Principles of Data visualization

In the spring of 2021, nearly all of the American West was in a drought. By April of that year, officials in Southern California had declared a water emergency, citing unprecedented conditions.

This wouldn’t have come as news to those living in California and other Western states. Drought conditions like those in the West in 2021 are becoming increasingly common. Yet communicating the extent of problem remains difficult. How can we show the data in a way that accurately represents it while making it compelling enough to get people to take notice?

Data-visualization designers Cédric Scherer and Georgios Karamanis took on this challenge in the fall of that year. By working with the magazine Scientific American to create a data visualization of drought conditions over the last two decades in the United States, they turned to the ggplot2 package to transform what could have been dry data (pardon the pun) into a visually arresting and impactful graph.

This chapter explores why the data visualization that Scherer and Karamanis created is effective and introduces you to the grammar of graphics, a theory to make sense of graphs that underlies the ggplot2 package. You’ll then learn how to use ggplot2 by recreating the drought graph step by step. In the process, we’ll highlight some key principles of high-quality data visualization that you can use to improve your own work.

The Drought Visualization

Other news organizations had relied on the same data as Scherer and Karamanis, from the National Drought Center, in their stories. But Scherer and Karamanis visualized it in a way that it both grabs attention and communicates the scale of the phenomenon. Figure 2-1 shows a section of the final visualization. Covering four regions over the last two decades, the graph makes apparent increase in drought conditions, especially in California and the Southwest.

[F02001.pdf]



* + - * 1. A section of the final drought visualization, with a few tweaks made so that the plots fit in this book

To understand why this visualization is effective, let’s break it down into pieces. At the broadest level, the data visualization is notable for its minimalist aesthetic. There are, for example, no grid lines and few text labels, as well as little text along the axes. Scherer and Karamanis removed what statistician Edward Tufte, in his 1983 book The Visual Display of Quantitative Information, calls chartjunk. Tufte wrote that extraneous elements often hinder, rather than help, our understanding of charts (and researchers, as well as data visualization designers since, have generally agreed).

Need proof that Scherer and Karamanis’s decluttered graph is better than the alternative? Figure 2-2 shows a version with a few small tweaks to the code to include grid lines and text labels on axes. Prepare yourself for clutter!

[F02002.pdf]



* + - * 1. The cluttered version of the drought visualization

Again, it’s not just that this cluttered version looks worse. The clutter actively inhibits understanding. Rather than focus on overall drought patterns (the point of the graph), our brain gets stuck reading repetitive and unnecessary axis text.

One of the best ways to reduce clutter is to break a single chart into what are known as small multiples. When we look closely at the data visualization, we see that it is not one chart but actually a set of charts. Each rectangle represents one region in one year. If we filter it to show the Southwest region in 2003 and add axes titles, we can see in Figure 2-3 that the x axis shows the week while the y axis shows the percentage of that region at different drought levels.

[F02003.pdf]



* + - * 1. A drought visualization for the Southwest in 2003

Zooming in on a single region in a single year also makes the color choices more obvious. The lightest bars show the percentage of the region that is abnormally dry while the darkest bars show the percentage in exceptional drought conditions. These colors, as we’ll see shortly, were intentionally chosen to make differences in the drought levels visible to all readers.

Even so, the R code that Scherer and Karamanis wrote to produce this complex graph is relatively simple, due largely to a theory called the grammar of graphics.

The Grammar of Graphics

If you’ve used Excel to make graphs, you’re probably familiar with the menu shown in Figure 2-4. When working in Excel, your graph-making journey begins by selecting the type of graph you want to make. Want a bar chart? Click the bar chart icon. Want a line chart? Click the line chart icon.

[F02004.png]



* + - * 1. The Excel chart chooser menu

If you’ve only ever made data visualization in Excel, this first step may seem so obvious that you’ve never even considered the process of creating data visualization in any other way. But there are different models for thinking about graphs. Rather than conceptualizing graphs types as being distinct, we can recognize the things that they have in common and use these commonalities as the starting point for making them.

This approach to thinking about graphs comes from the late statistician Leland Wilkinson. For years, Wilkinson thought deeply about what data visualization is and how we can describe it. In 1999, he published a book called The Grammar of Graphics that sought to develop a consistent way of describing all graphs. In it, Wilkinson argued that we should think of plots not as distinct types à la Excel, but as following a grammar that we can use to describe any plot. Just as English grammar tells us that a noun is typically followed by a verb (which is why “he goes” works, while the opposite, “goes he,” does not), knowledge of the grammar of graphics allows us to understand why certain graph types “work.”

Thinking about data visualization through the lens of the grammar of graphics allow us to see, for example, that graphs typically have some data that is plotted on the x axis and other data that is plotted on the y axis. This is the case no matter whether the graph is a bar chart or a line chart, for example. Consider Figure 2-5, which shows two charts that use identical data on life expectancy in Afghanistan.

[F02005.pdf]



* + - * 1. A bar chart and a line chart showing identical data on Afghanistan life expectancy

While they look different (and would, to the Excel user, be different types of graphs), Wilkinson’s grammar of graphics allows us to see their similarities. (Incidentally, Wilkinson’s feelings on graph-making tools like Excel became clear when he wrote that “most charting packages channel user requests into a rigid array of chart types.”)

When Wilkinson wrote his book, no data visualization tool could implement his grammar of graphics. This would change in 2010, when Hadley Wickham announced the ggplot2 package for R in an article titled “A Layered Grammar of Graphics,” published in the Journal of Computational and Graphical Statistics. By providing the tools to implement Wilkinson’s ideas, ggplot2 would come to revolutionize the world of data visualization.

Working with ggplot2

The ggplot2 R package (which I, like nearly everyone in the data visualization world, will refer to simply as ggplot) relies on the idea of plots having multiple layers. Let’s walk through some of the most important ones. We’ll begin by selecting variables to map to aesthetic properties. Then we’ll choose a geometric object to use to represent our data. Next, we’ll change the aesthetic properties of our chart (its color scheme, for example) using a scale\_ function. Finally, we’ll use a theme\_ function to set the overall look-and-feel of our plot.

Mapping Data to Aesthetic Properties

When creating a graph with ggplot, we begin by mapping data to aesthetic properties. All this really means is that we use things like the x or y axis, color, and size (the so-called aesthetic properties) to represent variables. To make this concrete, we’ll use the data on life expectancy in Afghanistan, introduced in the previous section, to generate a plot. Access this data with the following code:

library(tidyverse)

gapminder\_10\_rows <- read\_csv("https://data.rwithoutstatistics.com/data/

gapminder\_10\_rows.csv")

We first load the tidyverse, as we did in Chapter 1, then use the read\_csv() function to access data from the book’s website and assign it to the object gapminder\_10\_rows.

Here is what the resulting gapminder\_10\_rows data frame looks like:

#> # A tibble: 10 × 6

#> country continent year lifeExp pop gdpPercap

#> <fct> <fct> <int> <dbl> <int> <dbl>

#> 1 Afghanistan Asia 1952 28.8 8425333 779.

#> 2 Afghanistan Asia 1957 30.3 9240934 821.

#> 3 Afghanistan Asia 1962 32.0 10267083 853.

#> 4 Afghanistan Asia 1967 34.0 11537966 836.

#> 5 Afghanistan Asia 1972 36.1 13079460 740.

#> 6 Afghanistan Asia 1977 38.4 14880372 786.

#> 7 Afghanistan Asia 1982 39.9 12881816 978.

#> 8 Afghanistan Asia 1987 40.8 13867957 852.

#> 9 Afghanistan Asia 1992 41.7 16317921 649.

#> 10 Afghanistan Asia 1997 41.8 22227415 635.

This is a shortened version of the full gapminder data frame, which includes over 1,700 rows of data.

If we want to make a chart with ggplot, we need to first decide which variable to put on the x axis and which to put on the y axis. For data showing change over time, it is common to put the date on the x axis and the value of what you are showing on the y axis. That means we would use the variable year on the x axis and the variable lifeExp on the y axis. To do so, we begin by using the ggplot() function:

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp

)

)

This function contains numerous arguments. As you can see, we place each argument on its own line, for the sake of readability, separating them with commas. First, we tell R that we’re using the data frame gapminder\_10\_rows. We also map year to the x axis and lifeExp to the y axis.

When we run the code, what we get in Figure 2-6 doesn’t look like much.

[F02006.pdf]



* + - * 1. A blank chart

If you look closely, however, you should see that the x axis corresponds to year and the y axis corresponds to lifeExp. Also, the values on the x and y axes match the scope of our data. In the gapminder\_10\_rows data frame, the first year is 1952 and the last year is 1997. The range of the x axis seems to have been created with this data in mind (because it was). Likewise, lifeExp, which goes from about 28 to about 42, will fit nicely on our y axis.

Choosing the Geometric Objects

Axes are nice, but we’re missing any type of visual representation of the data. To get this, we need to add the next ggplot layer: geoms. Short for geometric objects, geoms are functions that provide different ways of representing data. For example, if we want to add points to the graph, we use geom\_point():

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp

)

) +

geom\_point()

Now, in Figure 2-7, we see that people in 1952 had a life expectancy of about 28 and that this value rose every year included in the data.

[F02007.pdf]



* + - * 1. The same chart but with points added

Let’s say we change our mind and want to make a line chart instead. All we have to do is replace geom\_point() with geom\_line():

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp

)

) +

geom\_line()

Figure 2-8 shows the result.

[F02008.pdf]



* + - * 1. The data as a line chart

To really get fancy, what if we add both geom\_point() and geom\_line()?

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp

)

) +

geom\_point() +

geom\_line()

This code generates a line chart with points, as shown in Figure 2-9.

[F02009.pdf]



* + - * 1. The data with points and a line

We can swap in geom\_col() to create a bar chart:

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp

)

) +

geom\_col()

Note in Figure 2-10 that the y axis range has been automatically updated, going from 0 to 40 to account for the different geom.

[F02010.pdf]



* + - * 1. The data as a bar chart

As you can see, the difference between a line chart and a bar chart isn’t as great as the Excel chart-type picker might have us think. Both can have the same underlying properties (namely, putting years on the x axis and life expectancies on the y axis). They simply use different geometric objects to visually represent the data.

There are many geoms built into ggplot. In addition to geom\_bar(), geom\_point(), and geom\_line(), the following are among the most common:

geom\_histogram()

used to make histograms

geom\_boxplot()

used to make box plots

geom\_area()

used to make area charts

To see all geoms, visit the ggplot documentation website at https://ggplot2.tidyverse.org/reference/index.html#geoms.

Altering Aesthetic Properties

Before we return to the drought data visualization, let’s look at a few additional layers that can help us can alter the bar chart. Say we want to change the color of the bars. In the grammar of graphics approach to chart-making, this means mapping some variable to the aesthetic property of fill. (Slightly confusingly, the aesthetic property of color would, for a bar chart, change only the outline of each bar). In the same way that we mapped year to the x axis and lifeExp to the y axis, we can map fill to a variable, such as year:

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp,

fill = year

)

) +

geom\_col()

Figure 2-11 shows the result. We see now that, for earlier years, the fill is darker, while for later years, it is lighter (the legend, added to the right of our plot, also indicates this).

[F02011.pdf]



* + - * 1. The same chart, now with added colors

What if we wanted to change the fill colors? For that, we use a new scale layer. To do this, I’ll use the scale\_fill\_viridis\_c() function. The c at the end of the function name refers to the fact that the data is continuous, meaning it can take any numeric value:

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp,

fill = year

)

) +

geom\_col() +

scale\_fill\_viridis\_c()

This function changes the default palette to one that is colorblind-friendly and prints well in grayscale. The scale\_fill\_viridis\_c() function is just one of many that start with scale\_ and can alter the fill scale. Chapter 11 of the third edition of the book ggplot2: Elegant Graphics for Data Analysis (Springer, 2023) discusses various color and fill scales. You can read it online at https://ggplot2-book.org/scales-colour.html.

Setting a Theme

A final layer we’ll look at is the theme layer. This layer allows us to change the overall look-and-feel of plots (including the plot background, grid lines, and so on). Just as there are a number of scale\_ functions, there are also a number of functions that start with theme\_. Here, we’ve added theme\_minimal():

ggplot(

data = gapminder\_10\_rows,

mapping = aes(

x = year,

y = lifeExp,

fill = year

)

) +

geom\_col() +

scale\_fill\_viridis\_c() +

theme\_minimal()

Notice in Figure 2-12 that this theme starts to declutter the plot.

[F02012.pdf]



* + - * 1. The same chart with theme\_minimal() added

By now, you should see why Hadley Wickham described the ggplot2 package as using a layered grammar of graphics. It implements Wilkinson’s theory through the creation of multiple layers. First, we select variables to map to aesthetic properties, such as x or y axes, color, and fill. Second, we choose the geometric object (or geom) we want to use to represent our data. Third, if we want to change aesthetic properties (for example, to use a different color palette), we do this with a scale\_ function. Fourth, we use a theme\_ function to set the overall look and feel of the plot.

We could improve the plot we’ve been working on in many ways, but rather than add to an ugly plot, let’s instead return to the drought data visualization by Cédric Scherer and Georgios Karamanis. By walking through their code, you’ll learn lessons about making high-quality data visualization with ggplot and R.

Recreating the Drought Visualization

The drought visualization code relies on a combination of ggplot fundamentals and some less-well-known tweaks that make it really shine. To understand how Scherer and Karamanis made their data visualization, we’ll start with a simplified version of their code, then build it up layer by layer, adding elements as we go.

First, let’s import the data. Because it’s in JavaScript Object Notation (JSON) format, Scherer and Karamanis do this with the import() function from the rio package, which simplifies the process of importing JSON data:

library(rio)

dm\_perc\_cat\_hubs\_raw <- import("https://data.rwithoutstatistics.com/dm\_export\_20000101\_

20210909\_perc\_cat\_hubs.json"))

JSON is a common format for data used in web applications, though it’s far less common in R, where it can be complicated to work with. Luckily, the rio package simplifies its import into just one line.

Plotting One Region and Year

Scherer and Karamanis’s final plot consists of many years and regions. To help us see how they created it, let’s start by looking at just one region (the Southwest) in one year (2003). To do this, we need to create a data frame. We’ll use the filter() function twice: the first time to keep only data the Southwest region, and the second time to keep only data from 2003. Each time, we use the following syntax:

filter(variable\_name == value)

This tells R to keep only observations where the variable\_name is equal to some value. Our code starts with the dm\_perc\_cat\_hubs\_raw data frame before filtering it and then saving it as a new object called southwest\_2003:

southwest\_2003 <- dm\_perc\_cat\_hubs %>%

filter(hub == "Southwest") %>%

filter(year == 2003)

We can take a look at this object to see the variables we have to work with by entering southwest\_2003 in the console, which should return this:

#> # A tibble: 255 × 7  
#> date hub category percentage year week max\_week  
#> <date> <fct> <fct> <dbl> <dbl> <dbl> <dbl>  
#> 1 2003-12-30 Sout… D0 0.0718 2003 52 52  
#> 2 2003-12-30 Sout… D1 0.0828 2003 52 52  
#> 3 2003-12-30 Sout… D2 0.2693 2003 52 52  
#> 4 2003-12-30 Sout… D3 0.3108 2003 52 52  
#> 5 2003-12-30 Sout… D4 0.0796 2003 52 52  
#> 6 2003-12-23 Sout… D0 0.0823 2003 51 52  
#> 7 2003-12-23 Sout… D1 0.1312 2003 51 52  
#> 8 2003-12-23 Sout… D2 0.1886 2003 51 52  
#> 9 2003-12-23 Sout… D3 0.3822 2003 51 52  
#> 10 2003-12-23 Sout… D4 0.0828 2003 51 52  
#> # ℹ 245 more rows

The date variable represents the start date of the week in which the observation took place. The hub variable is the region, and category is the level of drought: a value of D0 indicates the lowest level of drought, while D5 indicates the highest level. The percentage variable is the percentage of that region in that drought category, ranging from 0 to 1. The year and week variables are the observation year and week number (beginning with week 1). The max\_week variable is the maximum number of weeks in a given year.

Now we can use this southwest\_2003 object for our plotting:

ggplot(

data = southwest\_2003,

aes(

x = week,

y = percentage,

fill = category

)

) +

geom\_col()

In the ggplot() function, we tell R to put week on the x axis and percentage on the y axis. We also use the category variable for the fill color. We then use geom\_col() to create a bar chart in which the fill color of each bar represents the percentage of the region in a single week at each drought level. You can see the result in in Figure 2-13.

[F02013.pdf]



* + - * 1. One year and region of the drought visualization

The colors don’t match the final version of the plot, but we can start to see the outlines of Scherer and Karamanis’s data visualization.

Changing Aesthetic Properties

Scherer and Karamanis next selected different fill colors for their bars. To do so, they used the scale\_fill\_viridis\_d() function. The d here means that the data to which the fill scale is being applied has discrete categories, called D0, D1, D2, D3, D4, and D5:

ggplot(

data = southwest\_2003,

aes(

x = week,

y = percentage,

fill = category

)

) +

geom\_col() +

scale\_fill\_viridis\_d(

option = "rocket",

direction = -1

)

They used the argument option = "rocket" to select the rocket palette, whose colors range from cream to nearly black. You could use several other palettes within the scale\_fill\_viridis\_d() function; see them at https://sjmgarnier.github.io/viridisLite/reference/viridis.html.

Then they used the direction = -1 argument to reverse the order of fill colors so that darker colors mean higher drought conditions.

Scherer and Karamanis also tweaked the appearance of the x and y axes:

ggplot(

data = southwest\_2003,

aes(

x = week,

y = percentage,

fill = category

)

) +

geom\_col() +

scale\_fill\_viridis\_d(

option = "rocket",

direction = -1

) +

scale\_x\_continuous(name = NULL,

guide = "none") +

scale\_y\_continuous(name = NULL,

labels = NULL,

position = "right")

On the x axis, they removed both the axis title (“week”) using name = NULL and the axis label text showing the week numbers 0 to 50 with guide = "none". On the y axis, they removed the title and text showing percentages using labels = NULL, which functionally does the same thing as guide = "none". They also moved the axis lines themselves to the right side using position = "right". These axis lines are apparent only as tick marks at this point but will become more visible later. Figure 2-14 shows the result of these tweaks.

[F02014.pdf]



* + - * 1. One year and region of the drought visualization with adjustments to the x and y axes

Up to this point, we’ve focused on one of the single plots that make up the larger data visualization. But the final product that Scherer and Karamanis made is actually 176 plots visualizing 22 years and eight regions. Let’s discuss the ggplot feature they used to create all of these plots.

Faceting the Plot

One of the most useful features of ggplot is what’s known as faceting (or, more commonly in the data visualization world, small multiples). Faceting takes a single plot and makes it into multiple plots using a variable. For example, think of a line chart showing life expectancy by country over time; instead of multiple lines on one plot, we might create multiple plots with one line per plot). With the facet\_grid() function, we can select which variable to put in the rows and which to put in the columns of our faceted plot:

dm\_perc\_cat\_hubs %>%

filter(hub %in% c("Northwest",

"California",

"Southwest",

"Northern Plains")) %>%

ggplot(aes(x = week,

y = percentage,

fill = category)) +

geom\_col() +

scale\_fill\_viridis\_d(

option = "rocket",

direction = -1

) +

scale\_x\_continuous(name = NULL,

guide = "none") +

scale\_y\_continuous(name = NULL,

labels = NULL,

position = "right") +

facet\_grid(rows = vars(year),

cols = vars(hub),

switch = "y")

Scherer and Karamanis put year in rows and hub (region) in columns. The switch = "y" argument moves the year label from the right side (where it appears by default) to the left. With this code in place, we can see the final plot coming together in Figure 2-15. Space considerations require me to include only four regions, but you get the idea.

[F02015.pdf]



* + - * 1. The faceted version of the drought visualization

Incredibly, the broad outlines of the plot took us just 10 lines to create. The rest of the code falls into the category of small polishes. That’s not to minimize how important small polishes are (very) or the time it takes to create them (lots). It does show, however, that a little bit of ggplot goes a long way.

Applying Small Polishes

Let’s look at a few of the small polishes that Scherer and Karamanis made. The first is to apply a theme, as shown in Figure 2-16. They used theme\_light(), which removes the default gray background and changes the font to Roboto using the base\_family argument.

The theme\_light() function is what’s known as a complete theme. So-called complete themes change the overall look-and-feel of a plot. The ggplot package has multiple complete themes that you can use (see them listed at <https://ggplot2.tidyverse.org/reference/index.html#themes>). Individuals and organizations also make their own themes, as we’ll do in Chapter 3. For a discussion of which themes you might consider using, see my blog post at https://rfortherestofus.com/2019/08/themes-to-improve-your-ggplot-figures.

Scherer and Karamanis didn’t stop by simply applying theme\_light(). They then used the theme() function to make additional tweaks to what theme\_light() gave them:

dm\_perc\_cat\_hubs %>%

filter(hub %in% c("Northwest",

"California",

"Southwest",

"Northern Plains")) %>%

ggplot(aes(x = week,

y = percentage,

fill = category)) +

geom\_col() +

scale\_fill\_viridis\_d(

option = "rocket",

direction = -1

) +

scale\_x\_continuous(name = NULL,

guide = "none") +

scale\_y\_continuous(name = NULL,

labels = NULL,

position = "right") +

facet\_grid(rows = vars(year),

cols = vars(hub),

switch = "y") +

theme\_light(base\_family = "Roboto") +

theme(

axis.title = element\_text(size = 14,

color = "black"),

axis.text = element\_text(family = "Roboto Mono",

size = 11),

1 axis.line.x = element\_blank(),

axis.line.y = element\_line(color = "black",

size = .2),

axis.ticks.y = element\_line(color = "black",

size = .2),

axis.ticks.length.y = unit(2, "mm"),

2 legend.position = "top",

legend.title = element\_text(color = "#2DAADA",

face = "bold"),

legend.text = element\_text(color = "#2DAADA"),

strip.text.x = element\_text(hjust = .5,

face = "plain",

color = "black",

margin = margin(t = 20, b = 5)),

strip.text.y.left = element\_text(3 angle = 0,

vjust = .5,

face = "plain",

color = "black"),

strip.background = element\_rect(fill = "transparent",

color = "transparent"),

panel.grid.minor = 4 element\_blank(),

panel.grid.major = element\_blank(),

panel.spacing.x = unit(0.3, "lines"),

panel.spacing.y = unit(0.25, "lines"),

5 panel.background = element\_rect(fill = "transparent",

color = "transparent"),

panel.border = element\_rect(color = "transparent",

size = 0),

plot.background = element\_rect(fill = "transparent",

color = "transparent",

size = .4),

plot.margin = margin(rep(18, 4))

)

The code in the theme() function does many different things, but let’s take a look at a few of the most important. First, it moves the legend from the right side (the default) to the top of the plot 2. Then, an angle = 0 argument rotates the year text in the columns so that it is no longer angled 3. Without this argument, the years would be much less readable.

Next, the theme() function makes the distinctive axis lines and ticks that show up on the right side of the final plot 1. Calling element\_blank() removes all grid lines 4. Finally, three lines remove the borders and make each of the individual plots have a transparent background 5.

Keen readers such as yourself may now be thinking, “Wait. Didn’t the individual plots have a gray background behind them?” Yes, dear reader, they did. Scherer and Karamanis made these with a separate geom, geom\_rect():

geom\_rect(

aes(

xmin = .5,

xmax = max\_week + .5,

ymin = -0.005,

ymax = 1

),

fill = "#f4f4f9",

color = NA,

size = 0.4

)

They set some additional aesthetic properties specific to this geom: xmin, xmax, ymin, and ymax, which determine the boundaries of the rectangle it produces. The result is a gray background drawn behind each small multiple, as shown in Figure 2-16.

[F02016.pdf]



* + - * 1. Faceted version of the drought visualization with gray backgrounds behind each small multiple

Finally, consider the tweaks made to the legend. We previously saw a simplified version of the scale\_fill\_viridis\_d() function. Here is a more complete version:

scale\_fill\_viridis\_d(

option = "rocket",

direction = -1,

name = "Category:",

labels = c(

"Abnormally Dry",

"Moderate Drought",

"Severe Drought",

"Extreme Drought",

"Exceptional Drought"

)

)

The name argument sets the legend title, and the labels argument determines the labels that show up in the legend. Figure 2-17 shows the result of these changes.

[F02017.pdf]



* + - * 1. Drought visualization with changes made to the legend text

Rather than D0, D1, D2, D3, and D4, we now have the legend text Abnormally Dry, Moderate Drought, Severe Drought, Extreme Drought, and Exceptional Drought.

The Complete Visualization Code

While I’ve showed you a nearly complete version of the code that Scherer and Karamanis wrote, I made some small changes to make it easier to understand. If you’re curious, the full code is here:

ggplot(dm\_perc\_cat\_hubs, aes(week, percentage)) +

geom\_rect(

aes(

xmin = .5,

xmax = max\_week + .5,

ymin = -0.005,

ymax = 1

),

fill = "#f4f4f9",

color = NA,

size = 0.4,

show.legend = FALSE

) +

geom\_col(

aes(

fill = category,

fill = after\_scale(addmix(darken(fill, .05,

space = "HLS"),

"#d8005a",

.15)),

color = after\_scale(darken(fill, .2,

space = "HLS"))

),

width = .9,

size = 0.12

) +

facet\_grid(rows = vars(year),

cols = vars(hub),

switch = "y") +

coord\_cartesian(clip = "off") +

scale\_x\_continuous(expand = c(.02, .02),

guide = "none",

name = NULL) +

scale\_y\_continuous(expand = c(0, 0),

position = "right",

labels = NULL,

name = NULL) +

scale\_fill\_viridis\_d(

option = "rocket",

name = "Category:",

direction = -1,

begin = .17,

end = .97,

labels = c(

"Abnormally Dry",

"Moderate Drought",

"Severe Drought",

"Extreme Drought",

"Exceptional Drought"

)

) +

guides(fill = guide\_legend(nrow = 2,

override.aes = list(size = 1))) +

theme\_light(base\_size = 18,

base\_family = "Roboto") +

theme(

axis.title = element\_text(size = 14,

color = "black"),

axis.text = element\_text(family = "Roboto Mono",

size = 11),

axis.line.x = element\_blank(),

axis.line.y = element\_line(color = "black",

size = .2),

axis.ticks.y = element\_line(color = "black",

size = .2),

axis.ticks.length.y = unit(2, "mm"),

legend.position = "top",

legend.title = element\_text(color = "#2DAADA",

size = 18,

face = "bold"),

legend.text = element\_text(color = "#2DAADA",

size = 16),

strip.text.x = element\_text(size = 16,

hjust = .5,

face = "plain",

color = "black",

margin = margin(t = 20, b = 5)),

strip.text.y.left = element\_text(size = 18,

angle = 0,

vjust = .5,

face = "plain",

color = "black"),

strip.background = element\_rect(fill = "transparent",

color = "transparent"),

panel.grid.minor = element\_blank(),

panel.grid.major = element\_blank(),

panel.spacing.x = unit(0.3, "lines"),

panel.spacing.y = unit(0.25, "lines"),

panel.background = element\_rect(fill = "transparent",

color = "transparent"),

panel.border = element\_rect(color = "transparent",

size = 0),

plot.background = element\_rect(fill = "transparent",

color = "transparent",

size = .4),

plot.margin = margin(rep(18, 4))

)

There are a few additional tweaks to colors and spacing, but most of the code reflects what you’ve seen so far.

Conclusion

You may start to think of ggplot as a solution to all of your data visualization problems. And yes, you have a new hammer, but no, everything is not a nail. If you look at the version of the data visualization that appeared in Scientific American in November 2021, you’ll see that some of its annotations aren’t visible in our recreation. That’s because they were added in post-production. While you could have found ways to create them in ggplot, it’s often not the best use of your time. Get yourself 90 percent of the way there with ggplot and then use Illustrator, Figma, or a similar tool to finish your work.

Even so, ggplot is a very powerful hammer, used to make plots that you’ve seen in The New York Times, FiveThirtyEight, the BBC, and other well-known news outlets. Although not the only tool that can generate high-quality data visualization, it makes the process straightforward. The graph by Scherer and Karamanis shows this in several ways:

It strips away extraneous elements, such as grid lines, to keep the focus on the data itself. Complete themes such as theme\_light() and the theme() function allowed Scherer and Karamanis to create a decluttered visualization that communicates effectively.

It uses well-chosen colors. The scale\_fill\_viridis\_d() function allowed them to create a color scheme that demonstrates differences between groups, is colorblind friendly, and shows up well when printed in grayscale.

It uses small multiples to break data from two decades and eight regions into a set of graphs that come together to create a single plot. With a single call to the facet\_grid() function, Scherer and Karamanis created over 100 small multiples that the tool automatically combined into a single plot.

Learning to create data visualization in ggplot involves a significant time investment. But the long-term payoff is even greater. Once you learn how ggplot works, you can look at others’ code and learn how to improve your own. By contrast, when you make a data visualization in Excel, the series of point-and-click steps disappears into the ether. To recreate a visualization you made last week, you’ll need to remember the exact steps you used, and to make someone else’s data visualization, you’ll need them to write up their process for you.

Because code-based data visualization tools allow you to keep that record of the steps you made, you don’t have to be the most talented designer to make high-quality data visualization with ggplot. You can study others’ code, adapt it to your own needs, and create your own data visualization that is beautiful and communicates effectively.

Learn More

Consult the following resources to learn more about data visualization principles and the ggplot2 package:

Data Visualization: A Practical Introduction by Kieran Healy (Princeton University Press, 2018), <https://socviz.co>

Fundamentals of Data Visualization by Claus Wilke (O'Reilly Media, 2019). <https://clauswilke.com/dataviz/>

ggplot2: Elegant Graphics for Data Analysis by Hadley Wickham, Danielle Navarro, and Thomas Lin Pedersen (Springer, Forthcoming), <https://ggplot2-book.org>

Graphic Design with ggplot2 by Cédric Scherer (CRC Press, Forthcoming)

The Glamour of Graphics course by Will Chase, <https://rfortherestofus.com/courses/glamour/>