Make Your Own Theme

In 2017, BBC data journalist Nassos Stylianou was working with a backend developer on a particularly large data set. Nassos was primarily an Excel user at the time, but this data was too large for Excel. Seeing the developer work through the data with ease, a light bulb went off for Stylianou: if he and his data journalism team learned to use R, they could do this type of analysis on their own.

This realization began a journey into R. This journey, which started with needing to analyze data too large for Excel to handle, would ultimately end up in a very different place. In 2018, Stylianou, his colleague Clara Guibourg, and their team created a custom ggplot theme to create plots that match the BBC style. The code in the bbplot package is a great example of the value of developing a custom theme. But the real story of the creation of bbplot is not just about technical tools. Through learning R and creating a custom theme for others to use, Nassos, Clara and their colleagues would change the culture, remove bottlenecks, and allow the BBC to be more creative with their data viz.

To understand how big these changes were, it’s helpful to understand what things looked like at the BBC before bbplot. In the mid-2010s, journalists at the BBC who wanted to make data visualization had two choices:

1. They could use an internal tool. This tool could create data visualization, but only 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 back-and-forth with a designer. As Stylianou described it, working with a designer “is just a very time-consuming workflow if you think of how many visualizations the BBC does.”

Neither of these choices was ideal. And this limited set of less-than-ideal choices led to a limited output of data viz.

That would all change when Stylianou, Guibourg, and their colleagues realized that R, the tool they had decided to learn for data analysis, could also do data visualization. As they began playing around with ggplot, they quickly saw its power. Guibourg said she found it “immediately addictive when I started working with ggplot to make charts.” No longer limited by the BBC’s inflexible internal tool, she found that ggplot was “completely flexible in a way that was just completely new to me.”

The biggest change, though, came from not having to work with a designer. Not because the designers were bad (they weren’t), but because ggplot allowed the BBC data journalists to explore different visualizations on their own. Working with a designer required the journalists to have a fully-formed idea that the designer could take and improve upon. Working in ggplot allowed BBC data journalists to explore different data viz ideas.

Clara Guibourg believes this freedom is what explains the addictive quality of ggplot. As she told me, “even before we got anywhere near having a production-ready chart, just trying things out, visualizing things for the first time” was completely captivating. Having learned the basics of ggplot, she saw that “you can make like the simplest chart with just a couple of lines of code.” Being able to explore different types of visualization on her own led Clara and others to produce more data viz than they had previously.

As the BBC data journalism team improved their ggplot skills, they realized that it might be possible produce for more than just exploratory data viz. They had learned to use R for data analysis and they were starting to use it for exploratory data visualization. Could they go all the way and create production-ready charts in R that could go straight onto the BBC website?

Stylianou, Guibourg, and their colleagues set about looking into what would be involved in creating production-ready charts from R. They realized that so much of this work involved small tweaks. What font should they use? Where should the legend go? Should axes have titles? Should charts have grid lines? These questions may seem small but they have a big impact. Having consistent answers to them is what enabled BBC designers to turn Excel mockups into high-quality data viz ready to go on the website. As the BBC data journalism team dug further into ggplot, they realized that they might be able to write code to make their data viz production-ready. They realized that, if making production-ready charts required asking question about fonts, legends, axes, and grid lines, ggplot had the answer. And the answer was to make a custom theme.

Enter bbplot

In 2018, Stylianou, Guibourg, and others at the BBC developed a package called bbplot. This package has two functions: bbc\_style() and finalise\_plot(). The latter deals with things like adding the BBC logo, saving plots in the correct dimensions, and other tasks done after the plot is complete (we’ll discuss it a bit later on). For now, let’s look at the bbc\_style() function. This function applies a custom ggplot theme to any plot, making all plots look consistent and follow BBC style guidelines.

To show how this function works, let’s create a plot. We’ll do so using the palmerpenguins package, which has data on penguins living on three islands in Antarctica. To give you a sense of what this data looks like, let’s 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 ten rows look like.

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).

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

#> # A tibble: 3 × 2  
#> island n  
#> <fct> <int>  
#> 1 Biscoe 168  
#> 2 Dream 124  
#> 3 Torgersen 52

Because we’re going to use this data multiple times below, 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 a plot with ggplot defaults. Here is the code we’ll use. We’re using our penguins\_summary data frame, putting the island on the x axis, the count of the number of penguins (n) on the y axis, and making each bar a different color with the fill aesthetic property.

The resulting plot, seen in Figure 3-1, isn’t the most aesthetically pleasing chart, but we’ll be improving it soon!

[F03001.pdf]



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

We’re going to use this plot multiple times (with some modifications each time). To simplify things, let’s save it as an object called penguins\_plot.

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"  
 )

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 load the bbplot package.

library(bbplot)

We can then apply the bbc\_style() function to our penguins\_plot.

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? Larger font size, legend on top, no axis titles, stripped down grid lines, and a white background – these are the major changes that the bbc\_style() function makes. Let’s look at them one by one.

Here’s the code for the bbc\_style() function (taken from the bbplot GitHub repository, found at <https://github.com/bbc/bbplot>). You may be a bit confused by the way some of the code is written. This is in part because it is the code used to create a function. The first line gives the function a name (bbc\_style) and indicates that it is, in fact, a function definition. We’ll discuss functions more in Chapter 12.

You’ll see that instead of loading the package ggplot2 with the code library(ggplot2) and then using the theme() function, the code below uses ggplot2::theme(). This indicates that the theme() function comes from the ggplot2 package. Writing code in this way is something that is done when making an R package, something we’ll discuss in Chapter 13.

I’ve made some minor formatting tweaks for readability. For example, you can see the comments in ALL CAPS, which show the category of modification that the section which follows makes. Fortunately for us, the code is organized nicely and allows us to see what each section does.

bbc\_style <- function() {  
 font <- "Helvetica"  
   
 ggplot2::theme(  
   
 # TEXT FORMAT  
 # This sets the font, size, type and colour   
 # of text for the chart's title  
 plot.title = ggplot2::element\_text(  
 family = font,  
 size = 28,  
 face = "bold",  
 color = "#222222"  
 ),  
 # This sets the font, size, type and colour  
 # of text for the chart's subtitle,  
 # as well as setting a margin between the title and the subtitle  
 plot.subtitle = ggplot2::element\_text(  
 family = font,  
 size = 22,  
 margin = ggplot2::margin(9, 0, 9, 0)  
 ),  
 # This leaves the caption text element empty,   
 # because it is set elsewhere in the finalise plot function  
 plot.caption = ggplot2::element\_blank(),  
   
 # LEGEND FORMAT  
 # This sets the position and alignment of the legend,   
 # removes a title and background for it  
 # and sets the requirements for any text within the legend.  
 # The legend may often need some more manual tweaking   
 # when it comes to its exact position based on the plot coordinates.  
 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"  
 ),  
   
 # AXIS FORMAT  
 # This sets the text font, size and colour for the axis test,   
 # as well as setting the margins and removes lines and ticks.  
 # In some cases, axis lines and axis ticks are things we would   
 # want to have in the chart -   
 # the cookbook shows examples of how to do so.  
 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(),  
   
 # GRID LINES  
 # This removes all minor gridlines and adds major y gridlines.  
 # In many cases you will want to change this to remove   
 # y gridlines and add x gridlines.  
 # The cookbook shows you examples for doing so.  
 panel.grid.minor = ggplot2::element\_blank(),  
 panel.grid.major.y = ggplot2::element\_line(color = "#cbcbcb"),  
 panel.grid.major.x = ggplot2::element\_blank(),  
   
 # BLANK BACKGROUND  
 # This sets the panel background as blank, removing the standard   
 # grey ggplot background colour from the plot.  
 panel.background = ggplot2::element\_blank(),  
   
 # STRIP BACKGROUND  
 # This sets the panel background for facet-wrapped plots to white,  
 # removing the standard grey ggplot background colour and sets the   
 # title size of the facet-wrap title to font size 22.  
 strip.background = ggplot2::element\_rect(fill = "white"),  
 strip.text = ggplot2::element\_text(size = 22, hjust = 0)  
 )  
}

Nearly all of the code in the bbc\_style() function exists within the theme() function from ggplot2. In the Chapter 2, we saw how Cédric Scherer and Georgios Karamanis customized their plot by applying the theme\_light() function. This a so-called complete theme, meaning you can call the function and will change the whole look-and-feel of your plot. After applying theme\_light(), Scherer and Karamanis used the theme() function make additional tweaks. The bbc\_style() theme does not use a complete theme to start. Instead, by jumping straight into the theme() function, they make tweaks to the ggplot defaults. As you can see, the bbc\_style() function does a lot of tweaking. So, let’s go through the changes it makes, section by section.

Text Formatting

The first section of the code deals with text formatting. First, it defines a variable called “font” and assigns it the value “Helvetica.” This allows later sections of code to simply write “font” rather than repeating “Helvetica” over and over again. And, if the team ever wanted to use a different font, they could simply change “Helvetica” to, say, “Comic Sans” and change all BBC plots (I suspect higher-ups at the BBC might not be on board).

font <- "Helvetica"

Subsequent pieces of this section of the code make changes to the title, subtitle, and caption. The pattern used in code to make changes is as follows:

AREA\_OF\_CHART = ELEMENT\_TYPE(  
 PROPERTY = VALUE  
)

We begin by selecting an area of the chart (for example, plot.title). Then, we have to say what type of element it is. The options are element\_text(), element\_line(), element\_rect(), and element\_blank(). We’ll deal with the other three later on. For now, we’re working with element\_text() to handle formatting of the title, subtitle, and caption since they’re all text elements. 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 that the bbc\_style() function does is to bump up the text size. As Nassos put it to me, on a lot of plots made with ggplot, “font and the numbers are just so small.” 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 using Helvetica 28-point bold font in a nearly black color (that’s the hex code #222222). The subtitle is 22-point Helvetica. Some spacing is added 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 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.

penguins\_plot +  
 theme(  
 plot.title = element\_text(  
 family = font,  
 size = 28,  
 face = "bold",  
 color = "#222222"  
 ),  
 plot.subtitle = element\_text(  
 family = font,  
 size = 22,  
 margin = margin(9, 0, 9, 0)  
 ),  
 plot.caption = element\_blank()  
 )

We can see these changes in Figure 3-3 below.

[F03003.pdf]



* + - * 1. Our chart with only text formatting changed

We then save our plot as an object in order to work with it in the next section.

penguins\_plot\_text <- penguins\_plot +  
 theme(  
 plot.title = element\_text(  
 family = font,  
 size = 28,  
 face = "bold",  
 color = "#222222"  
 ),  
 plot.subtitle = element\_text(  
 family = font,  
 size = 22,  
 margin = margin(9, 0, 9, 0)  
 ),  
 plot.caption = element\_blank()  
 )

Legend Formatting

Next, we deal with the legend. The code puts the legend on top of the plot, and left aligns the text within it. Then, it removes the legend background (this would only show up if the background color of the entire plot were different than the legend background), title, and legend key (this is a box that can show up around the boxes with the names of the islands). Finally, we make the legend text 18-point Helvetica with the same nearly black color.

penguins\_plot\_text +  
 theme(  
 legend.position = "top",  
 legend.text.align = 0,  
 legend.background = element\_blank(),  
 legend.title = element\_blank(),  
 legend.key = element\_blank(),  
 legend.text = element\_text(  
 family = font,  
 size = 18,  
 color = "#222222"  
 )  
 )

We can see the result in Figure 3-4.

[F03004.pdf]



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

And again, we save this plot so we can continue to alter it below.

penguins\_plot\_legend <- penguins\_plot\_text +  
 theme(  
 legend.position = "top",  
 legend.text.align = 0,  
 legend.background = element\_blank(),  
 legend.title = element\_blank(),  
 legend.key = element\_blank(),  
 legend.text = element\_text(  
 family = font,  
 size = 18,  
 color = "#222222"  
 )  
 )

Axis Formatting

Next up are the axes. The code first removes axis titles because, as Nassos told me, these tend to take up a lot of chart real estate and you can use the title and subtitle to make clear what the axes show. All text on axes becomes 18-point Helevetica nearly black. The text on the x axis (in our case, Biscoe, Dream, and Torgersen) gets a bit of spacing around it. And, finally, both axis ticks and axis lines are removed.

penguins\_plot\_legend +  
 theme(  
 axis.title = element\_blank(),  
 axis.text = element\_text(  
 family = font,  
 size = 18,  
 color = "#222222"  
 ),  
 axis.text.x = element\_text(margin = margin(5, b = 10)),  
 axis.ticks = element\_blank(),  
 axis.line = element\_blank()  
 )

We can see the changes to our axes in Figure 3-5.

[F03005.pdf]



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

Let’s now save this plot as an object for future tweaks.

penguins\_plot\_axes <- penguins\_plot\_legend +  
 theme(  
 axis.title = element\_blank(),  
 axis.text = element\_text(  
 family = font,  
 size = 18,  
 color = "#222222"  
 ),  
 axis.text.x = element\_text(margin = margin(5, b = 10)),  
 axis.ticks = element\_blank(),  
 axis.line = element\_blank()  
 )

Grid Lines Formatting

Now that we’ve tweaked 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, remove 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).

penguins\_plot\_axes +  
 theme(  
 panel.grid.minor = element\_blank(),  
 panel.grid.major.y = element\_line(color = "#cbcbcb"),  
 panel.grid.major.x = element\_blank()  
 )

We can see the result of these tweaks to the grid lines in Figure 3-6.

[F03006.pdf]



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

And, once again, we save our plot to an object.

Background Formatting

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

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

[F03007.pdf]



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

Small Multiples Formatting

And there we go! We’ve now recreated the plot that we made above using the bbc\_style() function. However, you may recall there is a bit more code in the bbc\_style() function. This code deals with strip.background and strip.text. Both of these occur when we make small multiples charts. Small multiples is a common technique in data visualization, where, instead of making one chart that incorporates all of the available data, we break the chart into multiple charts in order to make the final results easier for the reader to comprehend.

Let’s make an example small multiples chart to show what this looks like. I’ve used the code from the bbc\_style() function to make Figure 3-8 below.

[F03008.pdf]



* + - * 1. Small multiples chart with no changes to the strip text formatting

When we use the facet\_wrap() function, 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. And the gray background behind the text stands out when we have removed the gray background from other parts of the chart.

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 (in ggplot this text is called the “strip”). We remove the background (or, more accurately, make it white) and 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, something done in the chart on carbon impact of food chart, though not seen in the bbc\_style() code.)

penguins\_plot\_weight +  
 theme(  
 strip.background = element\_rect(fill = "white"),  
 strip.text = element\_text(size = 17, hjust = 0, face = "bold")  
 )

The result shows up 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 site, you’ll see how similar it is to our chart. All of the tweaks in the bbc\_style() function (text formatting, legends, axes, grid lines, and backgrounds) that we used to make our example show up in charts seen by millions on the BBC website.

What About Colors?

You might be thinking: wait, what about the colors? Doesn’t the theme change that? It’s a common point of confusion. If we read the documentation for the theme() function, though, 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.

Color (or, technically, in the case of the bar charts we have made in this chapter, fill) is used in plots as an aesthetic property to show something about data. In our small multiples chart, for instance, fill is mapped to the island (Biscoe is salmon, Dream is green, and Torgersen is blue). As we saw in Chapter 2, we can change fill using the various scale\_fill\_ functions. It is because fill is tied to the data rather than being about the overall look-and-feel that ggplot themes do not, on their own, change this component of plots.

Code is the Catalyst for Culture Change

When Nassos Stylianou and Clara 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 straight onto the BBC website? And, wouldn’t you know, they succeeded! The creation of the bbplot package allowed them to make plots that had a consistent look-and-feel, followed BBC standards, and, most importantly, did not need help from a designer.

Many of the principles of high-quality data visualization that we discussed in Chapter 2 can be seen 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 by creating a custom theme that only requires users to add a single line to their ggplot code, it became simple to get others on board. Telling users they could just append bbc\_style() to their code and get a BBC-style plot was an eye-opener.

The development of the bbplot package would lead to significant changes at the BBC. It inspired Stylianou, Guibourg, and the other data journalists who created it to use ggplot more than before. Knowing that they had the flexibility of ggplot at their fingertips gave them license to explore. And knowing that they did not have to work with a designer to create production-ready graphics empowered them to make more and better graphics.

In addition to the bbc\_style() function, the bbplot package also provides another function called finalise\_plot() that adds a source at the bottom of the chart (recall how the bbc\_style() function removed the caption), adds the BBC logo in the footer, and gives height, width, and file name options for saving the plot. These two functions combined allowed Nassos, Clara, and others to achieve their holy grail: creating production-ready graphs that could go straight from R to the BBC website.

The impact of bbplot would also come to be seen outside of the small team of data journalists that brought it to life. Others at the BBC saw how the data journalism team was now able to produce production-ready graphs and they wanted to do the same. This led the data journalism team to set up R trainings for their colleagues and to develop a “cookbook” (found at <https://bbc.github.io/rcookbook/>) that provided examples of how to make various types of charts.

These two resources led to a large increase in R users at the BBC. As Stylianou told me, they “spurred people a lot people outside of the data journalism team to take a real interest [in R].” Having bbplot made the value of R click for many people at the BBC. He continued:

There is no, “why am I doing this?” in their mind. It’s is worth the pain [to learn R], because it is a pain at first. But seeing this graphic that a few months ago you would have had to do in this old process … if you devote a bit of time each day, here are the five lines of code that you can run and you can [make a production-ready graphic] yourself.

As so many more people at the BBC came to learn R, the quality and quantity of data visualization produced 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.” The bbplot package came in particularly helpful during COVID. Being able to produce on-brand graphics on a quick turnaround was possible in a way it would not have been previously.

Reflecting on her experience, Guibourg attributes the successful transition to R at the BBC to its culture. As she put it, “I think that what helped me get started was that there was a really supportive environment internally at the BBC for learning.” And, indeed, this same supportive culture that led Clara to organically explore what R was capable of was reinforced after she and the data journalism team released bbplot. The custom theme they developed enabled the creation of so many BBC graphics that otherwise never would have seen the light of day. A culture open to learning led the data journalism team to insights about the power of code. And this code then facilitated a culture change around how graphics are produced at the BBC.