Making High-Quality Tables

In his book Fundamentals of Data Visualization, Claus Wilke writes that “tables are an important tool for visualizing data.” This statement might seem might odd. Tables are often seen as the opposite of data visualization: plots are (or should be) highly-designed tools of communication; tables are where we dump numbers for the few nerds who care to read them. But Wilke sees things differently. Tables should not be data dumps devoid of design. He writes: “because of their apparent simplicity, they may not always receive the attention they need.”

Tables should be treated as data visualization because that is exactly what they are. As the term data visualization has become codified, it has become a synonym for graphs. But think about what the phrase data visualization really means. Don’t overthink it. It simply means to visualize data. And while bars, lines, and points in graphs are visualizations, so too are numbers in a table. When we make tables, we visualize our data.

And since we’re visualizing data, we should care about design. Need proof that good design matters when it comes to making tables? Look at tables made by reputable news organizations. Data dumps these are not. News organizations, whose job is to communicate clearly and effectively, pay a lot of attention to table design.

We saw in Chapter 2 that a few simple but significant tweaks can drastically improve the quality of our graphs. In this chapter, we’ll see that a little bit of work can go a long way toward improving our tables.

The good news for you is that R is a great tool for making high-quality tables. If you are writing reports in RMarkdown (which you can learn about in 6), you can write code that will generate a table when you export your document. Working with the same tool to generate tables alongside your text and data visualization means you don’t have to copy and paste your data, running the risk of human error.

Generating tables in Microsoft Word, the tool that many use to make tables, has other potential pitfalls. Claus Wilke found that his version of Word had 105 built-in table styles. Of those, around 80 percent violated some key principles of table design. Wilke writes:

So if you pick a Microsoft Word table layout at random, you have an 80% chance of picking one that has issues. And if you pick the default, you will end up with a poorly formatted table every time.

In R, there are a number of packages to make a wide range of tables. And within these packages, there are a number of functions designed to make sure your tables follow important design principles.

The rest of this chapter will examine what these design principles are and show how to apply them in your tables made in R. We’ll begin by with a brief trip into the world of table design. After examining the principles that Claus Wilke and other experts recommend, we’ll learn how to apply these principles. For this chapter, I spoke with Tom Mock of Posit (the company that makes RStudio), who has become something of an R table connoisseur. His 2020 blog post “10+ Guidelines for Better Tables in R” takes table design principles and shows how to implement them using the gt package. We’ll walk through examples of Tom’s code to show how small tweaks can make a big difference in improving your tables.

Table Design Principles

Advice on data visualization has become ubiquitous in the last few years. Books, articles, blog posts, and more talk about how to make your graphs communicate effectively. Table design advice is less common, but it is out there. In addition to Claus Wilke, others including Jon Schwabish and Stephen Few have written about table design. All three of these experts come to discussing tables after having written about making effective graphs. The principles they discuss, not surprisingly, will sound similar to data visualization advice. The principles of effective communication apply no matter the form in which data is ultimately presented.

The principles below are adapted primarily from a conversation I had with Tom Mock, which focuses on his tables blog post. That blog post shows how to implement in R the ten table design principles that Jon Schwabish discusses in his article “Ten Guidelines for Better Tables.” Schwabish cites Stephen Few’s work on table design. As you can see, the world of table design is closely connected. Rather than trying to and show every single principle that Schwabish discusses and Mock implements in R, I’ve selected what I think are six of the most important.

In this chapter, I use the gt package. This is one of the most popular table-making packages and, as we’ll see below, it uses good design principles by default. The code below is a lightly adapted version of the code in Mock’s blog post.

Principle One: Minimize Clutter

As with data visualization, one of the most important principles of table design is to minimize clutter. One of the most important ways we can do this is by removing unnecessary elements. One of the most common unnecessary elements that clutter tables is gridlines. To show you how we can make more effective tables by removing gridlines, let’s first load the packages we need. We’re relying on the tidyverse package for general data manipulation functions, gapminder for the data we’ll use, and gt to make the tables.

library(tidyverse)

library(gapminder)

library(gt)

library(gtExtras)

As we saw in Chapter 2, the gapminder package provides data on country-level demographic statistics. To make a data frame we’ll use for our table, let’s use just a few countries (the first four in alphabetical order: Afghanistan, Albania, Algeria, and Angola) and a few years (1952, 1972, and 1992). The gapminder data has many years but we only need a few to demonstrate table-making principles.

I’ve created a data frame called gdp. Let’s see what it looks like.

#> # A tibble: 4 × 4

#> Country `1952` `1972` `1992`

#> <chr> <dbl> <dbl> <dbl>

#> 1 Afghanistan 779. 740. 649.

#> 2 Albania 1601. 3313. 2497.

#> 3 Algeria 2449. 4183. 5023.

#> 4 Angola 3521. 5473. 2628.

Now that we’ve created the data frame we can work with, it’s time to talk about reducing clutter by getting rid of gridlines. Often, you see tables that look like this:

[F05001.png]



* + - * 1. Table with gridlines everywhere

Having gridlines around every single cell in our table is unnecessary and creates visual clutter that distracts from the goal of communicating clearly. A table with minimal or even no gridlines is a much more effective communication tool.

[F05002.png]



* + - * 1. Table with only horizontal gridlines

You know how I mentioned before that gt uses good table design principles by default? This is a great example of it. The second table, with minimal gridlines, requires just two lines. We pipe our gdp data into the gt() function, which creates a table.

gdp %>%

gt()

To make the example with gridlines everywhere, we would have to add additional code. The code that follows gt() here adds gridlines.

gdp %>%

gt() %>%

tab\_style(

style = cell\_borders(

side = "all",

color = "black",

weight = px(1),

style = "solid"

),

locations = list(

cells\_body(

everything()

),

cells\_column\_labels(

everything()

)

)

) %>%

opt\_table\_lines(extent = "none")

Since I don’t recommend doing this, I won’t walk through the code. The important thing to remember is that you get good defaults using gt(). Take advantage of them!

If we wanted to remove additional gridlines, we could use the following code. The tab\_style() function uses a two-step approach:

1. Identify the style we want to modify (in this case the borders).
2. Tell the function where to apply these styles.

Here, we tell tab\_style() that we want to modify the borders using the cell\_borders() function, making our borders transparent. Then, we say that we want this to apply to the cells\_body() location (other options include cells\_column\_labels() for the row with country, 1952, 1972, and 1992).

gdp %>%

gt() %>%

tab\_style(

style = cell\_borders(color = "transparent"),

locations = cells\_body()

)

Doing this gives us a table with no gridlines at all in the body.

[F05003.png]



* + - * 1. Table with gridlines only on the header row and bottom

I’ll then save this table as an object called table\_no\_gridlines so that we can add onto it below.

Principle Two: Differentiate the Header from the Body

While reducing clutter is an important goal, going too far can have negative consequences. A table with no gridlines at all can make it hard to differentiate between the header row and the table body.

[F05004.png]



* + - * 1. Table with all gridlines removed

We saw how to use appropriate gridlines above. We can make our header row bold to make it stand out even more. We start with the table\_no\_gridlines object (our saved table from above). Then, we apply our formatting with the tab\_style() function two-step, first saying we want to alter the text (using the cell\_text() function) by setting the weight to bold and then saying we want this to happen only to the header row (using the cells\_column\_labels() function).

table\_no\_gridlines %>%

tab\_style(

style = cell\_text(weight = "bold"),

locations = cells\_column\_labels()

)

We can see what our table with headers bolded looks like below.

[F05005.png]



* + - * 1. Table with header row bolded

Let’s save this table as table\_bold\_header in order to reuse it below and add additional formatting on top of what’s already there.

Principle Three: Align Appropriately

A third principle of high-quality table design is appropriate alignment. Specifically, numbers in tables should be right-aligned. Tom Mock explains why:

Left-alignment or center-alignment of numbers impairs the ability to clearly compare numbers and decimal places. Right-alignment lets you align decimal places and numbers for easy parsing.

We can see this in action. In the table below, we’ve left aligned 1952, center aligned 1972, and right aligned 1992. You can see how much easier it is to compare the values in 1992 than in the other two columns. In both 1952 and 1972, it is much more difficult to compare the numeric values because the numbers in the same columns (the tens place, for example) are not in the same vertical position. In 1992, however, the number in the tens place in Afghanistan (4) aligns with the number in the tens place in Albania (9) and all other countries. This vertical alignment makes it easier to scan the table.

[F05006.png]



* + - * 1. Table with year columns aligned left, center, and right

As with other tables, we’ve actually had to override the defaults to get the gt package to misalign the columns (code shown below). By default, gt will right align numeric values. So, just don’t change anything and you’ll be golden!

table\_bold\_header %>%

cols\_align(align = "left",

columns = 2) %>%

cols\_align(align = "center",

columns = 3) %>%

cols\_align(align = "right",

columns = 4)

Right alignment is best practice for numeric columns, but for text columns, we want to use left alignment. As Jon Scwabish points out, it’s much easier to read longer text cells when they are left aligned. This is even easier to see if we add a country with a long name to our table. I’ve added Bosnia and Herzegovina and saved this as a data frame called gdp\_with\_bosnia.

gdp\_with\_bosnia

#> # A tibble: 5 × 4

#> Country `1952` `1972` `1992`

#> <chr> <dbl> <dbl> <dbl>

#> 1 Afghanistan 779. 740. 649.

#> 2 Albania 1601. 3313. 2497.

#> 3 Algeria 2449. 4183. 5023.

#> 4 Angola 3521. 5473. 2628.

#> 5 Bosnia and Herzegovina 974. 2860. 2547.

Let’s then take the gdp\_with\_bosnia data frame and create a table with the country column center aligned. In this table, it is hard to scan the country names and that center-aligned column just looks a bit weird.

[F05007.png]



* + - * 1. Table with country column center aligned

This is another example where we’ve had to change the gt defaults to mess things up. The gt package has good default alignment practices for other column types as well. In addition to right aligning numeric columns by default, it will also left align character columns. So, if we don’t touch anything, gt will give us the alignment we’re looking for.

[F05008.png]



* + - * 1. Table with country column left aligned

If you ever do want to override the default alignments, you can use the cols\_align() function. Within this function, we use the columns argument to tell gt which columns to align and the align argument to select our alignment. That table above with the country names center aligned? Here’s how I made it.

gdp\_with\_bosnia %>%

gt() %>%

tab\_style(

style = cell\_borders(color = "transparent"),

locations = cells\_body()

) %>%

tab\_style(

style = cell\_text(weight = "bold"),

locations = cells\_column\_labels()

) %>%

cols\_align(columns = "Country",

align = "center")

Principle Four: Use the Right Level of Precision

In all of the tables we’ve made so far, we’ve used the data exactly as it came to us. In all of the numeric columns, we have data to four decimal places. This is almost certainly too many. Having more decimal places than necessary makes our table harder to read. There is always a balance between what Jon Schwabish describes as “necessary precision and a clean, spare table.” I’ve also heard it described that, if adding additional decimal places would change some action, keep them; otherwise, take them. out My general experience is that people tend to leave too many decimal places in, assuming that accuracy to a very high degree is more important than it is (and, in the process, they reduce the legibility of their tables).

Looking at our GDP table, we can use the fmt\_currency() function to format our numeric values. The gt package has a whole series of functions for formatting values in tables. All of these functions start with fmt\_. In the code below, we set fmt\_currency() to be applied to the 1952, 1972, and 1992 columns. I then use decimals argument to tell fmt\_currency() to format the values with zero decimal places (the difference between a GDP of $799.4453 and $779 is unlikely to lead to different decisions so I’m comfortable with sacrificing precision for legibility).

table\_bold\_header %>%

fmt\_currency(

columns = c(`1952`, `1972`, `1992`),

decimals = 0

)

What we end up with is values formatted as dollars, with a thousands-place comma automatically added by fmt\_currency() to make the values even easier to read.

[F05009.png]



* + - * 1. Table with numbers rounded to whole numbers and dollar sign added

Let’s now save our table for reuse below.

Principle Five: Use Color Intentionally

Up to this point, our table has not had any color. We’re now going to add some, using color to highlight outliers. Especially for those readers who want to scan your table, highlighting outliers with color can help significantly. Let’s make the highest value in any single year a different color. To do this, we again use the tab\_style() function. Within this function, I’m using the cell\_text() function to change both the color of text to orange and make it bold. I’m then using the locations argument to say that we want to adjust cells in the body of the table. Within the cells\_body() function, we have to specify both the columns we want to apply our change to and the rows. If we just look at 1952, we see that we set the columns equal to that year. The rows are set to a more complicated formula. The text rows = 1952 == max(1952) means that the text transformation will occur in rows where the value is equal to the maximum value in that year.

table\_whole\_numbers %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1952`,

rows = `1952` == max(`1952`)

))

We then repeat this same code for 1972 and 1992, with the result shown below.

[F05010.png]



* + - * 1. Table with color added to show the highest value in each year

As always, we save this table to avoid having to repeat all of the formatting code we’ve created up to this point.

Principle Six: Add Data Visualization Where Appropriate

Adding color to highlight outliers is one way to help guide the reader’s attention. Another way is to incorporate graphs into tables. Tom Mock has developed an add-on package for gt called gtExtras that makes it possible to do just this. In our table that we’ve made we might want to show the trend of GDP by country. To do that, we’ll add a new column that shows this trend using a sparkline (essentially, a simple line chart). The gt\_plt\_sparkline() function that we’ll use to do this requires us to have a single column with all of the values needed to make the sparkline. We’ll create a variable called Trend using group\_by() and mutate(). This variable will be a list of the values for each country (so, for Afghanistan, it would be 779.4453145, 739.9811058, and 649.3413952). We’ll save this as an object called gdp\_with\_trend.

gdp\_with\_trend <- gdp %>%

group\_by(Country) %>%

mutate(Trend = list(c(`1952`, `1972`, `1992`))) %>%

ungroup()

From there, we create our table, same as before. But at the end of our code, we add the gt\_plt\_sparkline() function. Within this function, we specify which column to use to create the sparkline (Trend). We set label = FALSE to remove text labels that gt\_plt\_sparkline() adds by default. And we add palette = c("black", "transparent", "transparent", "transparent", "transparent") to make the sparkline black and all other elements of it transparent (by default, the function will make different parts of the sparkline different colors).

gdp\_with\_trend %>%

gt() %>%

tab\_style(

style = cell\_borders(color = "transparent"),

locations = cells\_body()

) %>%

tab\_style(

style = cell\_text(weight = "bold"),

locations = cells\_column\_labels()

) %>%

fmt\_currency(

columns = c(`1952`, `1972`, `1992`),

decimals = 0

) %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1952`,

rows = `1952` == max(`1952`)

)) %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1972`,

rows = `1972` == max(`1972`)

)) %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1992`,

rows = `1992` == max(`1992`)

)) %>%

gt\_plt\_sparkline(column = Trend,

label = FALSE,

palette = c("black", "transparent", "transparent", "transparent", "transparent"))

This stripped-down sparkline now allows the reader to see the trend for each country at a glance.

[F05011.png]



* + - * 1. Table with sparkline added to show trend over time

Conclusion

Many of the tweaks we made to create an effective table are quite subtle. Things like removing excess gridlines, bolding header text, right aligning numeric values, and adjusting the level of precision can often go unnoticed. But skip them and your table will be far less effective. What we ended up with is not flashy, but it does communicate clearly, which is the main goal of tables.

We used the gt package to make a high-quality table. One benefit of using this package is that we were able to use the gt\_plt\_sparkline() function from the gtExtras package to easily add a sparkline to our table. gtExtras does way more than this, though. This package has a set of “theme” functions to allow you to make your tables look like those made by FiveThirtyEight, the New York Times, the Guardian, and other news outlets. I’ve removed the formatting we created and instead used the gt\_theme\_538() function to make our tables look like they came from that organization.

gdp %>%

group\_by(Country) %>%

mutate(Trend = list(c(`1952`, `1972`, `1992`))) %>%

ungroup() %>%

gt() %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1952`,

rows = `1952` == max(`1952`)

)) %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1972`,

rows = `1972` == max(`1972`)

)) %>%

tab\_style(style = cell\_text(color = "orange",

weight = "bold"),

locations = cells\_body(

columns = `1992`,

rows = `1992` == max(`1992`)

)) %>%

fmt\_currency(

columns = c(`1952`, `1972`, `1992`),

decimals = 0

) %>%

gt\_plt\_sparkline(column = Trend,

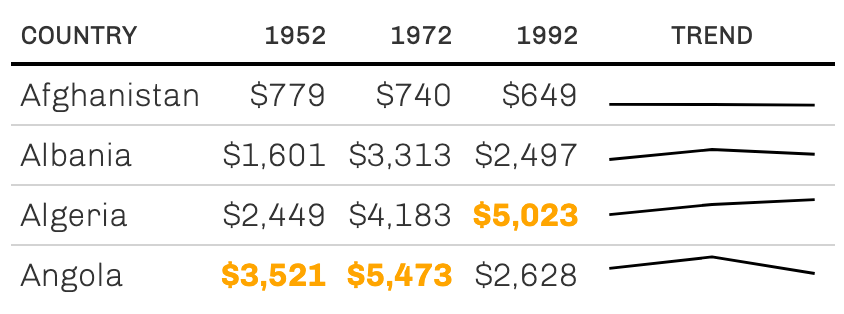
label = FALSE,

palette = c("black", "transparent", "transparent", "transparent", "transparent")) %>%

gt\_theme\_538()

Take a look at tables on the FiveThirtyEight website and you’ll see the similarities to this table.

[F05012.png]



* + - * 1. Table redone in FiveThirtyEight style

Add-on packages like gtExtras are common in the table-making landscape. If you are working with the reactable package to make interactive tables, for example, you can also use the reactablefmtr to add interactive sparklines, themes, and more. The functionality that you get from these packages is enough to never make you go back to making tables in Word!

No matter which package you use to make tables, it’s essential to treat them as worthy of as much thought as data visualization (because, let me remind you, tables are data visualization). Good tables are well designed; they are not data dumps. And fortunately for us, R is well-suited to making well designed tables. The gt package, as we’ve repeatedly seen, has good defaults built in. Oftentimes, you don’t need to change much to end up with high-quality tables.

And it’s not just that we have good packages to make tables. R is a great tool for making tables because it’s the tool you’re already using to create your reports (especially if you’re using RMarkdown, a tool we discuss in 6). What better than using just a few lines of code to make publication-ready tables?