Custom Data Visualization Themes

A custom theme is nothing more than a chunk of code that applies a set of small tweaks to all plots. So much of the work involved in making a professional chart consists of these kinds of adjustments. What font should you use? Where should the legend go? Should axes have titles? Should charts have grid lines? These questions may seem minor, but they have a major impact on the final product.

In 2018, BBC data journalists Nassos Stylianou and Clara Guibourg, along with their team, developed a custom ggplot theme that matches the BBC’s style. By introducing this bbplot package for others to use, they changed their organization’s culture, removed bottlenecks, and allowed the BBC to visualize data more creatively.

Rather than forcing everyone to copy the long code to tweak each plot they make, custom themes enable everyone who uses them to follow style guidelines and ensure that all data visualizations meet a brand’s standards. For example, to understand the significance of the custom theme introduced at the BBC, it’s helpful to know how things worked before bbplot. In the mid-2010s, journalists who wanted to make data visualization had two choices:

* Using an internal tool that could create data visualizations but was limited to the predefined charts it had been designed to generate.
* Using Excel to create mockups and then working with a graphic designer to finalize the charts. This approach led to better results and was much more flexible, but it required extensive, time-consuming back-and-forth with a designer.

Neither of these choices was ideal, and the BBC’s data visualization output was limited. R freed the journalists from having to work with a designer. It wasn’t that the designers were bad (they weren’t), but ggplot allowed the journalists to explore different visualizations on their own. As the team improved their ggplot skills, they realized that it might be possible to produce more than just exploratory data visualizations and to create production-ready charts in R that could go straight onto the BBC website.

This chapter discusses the power of custom ggplot themes, then walks through the code in the bbplot package to demonstrate how custom themes work. You’ll learn how to consolidate your styling code into a reusable function and how to consistently modify your plots’ text, axes, grid lines, background, and other elements.

Styling a Plot with a Custom Theme

The bbplot package has two functions: bbc\_style() and finalise\_plot(). The latter deals with tasks like adding the BBC logo and saving plots in the correct dimensions. For now, let’s look at the bbc\_style() function, which applies a custom ggplot theme to make all the plots look consistent and follow BBC style guidelines.

An Example Plot

To see how this function works, you’ll create a plot showing population data about several penguin species. You’ll be using the palmerpenguins package, which contains data about penguins living on three islands in Antarctica. For a sense of what this data looks like, load the palmerpenguins and tidyverse packages:

library(palmerpenguins)

library(tidyverse)

Now you have data you can work with in an object called penguins. Here’s what the first 10 rows look like:

#> # A tibble: 344 × 8

#> species island bill\_le...¹ bill\_...² flipp...³ body\_...⁴ sex

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

#> 1 Adelie Torgersen 39.1 18.7 181 3750 male

#> 2 Adelie Torgersen 39.5 17.4 186 3800 fema...

#> 3 Adelie Torgersen 40.3 18.0 195 3250 fema...

#> 4 Adelie Torgersen NA NA NA NA <NA>

#> 5 Adelie Torgersen 36.7 19.3 193 3450 fema...

#> 6 Adelie Torgersen 39.3 20.6 190 3650 male

#> 7 Adelie Torgersen 38.9 17.8 181 3625 fema...

#> 8 Adelie Torgersen 39.2 19.6 195 4675 male

#> 9 Adelie Torgersen 34.1 18.1 193 3475 <NA>

#> 10 Adelie Torgersen 42 20.2 190 4250 <NA>

--snip--

To get the data in a more usable format, you’ll count how many penguins live on each island with the count() function from the dplyr package (one of several packages that are loaded with tidyverse):

penguins %>%

count(island)

This gives you some simple data that you can use for plotting:

#> # A tibble: 3 × 2

#> island n

#> <fct> <int>

#> 1 Biscoe 168

#> 2 Dream 124

#> 3 Torgersen 52

You’ll use this data multiple times in the chapter, so save it as an object called penguins\_summary like so:

penguins\_summary <- penguins %>%

count(island)

Now you’re ready to create a plot. Before you see what bbplot does, make a plot with the ggplot defaults:

penguins\_plot <- ggplot(

data = penguins\_summary,

aes(

x = island,

y = n,

fill = island

)

) +

geom\_col() +

labs(

title = "Number of Penguins",

subtitle = "Islands are in Antarctica",

caption = "Data from palmerpenguins package"

)

This code tells R to use the penguins\_summary data frame, putting the island on the x-axis and the count of the number of penguins (n) on the y-axis, and making each bar a different color with the fill aesthetic property. Since you’ll modify this plot multiple times, saving it as an object called penguins\_plot simplifies the process. Figure 3-1 shows the resulting plot.

[F03001.pdf]



* + - * 1. A chart with the default theme

This isn’t the most aesthetically pleasing chart. The gray background is ugly, the y-axis title is hard to read because it’s angled, and the text size overall is quite small. But don’t worry, you’ll be improving it soon.

The BBC’s Custom Theme

Now that you have a basic plot to work with, you’ll start making it look like a BBC chart. To do this, you need to install the bbplot package. First, install the remotes package using install.packages("remotes") so that you can access packages from remote sources. Then, run the following code to install bbplot from the GitHub repository at https://github.com/bbc/bbplot:

library(remotes)

install\_github("bbc/bbplot")

Once you’ve installed the bbplot package, load it and apply the bbc\_style() function to the penguins\_plot as follows:

library(bbplot)

penguins\_plot +

bbc\_style()

Figure 3-2 shows the result.

[F03002.pdf]



* + - * 1. The same chart with BBC style applied

Vastly different, right? The font size is larger, the legend is on top, there are no axis titles, the grid lines are stripped down, and the background is white. Let’s look at these changes one by one.

The BBC Theme Components

You’ve just seendifference to a basic chartThis section walks you through the function’s code, with some minor tweaks for readability. Functions are discussed further in Chapter 12.

Function Definition

The first line gives the function a name and indicates that what follows is, in fact, a function definition:

bbc\_style <- function() {

font <- "Helvetica"

ggplot2::theme(

--snip--

The code then defines a variable called font and assigns it the value Helvetica. This allows later sections to simply use font rather than repeating Helvetica multiple times. If the BBC team ever wanted to use a different font, they could change Helvetica here to, say, Comic Sans and it would update the font for all of the BBC plots (though I suspect higher-ups at the BBC might not be on board with that choice).

Historically, working with custom fonts in R was notoriously tricky, but recent changes have made the process much simpler. To ensure that custom fonts such as Helvetica work in ggplot, first install the systemfonts and ragg packages by running this code in the console:

install.packages(c("systemfonts", "ragg"))

The systemfonts package allows R to directly access fonts you’ve installed on your computer, and ragg allows ggplot to use those fonts when generating plots.

Next, select Tools4Global Options. Click the Graphics menu at the top of the interface and, under the Backend option, select AGG. This change should ensure that RStudio renders the previews of any plots with the ragg package. With these changes in place, you should be able to use any fonts you’d like (assuming you have them installed) in the same way that the bbc\_style() function uses Helvetica.

After specifying the font to use, the code calls ggplot’s theme() function. Rather than first loading ggplot with library(ggplot2) and then calling its theme() function, the ggplot2::theme() syntax indicates in one step that the theme() function comes from the ggplot2 package. You’ll write code in this way when making an R package in Chapter 12.

Nearly all of the code in bbc\_style() exists within this theme() function. Remember from Chapter 2 that theme() makes additional tweaks to an existing theme; it isn’t a complete theme like theme\_light(), which will change the whole look and feel of your plot. In other words, by jumping straight into the theme() function, bbc\_style() makes adjustments to the ggplot defaults. As you’ll see, the bbc\_style() function does a lot of tweaking.

Text

The first code section within the theme() function formats the text:

plot.title = ggplot2::element\_text(

family = font,

size = 28,

face = "bold",

color = "#222222"

),

plot.subtitle = ggplot2::element\_text(

family = font,

size = 22,

margin = ggplot2::margin(9, 0, 9, 0)

),

plot.caption = ggplot2::element\_blank(),

--snip--

To make changes to the title, subtitle, and caption, it follows this pattern:

area\_of\_chart = element\_type(

property = value

)

For each area, this code specifies the element type: element\_text(), element\_line(), element\_rect(), or element\_blank(). Within the element type is where you assign values to properties—for example, setting the font family (the property) to Helvetica (the value). The bbc\_style() function uses the various element\_ functions to make tweaks, as you’ll see later in this chapter.

Note For additional ways to customize pieces of your plots, see the ggplot2 package documentation (https://ggplot2.tidyverse.org/reference/element.html), which provides a comprehensive list.

One of the main adjustments the bbc\_style() function makes is bumping up the font size to help with legibility, especially when plots made with the bbplot package are viewed on smaller mobile devices. The code first formats the title (with plot.title) using Helvetica 28-point bold font in a nearly black color (the hex code #222222). The subtitle (plot.subtitle) is 22-point Helvetica.

The bbc\_style() code also adds some spacing between the title and subtitle with the margin() function, specifying the value in points for the top (9), right (0), bottom (9), and left (0) sides. Finally, the caption (through the plot.caption argument) is removed with the element\_blank() function. (As mentioned earlier, the finalise\_plot() function in the bbplot package adds elements, including a caption and the BBC logo, to the bottom of the plots.) Figure 3-3 shows these changes.

[F03003.pdf]



* + - * 1. The penguin chart with only the text formatting changed

With these changes in place, you’re on your way to the BBC look.

Legend

Next up is formatting the legend, positioning it above the plot and left-aligning its text:

legend.position = "top",

legend.text.align = 0,

legend.background = ggplot2::element\_blank(),

legend.title = ggplot2::element\_blank(),

legend.key = ggplot2::element\_blank(),

legend.text = ggplot2::element\_text(

family = font,

size = 18,

color = "#222222"

),

This code removes the legend background (which would show up only if the background color of the entire plot were different), the title, and the legend key (the borders on the boxes that show the island names, just barely visible in Figure 3-3). Finally, the code sets the legend’s text to 18-point Helvetica with the same nearly black color. Figure 3-4 shows the result.

[F03004.pdf]



* + - * 1. The penguin chart with changes to the legend

The legend is looking better, but now it’s time to format the rest of the chart so it matches.

Axes

The code first removes the axis titles because they tend to take up a lot of chart real estate, and you can use the title and subtitle to clarify what the axes show:

axis.title = ggplot2::element\_blank(),

axis.text = ggplot2::element\_text(

family = font,

size = 18,

color = "#222222"

),

axis.text.x = ggplot2::element\_text(margin = ggplot2::margin(5, b = 10)),

axis.ticks = ggplot2::element\_blank(),

axis.line = ggplot2::element\_blank(),

All text on the axes becomes 18-point Helvetica and nearly black. The text on the x-axis (Biscoe, Dream, and Torgersen) gets a bit of spacing around it. Finally, both axes’ ticks and lines are removed. Figure 3-5 shows these changes.

[F03005.pdf]



* + - * 1. The penguin chart with axis formatting changes

The axis text matches the legend text, and the axis tick marks and lines are gone.

Grid Lines

Now for the grid lines:

panel.grid.minor = ggplot2::element\_blank(),

panel.grid.major.y = ggplot2::element\_line(color = "#cbcbcb"),

panel.grid.major.x = ggplot2::element\_blank(),

--snip--

The approach here is fairly straightforward: this code removes minor grid lines for both axes, removes major grid lines on the x-axis, and keeps major grid lines on the y-axis but makes them a light gray (the #cbcbcb hex code). Figure 3-6 shows the result.

[F03006.pdf]



* + - * 1. The penguin chart with adjustments to the grid lines

Notice that the grid lines on the x-axis have disappeared.

Background

The previous iteration of the plot still has a gray background. The bbc\_style() function removes it with the following code:

panel.background = ggplot2::element\_blank(),

Figure 3-7 shows the resulting plot.

[F03007.pdf]



* + - * 1. The chart with the gray background removed

You’ve nearly re-created the penguin plot using the bbc\_style() function.

Small Multiples

The bbc\_style() function contains a bit more code to modify strip.background and strip.text. In ggplot, the strip refers to the text above faceted charts like the ones discussed in Chapter 2. Next, you’ll turn your penguin chart into a faceted chart to see these components of the BBC’s theme. I’ve used the code from the bbc\_style() function, minus the sections that deal with small multiples, to make Figure 3-8.

[F03008.pdf]



* + - * 1. The faceted chart with no changes to the strip text formatting

Using the facet\_wrap() function to make a small multiples chart leaves you with one chart per island, but by default, the text above each small multiple is noticeably smaller than the rest of the chart. What’s more, the gray background behind the text stands out because you’ve already removed the gray background from the other parts of the chart. The consistency you’ve worked toward is now compromised, with small text that is out of proportion to the other chart text and a gray background that sticks out like a sore thumb.

The following code changes the strip text above each small multiple:

strip.background = ggplot2::element\_rect(fill = "white"),

strip.text = ggplot2::element\_text(size = 22, hjust = 0)

)

--snip--

This code removes the background (or, more accurately, colors it white). Then it makes the text larger, bold, and left-aligned using hjust = 0. Note that I did have to make the text size slightly smaller than in the actual chart to fit the book and I added code to make it bold. Figure 3-9 shows the result.

[F03009.pdf]



* + - * 1. The small multiples chart in the BBC style

If you look at any chart on the BBC website, you’ll see how similar it looks to yours. The tweaks in the bbc\_style() function to the text formatting, legends, axes, grid lines, and backgrounds show up in charts viewed by millions of people worldwide.

Color

You might be thinking, Wait, what about the color of the bars? Doesn’t the theme change those? This is a common point of confusion, but the answer is that it doesn’t. The documentation for the theme() function explains why this is the case: “Themes are a powerful way to customize the non-data components of your plots: i.e. titles, labels, fonts, background, gridlines, and legends.” In other words, ggplot themes change the elements of the chart that aren’t mapped to data.

Plots, on the other hand, use color to communicate information about data. In the faceted chart, for instance, the fill property is mapped to the island (Biscoe is salmon, Dream is green, and Torgersen is blue). As you saw in Chapter 2, you can change the fill using the various scale\_fill\_ functions. In the world of ggplot, these scale\_ functions control color, while the custom themes control the chart’s overall look and feel.

Conclusion

When Stylianou and Guibourg started developing a custom theme for the BBC, they had one question: Would they be able to create graphs in R that could go directly onto the BBC website? Using ggplot, they succeeded. The bbplot package allowed them to make plots with a consistent look and feel that followed BBC standards and, most important, did not require a designer’s help.

You can see many of the principles of high-quality data visualization discussed in Chapter 2 in this custom theme. In particular, the removal of extraneous elements (axis titles and grid lines, for instance) helps keep the focus on the data itself. And because applying the theme requires users to add only a single line to their ggplot code, it was easy to get others on board. They had only to append bbc\_style() to their code to produce a BBC-style plot.

Over time, others at the BBC noticed the data journalism team’s production-ready graphs and wanted to make their own. The team members set up R trainings for their colleagues and developed a “cookbook” (https://bbc.github.io/rcookbook/) showing how to make various types of charts. Soon, the quality and quantity of BBC’s data visualization exploded. Stylianou told me, “I don’t think there’s been a day where someone at the BBC hasn’t used the package to produce a graphic.”

Now that you’ve seen how custom ggplot themes work, try making one of your own. After all, once you’ve written the code, you can apply it with only one line of code.

Resources

BBC Visual and Data Journalism Team, “BBC Visual and Data Journalism Cookbook for R Graphics,” GitHub, January 24, 2019, https://bbc.github.io/rcookbook/.

BBC Visual and Data Journalism Team, “How the BBC Visual and Data Journalism Team Works with Graphics in R,” Medium, February 1, 2019, https://medium.com/bbc-visual-and-data-journalism/how-the-bbc-visual-and-data-journalism-team-works-with-graphics-in-r-ed0b35693535.