Making Your Own Theme

custom theme

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 ensures that all data visualization meets 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:

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

Neither of these choices was ideal, and they limited the BBC’s data visualization output. 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 create production-ready charts in R that could go straight onto the BBC website.

In this chapter, we discuss the power of custom ggplot themes, then walk 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.

Using a Custom Theme to Style a Plot

The bbplot package has two functions: bbc\_style() and finalise\_plot(). The latter deals with things 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 any plot, making all plots look consistent and follow BBC style guidelines.

Creating an Example Plot

To show how this function works, let’s create a plot. We’ll do so using the palmerpenguins package, which contains data about penguins living on three islands in Antarctica. To give you a sense of what this data looks like, load the palmerpenguins and tidyverse packages:

library(palmerpenguins)

library(tidyverse)

We now have data that we 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 18 195 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 our data in a more usable format, let’s count how many penguins live on each island. We do this with the count() function from the dplyr package (one of several packages that are loaded when we load the tidyverse):

penguins %>%

count(island)

This gives us some simple data that we can use for plotting:

#> # A tibble: 3 × 2

#> island n

#> <fct> <int>

#> 1 Biscoe 168

#> 2 Dream 124

#> 3 Torgersen 52

Because we’ll use this data multiple times in the chapter, let’s save it as an object called penguins\_summary:

penguins\_summary <- penguins %>%

count(island)

Now that we’ve got some data to work with, we’re ready to create a plot. Before showing what bbplot does, let’s make our plot with the ggplot defaults. Here is the code we’ll use:

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"

)

We use our 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. We’ll modify this plot multiple times, so to simplify this process, we save it as an object called penguins\_plot. Figure 3-1 shows the resulting plot.

[F03001.pdf]



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

It 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: we’ll be improving it soon!

Applying the bbc\_style() Function

Now that we have a basic plot to work with, let’s make it look like a BBC chart. To do this, we must install the bbplot package. First, install the remotes package using install.packages("remotes"). From there, you can run the following code to install bbplot.

library(remotes)

install\_github("bbc/bbplot")

Once the bbplot package is installed, we can then apply the bbc\_style() function to our penguins\_plot:

library(bbplot)

penguins\_plot +

bbc\_style()

Take a look at what happens in Figure 3-2 with the application of bbc\_style() to our plot.

[F03002.pdf]



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

Way different, right? The font size is larger, the legend is on top, there are no axis titles, the grid lines are stripped down, and there is a white background. These are the major changes that the bbc\_style() function makes. Let’s look at them one by one.

Breaking Down the Custom Theme

This section walks through the code for the bbc\_style() function (taken from the bbplot GitHub repository at <https://github.com/bbc/bbplot>, with some minor tweaks for readability). We’ll discuss functions more in Chapter 12.

Setting Up

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--

We then define a variable called font and assign it the value Helvetica. This allows later sections to simply write font rather than repeating Helvetica over and over again. Also, if the BBC team ever wanted to use a different font, they could change Helvetica to, say, Comic Sans and update the font of all BBC plots (though I suspect higher-ups at the BBC might not be on board).

Until recently, working custom fonts in R was notoriously tricky. However, recent changes have made the process much simpler. To ensure that custom fonts such as Helvetica work in ggplot, follow these steps. First, install two packages. systemfonts and ragg, by running this code:

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

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

Second, select Tools 4 Global 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, we call the ggplot2 package’s theme() function. Rather than first loading the package with the code library(ggplot2) and then using its function, we use the syntax ggplot2::theme(), indicating that the theme() function comes from the ggplot2 package. We write code in this way when making an R package, something we’ll discuss in Chapter 12.

Nearly all of the code in the bbc\_style() function 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 tweaks to the ggplot defaults.

As you’ll see, the bbc\_style() function does a lot of tweaking. Let’s go through the changes it makes, section by section.

Text Formatting

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 uses using the following pattern:

AREA\_OF\_CHART = ELEMENT\_TYPE(

PROPERTY = VALUE

)

For each area, we say what type of element it is: element\_text(), element\_line(), element\_rect(), or element\_blank(). Within the element type, we give values to properties. This can be, say, setting the font family (the property) to Helvetica (the value).

One of the main things the bbc\_style() function does is bump up the text size. Increasing font size helps with legibility, especially when plots made using 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 (that’s the hex code #222222). The subtitle (using plot.subtitle) is 22-point Helvetica.

We add some spacing between the title and subtitle using the margin() function, which gives the spacing, in points, for the top (9), right (0), bottom (9), and left (0) sides. Finally, the caption (through the plot.caption argument) is removed using the element\_blank() function. This is done because the finalise\_plot() function in the bbplot package adds elements, including a caption and the BBC logo, to the bottom of plots. Figure 3-3 shows these changes.

[F03003.pdf]



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

With these changes in place, we’re on our way toward the BBC look. Let’s now tweak the legend.

Legend Formatting

Next, we format the legend, putting it on top of the plot and left-aligning the text within it:

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"

),

We remove 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 red, green, and blue boxes that show the island names). Finally, we make the legend’s text 18-point Helvetica with the same nearly black color. We can see the result in Figure 3-4.

[F03004.pdf]



* + - * 1. Our chart with changes to the legend

The legend is looking better, but now we need to format the rest of the chart so it matches.

Axis Formatting

Next are the axes. The code first removes axis titles because these tend to take up a lot of chart real estate, and you can use the title and subtitle to make it clear 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 Helevetica and nearly black. The text on the x axis (in our case, Biscoe, Dream, and Torgersen) gets a bit of spacing around it. Finally, we remove both axis ticks and axis lines. We can see the changes to the axes in Figure 3-5.

[F03005.pdf]



* + - * 1. Our chart with changes to axis formatting

With the axis text now matching the legend text, and the axis ticks and lines removed, we’re ready to deal with the grid lines.

Grid Lines Formatting

Now that we’ve tweaked the overall text formatting, the legend, and the axes, let’s move onto grid lines. The approach here is fairly straightforward: remove all minor grid lines and the major grid lines on the x axis, keeping only major grid lines on the y axis, but making them a light gray (using the #cbcbcb hex code).

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

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

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

--snip--

[F03006.pdf]



* + - * 1. Our chart with tweaks to the grid lines

Background Formatting

The previous iteration of our plot still had a gray background. The bbc\_style() function removes this with the following code.

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

The plot without the gray background is seen in Figure 3-7.

[F03007.pdf]



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

We’ve nearly recreated the penguin plot using the bbc\_style() function. There is just one more tweak to go.

Small Multiples Formatting

The function contains a bit more code, to modify strip.background and strip.text. These elements become relevant in small multiples charts like the one discussed in Chapter 2. Let’s turn our penguin chart into a small multiples 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. Small multiples chart with no changes to the strip text formatting

When we use the facet\_wrap() function to make a small multiples chart, we are left with one chart per island. But note that, by default, the text above each chart is noticeably smaller than the rest of the chart. What’s more, the gray background behind the text stands out when we have removed the gray background from other parts of the chart. The consistency we’ve worked toward is now gone, with small text that is out of proportion to the other text in the chart and a gray background that sticks out like a sore thumb.

I’ve saved the code used to make Figure 3-8 as an object, penguins\_plot\_weight. We now use this object in order to show how to change the text that shows up above each small multiples chart (called the strip in ggplot):

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

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

)

--snip--

We remove the background (or, more accurately, color it white). Then we make the text larger, bold, and left aligned using hjust = 0. I did have to make the text size slightly smaller to fit in the book and added code to make it bold. You can see the result in Figure 3-9.

[F03009.pdf]



* + - * 1. Small multiples chart in the BBC style

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

What About Colors?

You might be thinking: Wait, what about the color of the bars? Doesn’t the theme change those? It’s a common point of confusion, but the answer is that it doesn’t. If we read the documentation for the theme() function, it becomes clearer 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 our small multiples chart, for instance, the fill property is mapped to the island (Biscoe is salmon, Dream is green, and Torgersen is blue). As we saw in Chapter 2, we 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 overall look-and-feel of charts.

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 importantly, did not need help from a designer.

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 became simple to get others on board. Users 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” (found at <https://bbc.github.io/rcookbook/>) that showed 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.

Learn More

Consult the following resources to learn more about how the BBC created and used their custom theme:

BBC Visual and Data Journalism cookbook for R graphics (2019), <https://bbc.github.io/rcookbook/>

“How the BBC Visual and Data Journalism team works with graphics in R” by the BBC Visual and Data Journalism team (2019), <https://medium.com/bbc-visual-and-data-journalism/how-the-bbc-visual-and-data-journalism-team-works-with-graphics-in-r-ed0b35693535>