

7. Grammar of Graphics

Self-Paced Workshop #7

Prof. Jack Reilly

Fall 2026

A Graph Is Made of Layers

Last week's workshop built a table from parts. This week's builds a graph from *layers*, which is the idea behind `ggplot2` and the reason for its name: the `gg` is "grammar of graphics." While we have talked regularly about layers in graphs (and used them in base R as well as `ggplot`), we haven't yet introduced the *grammar*. Like with tables, the grammar is the particular way we express things so they make sense intuitively and come together cleanly for each graph.

The grammar says that every statistical graph, however elaborate, come from the same elements (layers):

Layer	What it decides	The function
data	which table	<code>ggplot(data)</code>
aesthetics	which columns become position, colour, size, shape	<code>aes()</code>
geoms	what mark to draw: point, line, bar, ...	<code>geom_*()</code>
stats	whether to compute something first: counts, bins, a fit	built into most geoms
scales	how values map to the page: axis range, log scale, colour palette	<code>scale_*()</code>
facets	whether to split into small multiples	<code>facet_wrap()</code>
coordinates	the coordinate system: flipped, polar, zoomed	<code>coord_*()</code>
labels and theme	titles, axis names, everything non-data	<code>labs()</code> , <code>theme_*()</code>

Once you can name the layers, any `ggplot2` code you meet is readable, and any graph you can describe is buildable. This page builds one layer at a time, with the smallest example of each.

Before you start

This page assumes the pipe and `filter()` from [Self-Paced Workshop #5](#), and uses the same `gapminder` data as [Self-Paced Workshop #6](#). The boxes share one R session; run them top to bottom. Graphs take a second or two to appear the first time.

Step 1. Load the packages

```
library(dplyr)
library(ggplot2)
library(gapminder)
```

In your own R, `library(tidyverse)` loads both `dplyr` and `ggplot2`; `gapminder` needs its own `install.packages("gapminder")` once.

Step 2. The data layer

`gapminder` is one row per country per year. For most of this page we want a single year, so make that table first:

```
gap07 <- gapminder |>
  filter(year == 2007)
gap07
```

142 countries, one row each. Now hand it to `ggplot()`:

```
ggplot(data = gap07)
```

A blank grey panel. That is correct: the data layer alone says *what* to plot but not *how*, so nothing is drawn.

Step 3. The aesthetics layer: `aes()`

An *aesthetic* is anything about a mark you can see — its horizontal position, its vertical position, its colour, its size, its shape. `aes()` *maps* columns of the data onto those. Map GDP per person to `x` and life expectancy to `y`:

```
ggplot(data = gap07, mapping = aes(x = gdpPercap, y = lifeExp))
```

Now there are axes, with sensible ranges read off the data, and still nothing drawn. Data plus aesthetics is a *frame*; it takes a *geom* to put marks in it. (People drop the `data =` and `mapping =` names and write `ggplot(gap07, aes(x = gdpPercap, y = lifeExp))`. We will too from here on.)

Step 4. The geom layer: `geom_*()`

A *geom* is the kind of mark. Add one with a plus sign:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point()
```

A scatterplot. Two things to notice about the code. First, layers are joined with `+`, not `|>`: `ggplot2` is older than the pipe, and the plus is the one place in the tidyverse where you *add* rather than pipe.¹ Second, the geom inherited `x` and `y` from `ggplot()`, so it needed no arguments of its own.

Layers stack. Add a second geom that draws a smooth trend through the same points:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  geom_smooth()
```

Try it yourself: swap `geom_point()` for `geom_line()` and look at the result. Then put `geom_point()` back and change `geom_smooth()` to `geom_smooth(method = "lm")`, which fits a straight line instead of a curve.²

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point()
```

Step 5. Mapping versus setting

There is a difference between making colour *depend on the data* and making everything *one* colour, and the grammar keeps them apart. Inside `aes()`, colour is *mapped* to a column, and `ggplot2` picks a palette and draws a legend:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +  
  geom_point()
```

Outside `aes()`, inside the geom, colour is *set* to a constant for every point:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point(colour = "steelblue")
```

The classic mistake is `aes(colour = "steelblue")`, which maps every point to a one-level variable whose only value is the word “steelblue” — try it, and look at the legend. Size, shape, and transparency (`alpha`) work the same way; mapping population to size is a graph you have probably seen before:³

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent, size = pop)) +  
  geom_point(alpha = 0.6)
```

¹The commonest `ggplot2` error is a `|>` where a `+` should be, or a `+` at the start of a line rather than the end of the one before. Either way R will tell you something is “not a function” or draw only part of the graph; look at the line endings.

²A line through these points is a poor description — it is pulled flat by the handful of very rich countries — which is the motivation for the log scale in Step 6.

³Hans Rosling’s bubble charts from Gapminder, which is where this dataset comes from.

Try it yourself: `alpha = 0.6` is *set*; make the points `shape = 17` (filled triangles) as a setting too, and then try mapping `shape = continent` instead and see how `ggplot2` handles two aesthetics for one variable.

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +  
  geom_point(alpha = 0.6)
```

Step 6. The scales layer: `scale_*()`

A *scale* controls how values turn into positions and colours. Every aesthetic has one already, chosen by default; a `scale_*()` function replaces it. Income is bunched near zero with a long right tail, so a log scale for `x` spreads the points out:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +  
  geom_point() +  
  scale_x_log10()
```

The relationship is nearly a straight line now, which was invisible before. The naming is systematic: `scale_` + the aesthetic + `_` + the kind of scale. The same scale can also format its labels, using the `scales` package that comes with `ggplot2`:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +  
  geom_point() +  
  scale_x_log10(labels = scales::label_dollar()) +  
  scale_colour_brewer(palette = "Set2")
```

Try it yourself: `scale_colour_brewer()` offers the ColorBrewer palettes; try "Dark2", then a *sequential* one like "Blues" and decide whether it suits a categorical variable. Then set `scale_y_continuous(limits = c(0, 90))` to make the vertical axis start at zero.

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +  
  geom_point() +  
  scale_x_log10(labels = scales::label_dollar()) +  
  scale_colour_brewer(palette = "Set2")
```

Step 7. The stats layer: geoms that compute

`geom_point()` draws the data as given. Many geoms *compute* something first and draw the result; that computation is the stats layer, and it is usually hidden inside the geom. A histogram bins one variable and counts the bins:

```
ggplot(gap07, aes(x = lifeExp)) +  
  geom_histogram(binwidth = 5)
```

Only an `x` aesthetic was supplied; the `y` is a count the geom made up. This is the univariate graph from this week's lecture, and `binwidth` is the one decision it needs from you. `geom_bar()` does the same with a categorical variable, counting rows per category:

```
ggplot(gap07, aes(x = continent)) +  
  geom_bar()
```

If you already have the counts — from `summarise()`, say — use `geom_col()`, which draws the bar heights you give it instead of counting:

```
gap07 |>  
  group_by(continent) |>  
  summarise(median_life = median(lifeExp)) |>  
  ggplot(aes(x = continent, y = median_life)) +  
  geom_col()
```

Notice the pipe *into* `ggplot()`: a dplyr pipeline that ends in a graph is the normal shape of a plotting script, and the data argument is filled in by the pipe.⁴

Try it yourself: change `binwidth = 5` to 1, then to 15, and decide which tells the truth about the distribution. Then replace `geom_histogram()` with `geom_density()`, which smooths rather than bins (drop the `binwidth` argument).

```
ggplot(gap07, aes(x = lifeExp)) +  
  geom_histogram(binwidth = 5)
```

Step 8. The facets layer: small multiples

Colour separates groups within one panel. *Facets* give each group its own panel, with shared axes so the panels can be compared:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  scale_x_log10() +  
  facet_wrap(~ continent)
```

The `~ continent` is R's formula notation, read "by continent." Facets are the honest way to show many groups at once: five overlapping colours on one panel is hard to read, five panels is not.

Try it yourself: facet the histogram from Step 7 by continent. Then add `ncol = 1` inside `facet_wrap()` to stack the panels, which lines up the horizontal axes for comparison.

```
ggplot(gap07, aes(x = lifeExp)) +  
  geom_histogram(binwidth = 5)
```

Step 9. The coordinates layer: `coord_*`

The default coordinate system is the ordinary Cartesian one, `x` across and `y` up. `coord_flip()` swaps them, which turns a bar chart on its side and makes long category labels readable:

⁴Watch the punctuation at the join. The pipeline uses `|>` up to and including the line that ends in `ggplot(...)`, then `+` from there on.

```
gap07 |>
  group_by(continent) |>
  summarise(median_life = median(lifeExp)) |>
  ggplot(aes(x = continent, y = median_life)) +
  geom_col() +
  coord_flip()
```

One subtlety worth knowing now. `coord_cartesian(ylim = c(60, 85))` *zooms* the graph; `scale_y_continuous(limits = c(60, 85))` *throws away* every row outside the range before anything is computed. For a bar chart or a smooth, that changes the answer, not just the view. Zoom with `coord_*`; filter with `scale_*` only when you mean it.

Try it yourself: add `coord_cartesian(ylim = c(60, 85))` to the box above and watch the bars get cropped. Then replace it with `scale_y_continuous(limits = c(60, 85))` — the bars that start at zero are now outside the limits, so they vanish.

```
gap07 |>
  group_by(continent) |>
  summarise(median_life = median(lifeExp)) |>
  ggplot(aes(x = continent, y = median_life)) +
  geom_col() +
  coord_flip()
```

Step 10. Labels and theme

Everything so far has been *data* ink. The last layer is everything else: titles, axis names, the legend title, the source line, and the look of the panel. `labs()` names things — the argument names are the aesthetics, plus title, subtitle, and caption — and a `theme_*()` function changes the look all at once:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +
  geom_point(alpha = 0.7) +
  scale_x_log10(labels = scales::label_dollar()) +
  labs(title = "Richer countries live longer, with diminishing returns",
       subtitle = "142 countries in 2007",
       x = "GDP per person (log scale)",
       y = "Life expectancy at birth (years)",
       colour = NULL,
       caption = "Source: Gapminder, via the gapminder R package.") +
  theme_minimal()
```

`colour = NULL` drops the legend title, since the continent names explain themselves. `theme()` itself adjusts individual pieces; moving the legend is the one you will use most:

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +
  geom_point(alpha = 0.7) +
  scale_x_log10(labels = scales::label_dollar()) +
```

```
labs(title = "Richer countries live longer, with diminishing returns",
      x     = "GDP per person (log scale)",
      y     = "Life expectancy at birth (years)",
      colour = NULL) +
theme_minimal() +
theme(legend.position = "bottom")
```

Try it yourself: `theme_minimal()` is one of several complete themes; try `theme_bw()`, `theme_classic()`, and `theme_light()`. Then pick one and keep it for every graph in a write-up — the point of a theme is consistency.⁵

```
ggplot(gap07, aes(x = gdpPercap, y = lifeExp, colour = continent)) +
  geom_point(alpha = 0.7) +
  scale_x_log10() +
  theme_minimal()
```

Step 11. Lines over time, and the group aesthetic

The assignments ask for trends over years, so one more geom. Pick a few countries from the *full* dataset and draw life expectancy against year:

```
gapminder |>
  filter(country %in% c("United States", "China", "Nigeria", "Japan")) |>
  ggplot(aes(x = year, y = lifeExp, colour = country)) +
  geom_line()
```

Four lines, one per country, because mapping colour to country also told `geom_line()` which points belong together. If you *don't* map colour — all lines the same — you have to say that explicitly with the group aesthetic, or the geom joins every point into one zigzag:

```
gapminder |>
  filter(continent == "Americas") |>
  ggplot(aes(x = year, y = lifeExp, group = country)) +
  geom_line(alpha = 0.4)
```

Try it yourself: delete `group = country` from the box above and run it, so you recognise the zigzag when it happens to you. Then put it back and add a second layer, `geom_smooth(aes(group = NULL), colour = "black")`, which fits one trend through all the countries at once.

```
gapminder |>
  filter(continent == "Americas") |>
  ggplot(aes(x = year, y = lifeExp, group = country)) +
  geom_line(alpha = 0.4)
```

⁵Rule of thumb for the assignments: a title that states the finding, axis labels in words with units, a caption with the source, and one theme throughout. Default axis labels like `gdpPercap` are a sign the graph was never finished.

Step 12. Saving a graph

In a Quarto document a graph needs nothing special: the chunk that draws it puts it in the output. To write it to a file, `ggsave()` saves the most recent plot, with the size in inches:

```
ggsave("life_vs_income.png", width = 7, height = 4.5)
```

(Nothing to write to here in the browser; the download button under each graph does the same job. In RStudio and Positron this is how figures get into a folder like `figures/` for a write-up to pull in, as the Week 6 in-class project does.)

Another Moment of Sanity

Like the grammar of tables, the grammar of graphics is a very *stylistic* way to make a very *nice* graphic, and one that you have a lot of control over. It can also be a rather *complicated* way to make a figure. In other words, you shouldn't plan on doing this every time you just want to see a figure in R. Make it using simple commands first (`lattice` is great for this also!). 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.⁶

Work on Your Own

Eight layers, each one function: `ggplot()` and `aes()`, a `geom_*`(), a `scale_*`(), `facet_wrap()`, a `coord_*`(), `labs()`, and a `theme_*`(). Every graph in [Problem Set 7](#) and [Practicum 1](#) is some subset of those stacked on a `dplyr` pipeline. Keep the [ggplot2 cheatsheet](#) open — it is organised by exactly these layers — and read [DV ch. 3](#), which builds the same gapminder graph with the same grammar and more patience.

⁶Also like tables: the complexity of the syntax and finicky-ness of the options, combined with the relative simplicity of the output and ease of validation, also makes the grammar of graphics *particularly* suited to AI assistance with the code. You can usually describe the figure 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](#)).