *display* of a column, leaving the stored numbers alone — so you format last, and never round the data itself. Let’s save the table so far as an object first, since the pipeline is getting long:

```
#| output: asis
tbl <- by_cont |>
  gt(rownames_col = "continent") |>
  tab_stubhead(label = "Continent") |>
  tab_header(title = "Life expectancy rose everywhere, income did not",
             subtitle = "Continental averages, 1952 and 2007") |>
  cols_label(life_exp_1952 = "1952", life_exp_2007 = "2007",
             gdp_1952      = "1952", gdp_2007      = "2007") |>
  tab_spanner(label = "Life expectancy (years)", columns = starts_with("life")) |>
  tab_spanner(label = "GDP per person",          columns = starts_with("gdp"))

tbl |>
  fmt_number(columns = starts_with("life"), decimals = 1) |>
  fmt_currency(columns = starts_with("gdp"), decimals = 0)
```

A `gt` table is an ordinary R object, so `tbl` can be stored, passed along, and added to later like anything else.⁸

Try it yourself: show GDP with no decimals but *without* the dollar sign, and life expectancy with no decimals at all. You want `fmt_number()` for both — look at what `decimals = 0` does to the thousands separator.

⁸Other members of the family you will want: `fmt_percent()` for proportions (0.42 becomes 42%), `fmt_integer()` for counts with thousands separators, and `fmt_date()`. Each takes the same `columns =` and optional `rows =` arguments.

```
#| output: asis
tbl |>
  fmt_number(columns = starts_with("life"), decimals = 1) |>
  fmt_currency(columns = starts_with("gdp"), decimals = 0)
```

Step 8. The footer: source notes and footnotes

Every table in a write-up says where its numbers came from. `tab_source_note()` adds a line at the bottom; `tab_footnote()` attaches a marked note to a particular *location* in the table:

```
#| output: asis
tbl <- tbl |>
  fmt_number(columns = starts_with("life"), decimals = 1) |>
  fmt_currency(columns = starts_with("gdp"), decimals = 0) |>
  tab_source_note(source_note = "Source: Gapminder, via the gapminder R package.") |>
  tab_footnote(footnote = "Unweighted mean of the countries in each continent.",
               locations = cells_title(groups = "subtitle"))
tbl
```

That `locations =` argument is the other half of the grammar. Every part of the table has a `cells_*`() helper that points at it — `cells_title()`, `cells_column_labels()`, `cells_stub()`, `cells_body()` — and a footnote or a style goes wherever you point. For instance, to bold the single highest 2007 life expectancy:

```
#| output: asis
tbl |>
  tab_style(style = cell_text(weight = "bold"),
            locations = cells_body(columns = life_exp_2007,
                                   rows = life_exp_2007 == max(life_exp_2007)))
```

The `rows =` argument is a `filter()`-style question asked of the data, so the highlight follows the numbers if the data change.⁹

Try it yourself: add a second footnote to the “GDP per person” spanner saying the figures are in 2007 US dollars, adjusted for inflation. The location helper is `cells_column_spanners(spanners = "GDP per person")`.

```
#| output: asis
tbl # add a |> and a tab_footnote() here
```

Step 9. Exporting the table

In a Quarto document a `gt` table needs nothing special: put the code in a chunk and the rendered table appears in the output, in HTML and in PDF alike. That is the normal route

⁹Resist the urge to use this much. One highlighted cell directs the eye; a table full of bold and colour has no emphasis at all. The same goes for the many `opt_*`() and `tab_options()` settings `gt` offers — the defaults are good, and the assignments grade the *table*, not the decoration.

for the problem sets. If you need the table as a file, to drop into another program, say, `gtsave()` writes it in whatever format the file extension asks for. Here it is asked for `.html`, `.docx`, `.png`, and `.tex`:

```
gtsave(tbl, "continents.html")
gtsave(tbl, "continents.docx")
gtsave(tbl, "continents.png")
gtsave(tbl, "continents.tex")
```

(That will not run here, since the browser has no disk to write to, but it does in RStudio and Positron.)

Step 10. Transmitting table objects as text

Tables may be the first step of visualization, but they differ from graphics in one important way: a table is still *text*, not a picture. A graphic should go into your document as an image file. A table should not: a `.png` of a table cannot be searched, copied, read aloud, or resized without blurring, and its numbers are no longer numbers. We should prefer a route that keeps the table as text all the way to the page.

WHEN YOU KNOW THE FORMAT

In general, we like analysis scripts to save their figures to a folder from which the write-up pulls them in. We can use the same approach for tables; we just have to decide which format to write. If you know for sure what format your final document will be, you can use that format natively.

Quarto

Quarto is, at core, a simpler way of writing HTML (recall that HTML is Quarto's native output format), so a Quarto document can splice in an HTML file as text with the `include` shortcode. As usual, the path is relative to the `.qmd` file:

```
{{< include tables/continents.html >}}
```

Two cautions. First, `gtsave()` writes a complete web page, with its own `<html>` and `<head>` tags, rather than a bare table; for a fragment you can include, write `as_raw_html(tbl)` to the file instead. Second, raw HTML is dropped when Quarto renders to PDF, so this route works only for HTML output.

Other document editors

Likewise, other editors can take their native output: a LaTeX document can `\input{}` the `.tex` file, and a Word document can take the table from the `.docx` by copy and paste.

WHEN YOU WANT TO KEEP QUARTO'S FORMATS OPEN

If you want Quarto to keep rendering to whatever format you ask for, save the `gt object` itself rather than any rendering of it. Recall that `.rds` is R's native format for saving an object to be read back later. So, in the analysis script:

```
saveRDS(tbl, "tables/continents.rds")
```

And in the write-up, a chunk that reads it back and prints it:

```
readRDS("tables/continents.rds")
```

Because the write-up receives the table object, Quarto renders it natively for whichever format you ask for, HTML or PDF, and the analysis and the write-up stay in separate files that can each be re-run on their own.

A Moment of Sanity

The grammar of tables is a very *stylistic* way to make a very *nice* table, one that you have a lot of control over. It is also a very *complicated* way to make a table. In other words, you shouldn't plan on doing this every time you just want to see something in R; you build a table when you want to create something very clean and customized for a report, presentation, or paper.

Make it simple first. Then, if you want to keep it and present it professionally, make it using the grammar. Don't force yourself through the complicated table syntax unnecessarily.¹⁰

Work on Your Own

That is the whole grammar: a `gt()` table, a stub, a header, column labels and spanners, body formatting, a footer, and `cells_*`() locations to point at any of them. For practice, take any summary table in [Problem Set 6](#) or [Problem Set 7](#) and give it at least a `tab_header()`, a `fmt_*`(), and a `tab_source_note()` before it reaches the page. The [gt documentation](#) has a gallery of worked examples organised by exactly these parts, and the [gt cheatsheet](#) fits on one page.

¹⁰The complexity of the syntax, combined with the relative simplicity of the output and ease of validation, also makes tables like this *particularly* suited to AI assistance with the code. You can usually describe the table you want in words far more simply than in code. So don't hesitate to hand Claude the heavy lifting, so long as you validate the output (and see the course [AI policy](#)).