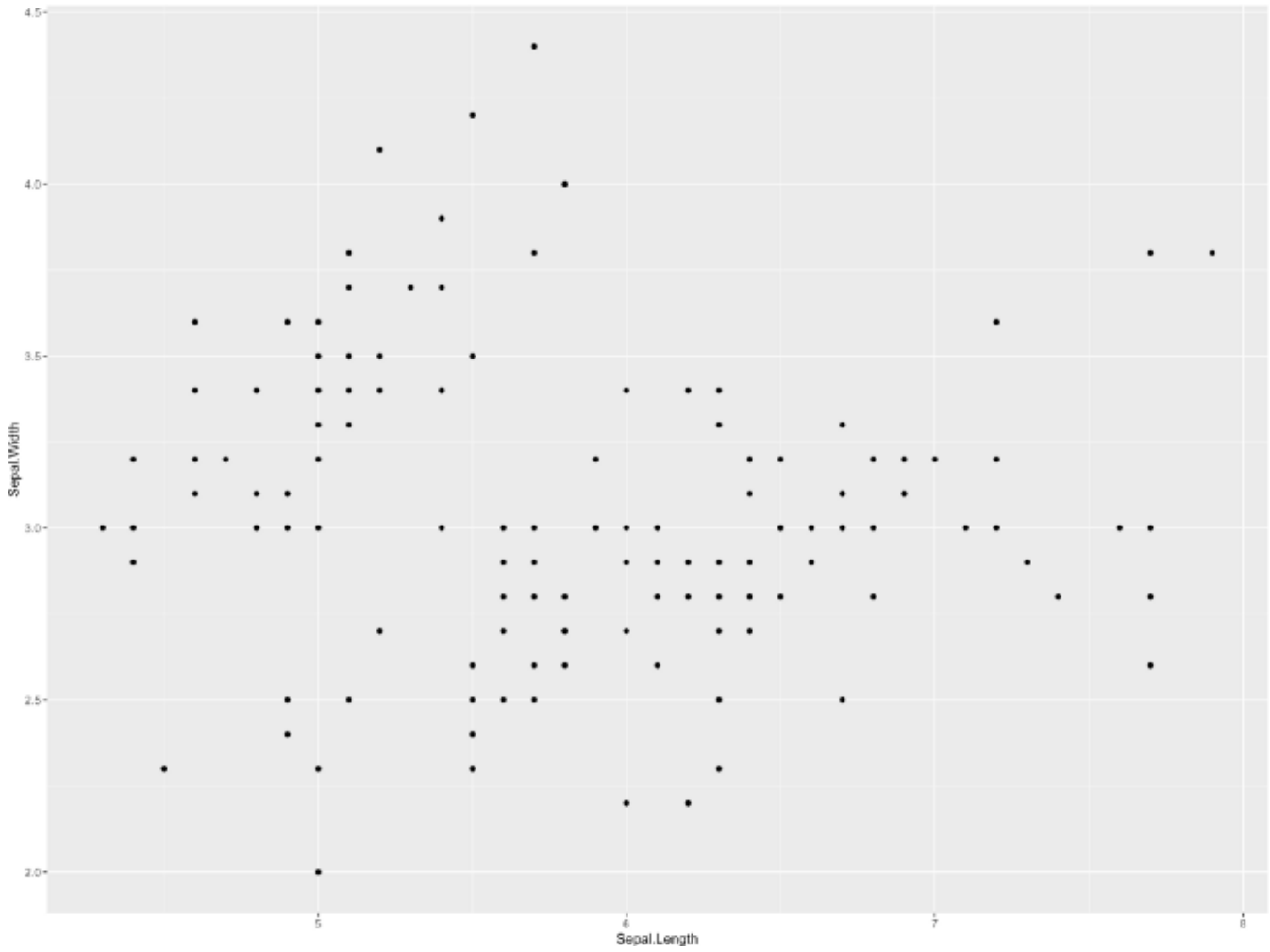
Photo by Edward Howell on Unsplash

8 Tips for Better Data Visualization

Practical Advice for Improving Your GGPlot

William Chon
Aug 12 · 11 min read ★

Ggplot is R's premier data visualization package. Its popularity can likely be attributed to its ease of use — with just a few lines of code you are able to produce great visualizations. This is especially great for beginners who are just beginning their journey into R, as it's very encouraging that you can create something visual with just two lines of code:



Output of above code

In this article, I want to highlight ggplot's flexibility and customizability. Alternatives such as Matplotlib and Seaborn (both Python) and Excel are also easy to use, but they are less customizable. In this article, I'll walk through 8 concrete steps you can do to improve your ggplot.

In order to make sure that the advice in this article is practical, I'm going to abide by two themes:

1. **Assume that the reader has some familiarity with ggplot:** If you understood the chunk of code above you should be good. If you're not familiar with ggplot, I'll try to make the tips as language-agnostic as possible, so if you use base R, Python, or other visualization tools these tips may still be helpful.

dataset, which is included with ggplot.

Source: https://ggplot2.tidyverse.org/reference/diamonds.html

. . .

1. Themes are your best friend

Themes control all non-data display and are an easy way to change the appearance of your graph. **It only takes one extra line of code** in order to do this, and ggplot already comes with 8 separate themes.

```
ggplot(data = diamonds, aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point() +
  theme_bw()
```

The ggplot themes are simple. They won't really stand out, but **they look great, are easy to read, and get the point across.** Also, if you want to use the same theme over and over, you can **set a global theme with one line of code and it will apply to all ggplots.**

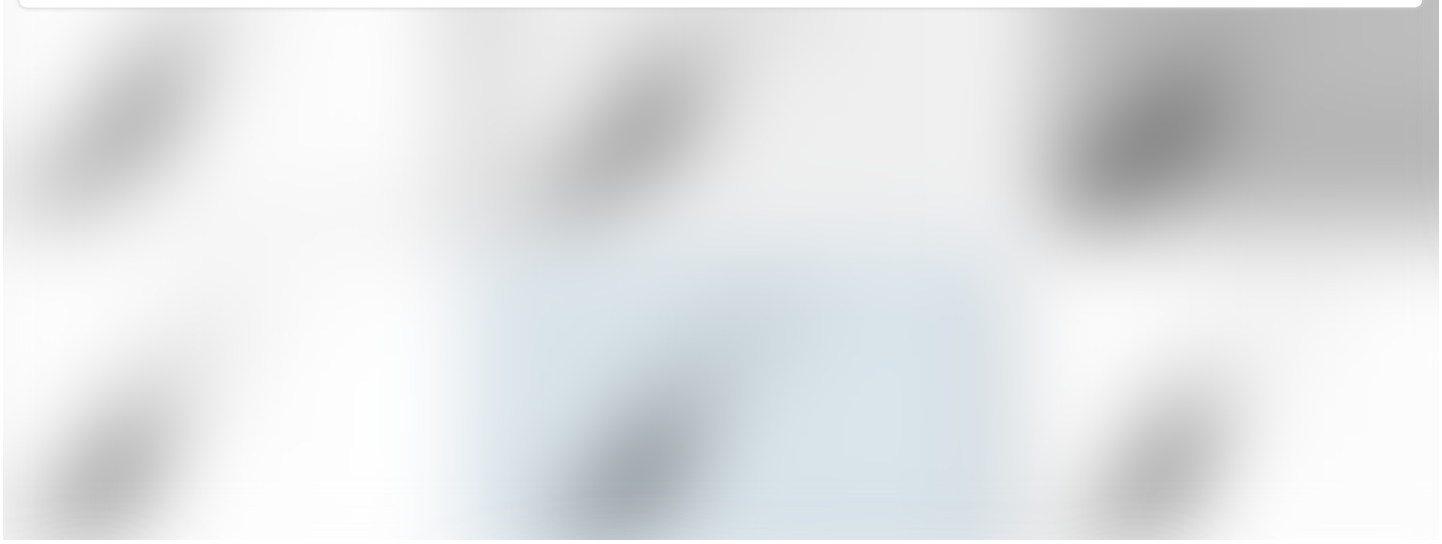
```
# Set global theme

theme_set(theme_bw())
```

ggplot themes, compared

Themes are also super customizable. Beyond the 8 themes that come with ggplot, you can also make your own theme but more importantly use themes that others have created already. In the few companies I've worked at, we've all had internal ggplot themes. For example, I helped create `theme_fb()` at Facebook which with input from designers at the company.

If you wanted to use some other themes outside of ggplot, the most popular package is ggthemes which has some interesting options such as `theme_fivethirtyeight()`, `theme_wsj()`, and `theme_economist()`. A sample of these themes are below, but I definitely recommend checking out **this link** to learn more.



ggthemes, compared

. . .

2. Facets are a superpower

When visualizing data, one thing you always want to think about is how many dimensions of data you want to display. A majority of graphs will typically only need 1–2 dimensions of data to get a point across, for example:

- Height x weight of basketball players on a scatter plot
- Heights of players on the Los Angeles Lakers on a bar graph

As you increase the number of dimensions, a single graph is going to get more cluttered, which makes it harder to get a point across. For example:

- Height x weight of basketball players on a scatter plot, *but different color dots for each of the 30 teams*. This will be hard to read because you'll need 30 separate colors to represent the different teams, and a legend to list out all of the team names.

This is where the magic of faceting shines. **What if we don't have to limit ourselves to one graph?** My hypothesis for why a lot of us think this way is because we are used to visualizing data in Excel, where we are constrained to a single graph. In ggplot, we can break this mode of thinking, and all it takes is a single line of code to do so. **Facets allow us to easily add up to two additional dimensions our visualizations.**

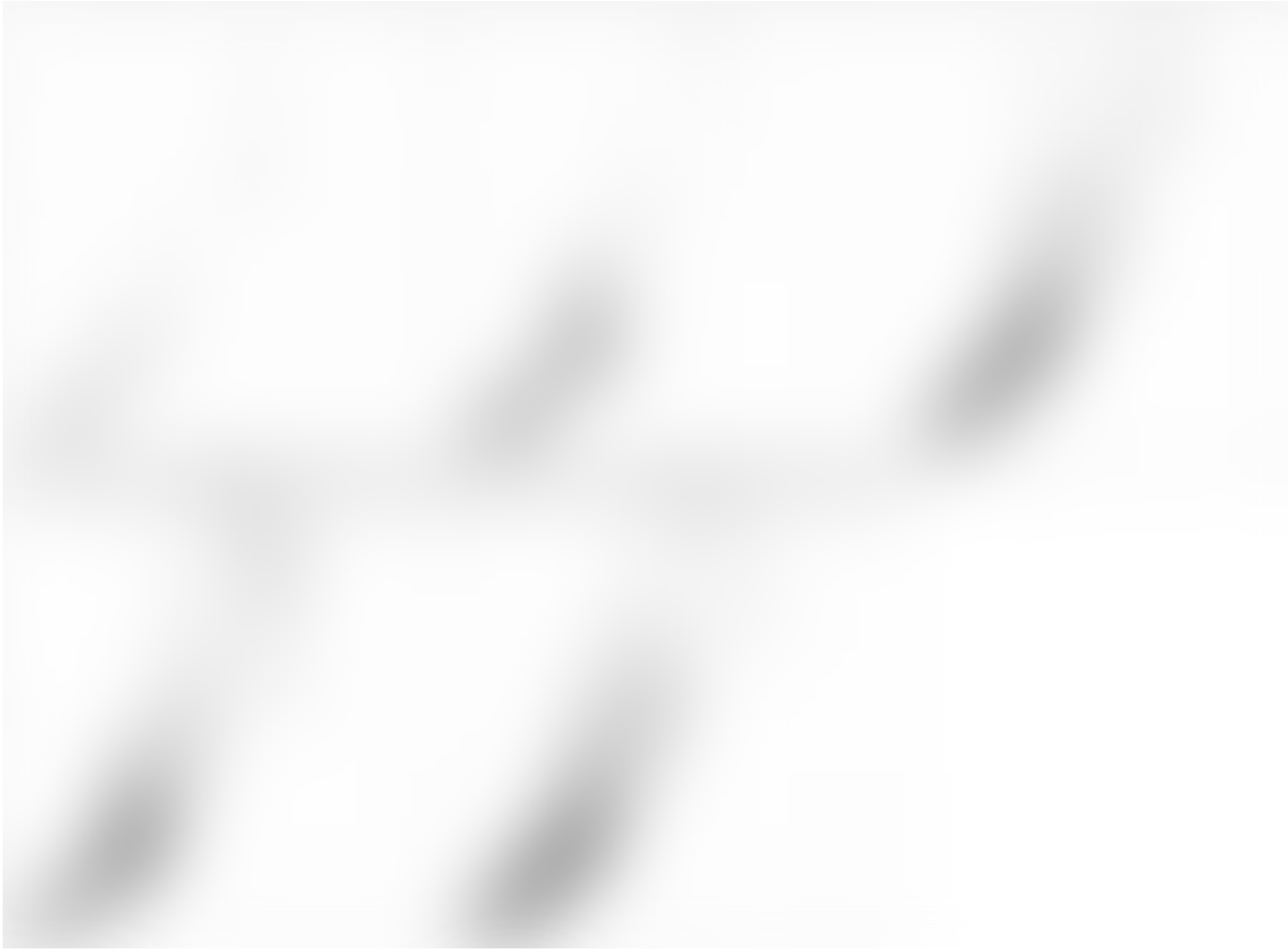
At a basic level, we can view the relationship between the carat of a diamond and its price, which is the main purpose of this dataset:

```
ex2 <-
  diamonds %>%
  sample_n(5000) %>%
  ggplot(aes(x = carat, y = price)) +
  geom_point()

ex2
```

This graph only shows two dimensions of data, but there are a few others are also important. The cut, color, and clarity of the diamond — all of these could are related to the price of the diamond. One way to bring these dimensions in is to have the dots be different colors or use different dot shapes, but let's give faceting a try instead.

Use **facet_wrap()** if you only want to break out the graph by a **single dimension:**



Use **facet_grid()** if you want to break out the graph by **two dimensions:**

```
ex2 +
  facet_grid(color~cut)
```

These are just two examples of how you can use `facet_wrap()` and `facet_grid()`, but the key takeaway from this section is that with ggplot, **you are not constrained to thinking about visualizations in a single graph.**

. . .

3. Colors!

Colors serve two key purposes in data visualization:

1. Makes a visualization more appealing
2. Represents an additional dimension of data

There are many ways to color your ggplots, but for simplicity this section focuses on Viridis palettes, which are my personal favorite as they are:

- **Colorful:** spanning as wide a palette as possible so as to make differences easy to see
- **Perceptually uniform:** meaning that values close to each other have similar-appearing colors and values far away from each other have more different-appearing colors, consistently across the range of values
- **Robust to colorblindness:** so that the above properties hold true for people with common forms of colorblindness, as well as in grey scale printing

Source: https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

You can read more theory behind the colors above **here,** but this section focuses on 4 key functions which allows you to use these colors:

- `scale_color_viridis_d()` & `scale_fill_viridis_d()` : Add this statement to your ggplot in order to color / fill your graph on a **discrete/categorical** variable. *(Notice the "d" at the end of the function)*
- `scale_color_viridis_c()` & `scale_fill_viridis_c()` : Add this statement to your ggplot in order to color / fill your graph on a **continuous** variable. *(Notice the "c" at the end of the function)*

```
# Discrete
ggplot(data = diamonds %>% sample_n(1e3),
         aes(x = carat, y = price,
             color = cut)) +
  geom_point() +
  scale_color_viridis_d()
```

```
# Continuous
ggplot(data = diamonds %>% sample_n(1e3),
       aes(x = carat, y = price,
           color = depth)) +
  geom_point() +
  scale_color_viridis_c(option = 'A')
```

Protip: Here, I'm using the `option` parameter to change the color palette within viridis. You can switch between options A-E which reflect the different color schemes in above.

. . .

4. Color vs. fill: Know the difference

I introduced this in the last section, but I wanted to address it more explicitly because it can be confusing when you first use ggplot. To color a ggplot, you'll either use `color` or `fill` , and this depends on the graph type.

So what's the difference? Generally, `fill` defines the color with which a geom is *filled* (i.e. `geom_bar()`), whereas *color* defines the color with which a geom is *outlined* (i.e. `geom_point()`).

```
ggplot(data = diamonds, aes(x = price)) +
  geom_histogram(color = 'blue',
                 fill = 'red')
```

So the takeaway here is that if you try to color a graph and it appears that nothing has changed, simply switch `color` to `fill` or vice versa.

Read more on StackOverflow

. . .

5. Put a label on it

Good visualizations have concise and descriptive labels. They help readers understand what they are seeing, and this especially important if you expect your visualization to be shared. Luckily it's super easy to label in ggplot.

Below are ggplot's most useful labelling functionalities, listed in how often they should probably be used. You can pick and choose which labels you want to use — for example, if you only want to add a title, you only need to enter the title parameter in `labs()` .

```
ggplot(data = diamonds %>% sample_n(1e3),
       aes(x = carat, y = price, color = cut)) +
  geom_point() +
  labs(title = 'title example',
       x = 'x-axis example',
       y = 'y-axis example',
       color = 'color example',
       subtitle = 'subtitle example',
       caption = 'caption example',
       tag = 'tag example')
```

Note: *The color field is only accessible if you have a color as an aesthetic in your ggplot. This labelling method will also work whether you use fill, color, size, alpha, etc.*

. . .

6. Line annotations

On the theme of telling a story with your visualization, line annotations are a very useful tool. Some examples that I've personally used include:

- Marking a before/after period on a line graph
- Plotting the mean of an x or y value on a scatter plot
- Annotating a goal metric that we want to hit

Whatever the use case, having a line annotation helps communicate an important point to those who will be viewing your visualization. To add a line to your ggplot, you'll use either:

- **geom_hline():** Adds a horizontal line (has a y intercept)
- **geom_vline():** Adds a vertical line (has an x intercept)

The example below will show both of these in action:

```
    geom_point() +
    geom_hline(data = . %>% summarise(y = mean(price)),
               aes(yintercept = y)) +
    geom_vline(data = . %>% summarise(x = mean(carat)),
               aes(xintercept = x))
```

Note that the above code may look a little more complicated than some of the other ggplot code in this article. I'll try to explain what's going on there. In order to get the average carat and price, a more straightforward way to get these values is to calculate them before your ggplot code. However, because I am lazy and like reducing the number of variables that I have, I instead pipe the data (`diamonds %>% sample_n(1e3)`) directly into the `geom_line()` statements, which work just as well.

. . .

7. Text annotations

example, if you blindly apply the text geom, you'll end up with a really ugly graph:

```
p <-
  ggplot(data = diamonds %>% sample_n(1e3),
       aes(x = carat, y = price, color = cut)) +
  geom_point()

p +
  geom_text(aes(label = price))
```

This is bad

In this section, I'll talk about three key tips for using `geom_text()` effectively.

1. **Filtering which labels are shown:** You can get creative with this, but the goal of doing this is to only show relevant data labels. In the case below, I only want to show the prices of high-carat diamonds:

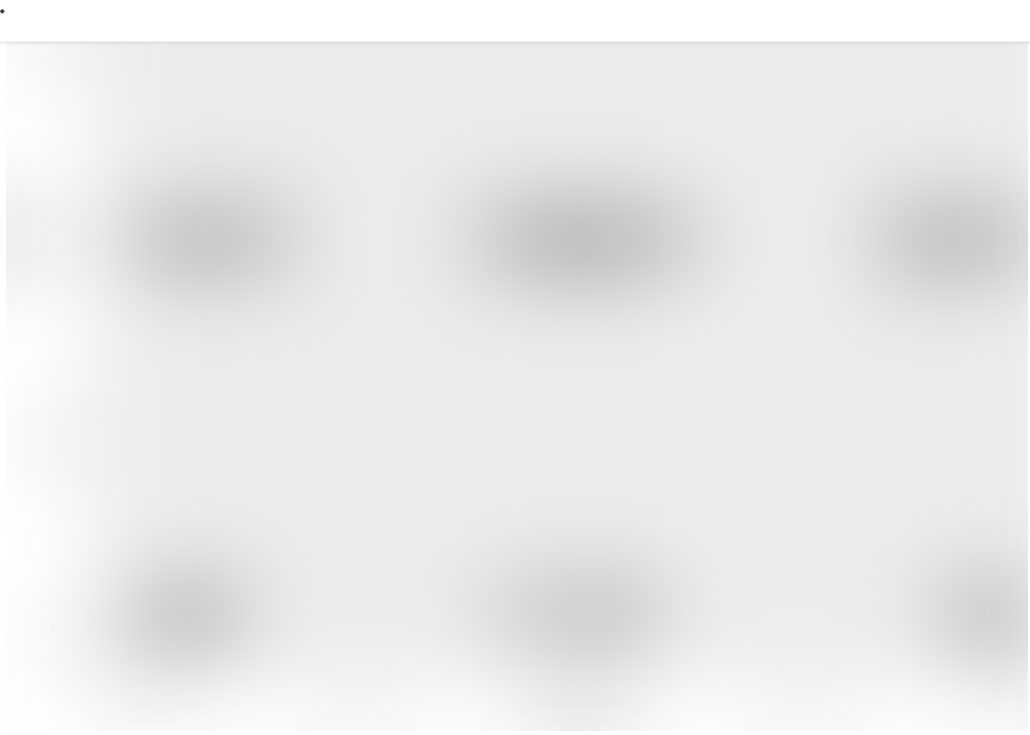
```
aes(label = price))
```


Notice that this is easier to read now, only a few prices are shown

2. hjust + vjust

In the above graph, you'll see that the text completely overlaps the point, which looks ugly. You can easily fix this by aligning your text within geom_text(). The way that I think of this is similar to left and right align in Microsoft Word.

Generally, you'll have vjust and hjust range from [0,1] but it also takes on negative values and values greater than one (it will just move your label further in the specified direction). The graph below shows how text will be aligned based on your hjust and vjust values:



```
p +
  geom_text(data = . %>% filter(carat >= 2.25),
            aes(label = price),
            hjust = 0,
            vjust = 0)
```

3. color

This is more of a preference, but know that you can change the color of your text. You generally want to have your **text contrast as much with the background as possible**, as this makes it the most legible. This is important if you have some lighter colors (i.e. yellow) that may be hard to read:

```
p +
  geom_text(data = . %>% filter(carat >= 2.25),
            aes(label = price),
            hjust = 0,
            vjust = 0,
            color = 'black')
```



```
# Another example where we add contrast
```

```
    ggplot(aes(x = clarity, y = price)) +
    geom_bar(stat = 'identity') +
    geom_text(aes(label = round(price, 2)),
              vjust = 1.25,
              color = 'white')
```

. . .

8. Order, order, order!

Lastly, ordering your graph can make it easier to read, and this is especially useful for bar graphs. All you have to do is use `fct_reorder()` on the x value such that it's sorted by the y-value:

```
# By default, ggplot will order by the x value
```

```
  ggplot(aes(x = clarity, y = price)) +
  geom_bar(stat = 'identity')
```

```
# Reordered:

diamonds %>%
  group_by(clarity) %>%
  summarise(price = mean(price)) %>%
  ggplot(aes(x = fct_reorder(clarity, price), y = price)) +
  geom_bar(stat = 'identity')
```

. . .

Concluding Thoughts

I had a tough time deciding what different topics I wanted to cover in this article. I ended up focusing on topics that were initially confusing to me, and that I wish I understood more when I first started learning ggplot. Hopefully, this article gives you some concrete ideas on how to improve your visualizations or demystifies some of the more confusing/hidden aspects of ggplot.