HW #1

Interpreting {ggplot2} code

Sofia Sarak

Table of contents

Some notes before you get started

- **Be sure to install any packages** in the Setup chunk that you don't already have.
- **Leave the code chunk options, `eval: false` and `echo: true`, set as they are.** The final infographic has been intentionally optimized (e.g., text size, spacing) for saving and viewing as a PNG file, not for display in the Plots pane or within a rendered Quarto document. As a result, the text in each individual ggplot may appear too large when viewed in the Plots pane, but will be correctly sized in the exported PNG. We'll talk more about the nuances of saving ggplots (and why these differences occur) in a later lab section.
- Some answers may become clearer once you've looked ahead at the code further

down in the script. **Consider revisiting questions as you go.**

I. Setup

```r
1  library(colorspace)
2  library(geofacet)
3  library(ggtext)
4  library(glue)
5  library(grid)
6  library(magick)
7  library(patchwork)
8  library(scales)
9  library(showtext)
10 library(tidyverse)
11
12 ufo_sightings <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/mai
13 places <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/main/data/
14
15 alien <- c("#101319", "#28ee85")
16 bg <- alien[1]
17 accent <- alien[2]
18
19 ufo_image <- magick::image_read(path = here::here("images", "ufo.png"))
20
21 sysfonts::font_add_google(name = "Orbitron", family = "orb")
22 sysfonts::font_add_google(name = "Barlow", family = "bar")
23
24 sysfonts::font_add(family = "fa-brands", regular = here::here("fonts", "Font Awesome 6 Brand
25 sysfonts::font_add(family = "fa-solid", regular = here::here("fonts", "Font Awesome 6 Free-S
26
27 showtext::showtext_auto(enable = TRUE)
```

1. **What is the author defining in lines 15-17? Where else in the code do these defined variables show up? What advantage(s) is there to defining these values here, as variables, rather than defining the values directly throughout the script?**

    - The author is defining specific colors using hex codes, and saving them together as a vector under `alien` and then each as their own variable in lines 16 and 17. The advantage to this is that since these are colors that will be used repeatedly in the visualization (with arguments like `fill=`), it is much easier to reference them as variables instead of finding and copying the hex codes (or memorizing them?) each time.

2. **In your own words, explain what the function, `font_add_google()`, does. What's the difference between the two arguments, `name` and `family`?**

   - The function `font_add_google()` imports specific fonts from the Google Fonts repository into `sysfonts`, which provides access to installed fonts within this R session. The argument `name` specifies the text that will be searched in the repository to access the desired font, while "family" can be any string (doesn't have to match `name`) that will be used to specify the font later on in R plotting.

II. Data wrangling

i. Create `df_pop`

```r
df_pop <- places |>
  filter(country_code == "US") |>
  mutate(state = str_replace(string = state,
                             pattern = "Fl",
                             replacement = "FL")) |>
  group_by(state) |>
  summarise(pop = sum(population)) |>
  ungroup()
```

3. **Describe what this data frame contains.**

   - This data frame contains two columns: one with all of the U.S. states, identified by their abbreviations (stored as character strings), and the other with the corresponding state population (a numeric variable).

ii. Create `df_us`

```r
df_us <- ufo_sightings |>
  filter(country_code == "US") |>
  mutate(state = str_replace(string = state,
                             pattern = "Fl",
                             replacement = "FL")) |>
  count(state) |>
  left_join(df_pop, by = "state") |>
  rename(num_obs = n) |>
  mutate(
    num_obs_per10k = num_obs / pop * 10000,
    opacity_val = num_obs_per10k / max(num_obs_per10k)
    )
```

4. **Describe what this data frame contains.**

    - This data contains 5 columns. Like `df_pop`, `df_us` also has a column of state abbreviations and one containing state populations. In addition to this, this dataframe also has the number of UFO observations in each state as its third column. Using state populations and UFO sightings, the number of UFO observations per 10,000 people (`num_obs_per10k`) was calculated and saved as its own column. Finally, `opacity_val`, the fifth column, contains the fraction that each `num_obs_per10k` value is of the maximum `num_obs_per10` value.

5. **What does `opacity_val` represent, and why is it calculated?**

    - `opacity_val` represents the proportion that `num_obs_per10k` is of its maximum value. This was calculated in order to have an opacity value to assign to the `alpha=` argument in plotting. By doing it in this way, it ensures that the higher `num_obs_per10k` values would appear darker and lower ones lighter. This makes intuitive sense, as we would expect something with a larger value to be presented as more concentrated.

iii. Create `df_shape`

```r
df_shape <- ufo_sightings |>
  filter(!shape %in% c("unknown", "other")) |>
  count(shape) |>
  rename(total_sightings = n) |>
  arrange(desc(total_sightings)) |>
  slice_head(n = 10) |>
  mutate(
    shape = fct_reorder(.f = shape,
                        .x = total_sightings),
    opacity_val = scales::rescale(x = total_sightings,
                                  to = c(0.3, 1))
    )
```

6. **Describe what this data frame contains.**

- This data frame contains the top 10 most common UFO shapes sighted, with a column for total sightings (in descending order) and one with opacity value. `opacity_val` was assigned to UFO shape using the rescale function from the scales package, which applies a proportional value, from 0.3 to 1, to each number of sightings based on its position between the maximum and minimum. In total, there are three columns.

7. **What does `fct_reorder` do when it is applied to the `shape` variable? What would have happened if this step was not performed?**

   - When applied to the `shape` variable, the `fct_reorder` function specifies that `shape` will be ordered by the `total_sightings` variable moving forward. Although the data frame was already arranged by `total_sightings` in descending order, this step ensures that during plotting the factors are actually plotted in the order we want them, instead of alphabetically (the default).

8. **What is the purpose of rescaling `opacity_val`? And why rescale from 0.3 to 1?**

   - Rescaling `opacity_val` translates the `total_sightings` column into values that the alpha argument (which comes up in plotting) comprehends. Rescaling from 0.3 to 1 is important, as opposed to 0 to 1 (which is the acceptable range of values for alpha), because otherwise the shape with the lowest value would have no opacity at all (alpha = 0). Specifying 0.3 to 1 ensures that the lighest value is still dark enough to discern.

iv. Create `df_day_hour`

```r
df_day_hour <- ufo_sightings |>
  mutate(
    day = wday(reported_date_time),
    hour = hour(reported_date_time),
    wday = wday(reported_date_time, label = TRUE)
  ) |>
  count(day, wday, hour) |>
  rename(total_daily_obs = n) |>
  mutate(
    opacity_val = total_daily_obs / max(total_daily_obs),
    hour_lab = case_when(
      hour == 0 ~ "12am",
      hour <= 12 ~ paste0(hour, "am"),
      hour == 12 ~ "12pm",
      TRUE ~ paste0(hour - 12, "pm"))
    )
```

9. **Describe what this data frame contains.**

   - This data frame has six columns. The first two are the days of the week, categorized both as an integer (`day`, where 1 represents Sunday) and character string abbreviation (`wday`, where Sun represents Sunday). There are also two columns that contain each hour of the day. Similarly to the days of the weeks column, one (`hour`) is an integer of the hour, and the other (`hour_lab`) is a character string with a digit and "am" or "pm". Finally, each hour of each day also has a corresponding value in the `total_daily_obs` column, representing observations within that specific time frame.

10. **What is the purpose of the last line inside the `case_when()` statement (`TRUE ~ paste0(hour - 12, "pm")`)?**

    - The last line in the `case_when()` statement defines the default value when none of the previously defined conditions are meant. In this case, it tells the `case_when()` function that if the hour is *not* 0, less than 12, or equal to 12, then to subtract 12 from the value and add "pm" to the label.

III. Prepare text elements

```r
quotes <- paste0('"...', str_to_sentence(ufo_sightings$summary[c(47816, 6795, 93833)]), '...'

original <- glue("Original visualization by Dan Oehm:")
dan_github <- glue("<span style='font-family:fa-brands;'>&#xf09b;</span> doehm/tidytues")
new <- glue("Updated version by Sam Shanny-Csik for EDS 240:")
link <- glue("<span style='font-family:fa-solid;'>&#xf0c1;</span> eds-240-data-viz.github.io
space <- glue("<span style='color:{bg};'>. .</span>")
caption <- glue("{original}{space}{dan_github}
                <br><br>
                {new}{space}{link}")
```

11. **In your own words, what is the difference between `paste0()` and `glue()`? Why did the author use `paste0` to construct `quotes` and `glue` to construct the other text elements?**

    - `paste0` interprets R syntax whereas `glue` can interpret HTML, which allows for easier customization. `paste0` was used to construct `quotes` because it was extracting data from a data frame, perhaps making using R syntax (such as for selecting out of said data frame) necessary. The other elements were constructed with `glue` because they had specifications like font, background color, and spacing that needed to be written in HTML format.

IV. Build plots

i. Build `plot_shape`

```r
plot_shape <- ggplot(data = df_shape) +
  geom_col(aes(x = total_sightings, y = shape, alpha = opacity_val),
           fill = accent) +
  geom_text(aes(x = 200, y = shape, label = str_to_title(shape)),
            family = "orb",
            fontface = "bold",
            color = bg,
            size = 14,
            hjust = 0,
            nudge_y = 0.2) +
  geom_text(aes(x = total_sightings-200, y = shape, label = scales::comma(total_sightings)),
            family = "orb",
            fontface = "bold",
            color = bg,
            size = 10,
            hjust = 1,
            nudge_y = -0.2) +
  scale_x_continuous(expand = c(0, NA)) +
  labs(subtitle = "10 most commonly reported shapes") +
  theme_void() +
  theme(
    plot.subtitle = element_text(family = "bar",
                                 size = 40,
                                 color = accent,
                                 hjust = 0,
                                 margin = margin(b = 10)),
    legend.position = "none"
  )
```

12. **Explain the values provided to the x aesthetic for both text geoms (`shape` & `total_sightings`).**

    - The values provided to the `x` aesthetic represent the x coordinates at which the `geom_text()` objects are to be located. In the first text geom (`shape`) the labels are specified to exist at the x-coordinate 200, whereas in the second (`total_sightings`), the label is to be 200 x-coordinates to the left of each `total_sightings` columns. In other words, `shape`'s position was defined in absolute terms, while `total_sightings`'s in relative terms.

ii. Build `plot_us`

HINT: Consider temporarily commenting out / rearranging the `geom_*()` layers to better understand how this plot is constructed

```r
plot_us <-  ggplot(df_us) +
  geom_rect(aes(xmin = 0, xmax = 1, ymin = 0, ymax = 1, alpha = opacity_val),
            fill = accent) +
  geom_text(aes(x = 0.5, y = 0.7, label = state),
            family = "orb",
            fontface = "bold",
            size = 9,
            color = bg) +
  geom_text(aes(x = 0.5, y = 0.3, label = round(num_obs_per10k, 1)),
            family = "orb",
            fontface = "bold",
            size = 8,
            color = bg) +
  geofacet::facet_geo(~state) +
  coord_fixed(ratio = 1) +
  labs(subtitle = "Sightings per 10k population") +
  theme_void() +
  theme(
    strip.text = element_blank(),
    plot.subtitle = element_text(family = "bar",
                                 size = 40,
                                 color = accent,
                                 hjust = 1,
                                 margin = margin(b = 10)),
    legend.position = "none"
  )
```

13. **Consider the order of `geom_*()` layers in the the above plot (`plot_us`). Why did the author order the layers in this way?**

    - In this plot, `geom_rect` is first because it acts as the background for the `geom_text` labels. By putting it first, the author ensures that it gets plotted first, and therefore the labels get plotted on top of it, as the second and third steps.

iii. Build `plot_day`

```r
plot_day <- ggplot(data = df_day_hour) +
  geom_tile(aes(x = hour, y = day, alpha = opacity_val),
            fill = accent,
            height = 0.9,
            width = 0.9) +
  geom_text(aes(x = hour, y = 9, label = hour_lab),
            family = "orb",
            color = accent,
            size = 10) +
  geom_text(aes(x = 0, y = day, label = str_sub(string = wday, start = 1, end = 1)),
            family = "orb",
            fontface = "bold",
            color = bg,
            size = 8) +
  ylim(-5, 9) +
  xlim(NA, 23.55) +
  coord_polar() +
  theme_void() +
  theme(
    plot.background = element_rect(fill = bg, color = bg),
    legend.position = "none"
  )
```

14. **This plot includes one-letter labels for each day of the week. How is ths accomplished when week days are written using their three-letter abbreviations (e.g. `Mon`, `Tue`) in the `df_day_hour` data frame?**

    - This is accomplished through the function `str_sub`, which takes the original `wday` string, and (according to the arguments in this plot), keeps only the first character in the string. This is done by assigning both the start and end of the character range to 1.

15. **What role do the `ylim()` and `xlim()` functions play in shaping a ggplot, and how do they change the visual layout of this particular plot? To better understand their effect, try rerunning the code with each of these lines commented out and observe how the plot's spacing and composition change.**

    - Generally, `ylim()` and `xlim()` change the y- and x-axis limits of a ggplot object. In this case, a y-axis from -5 to 9 creates a blank circle/hole in the middle of the plot, and "squishes" each data box to be a bit narrower than before. The x-axis limit of

23.55 puts a gap between 11pm and 12am, in order to match the gaps in the rest of the plot.

iv. Build `quote*s`

A comment from Dan Oehm's original code: "A bit clunky but the path of least resistance."

```r
quote1 <- ggplot() +
  annotate(geom ="text",
           x = 0,
           y = 1,
           label = str_wrap(string = quotes[1], width = 40),
           family = "bar",
           fontface = "italic",
           color = accent,
           size = 16,
           hjust = 0,
           lineheight = 0.4) +
  xlim(0, 1) +
  ylim(0, 1) +
  theme_void() +
  coord_cartesian(clip = "off")

quote2 <- ggplot() +
  annotate(geom = "text",
           x = 0,
           y = 1,
           label = str_wrap(string = quotes[2], width = 25),
           family = "bar",
           fontface = "italic",
           color = accent,
           size = 16,
           hjust = 0,
           lineheight = 0.4) +
  xlim(0, 1) +
  ylim(0, 1) +
  theme_void() +
  coord_cartesian(clip = "off")

quote3 <- ggplot() +
  annotate(geom = "text",
           x = 0,
           y = 1,
           label = str_wrap(string = quotes[3], width = 25),
           family = "bar",
           fontface = "italic",
           color = accent,
           size = 16,
           hjust = 0,
           lineheight = 0.4) +
  xlim(0, 1) +
  ylim(0, 1) +
  theme_void() +
  coord_cartesian(clip = "off")
```

16. **Why do you think the author chose to convert these text elements (and also in `plot_ufo`, below!) into ggplot objects (you may consider returning to this question after you've worked your way through all of the code)?**

    - By having all the visual components (plots, quotes, and images) as the same type of object, it is easier to combine them using the same commands within R.

v. Build `plot_ufo`

Note: Grob stands for **gr**aphical **ob**ject. Each visual element rendered in a a ggplot (e.g. lines, points, axes, entire panels, even images) is represented as a grob. Grobs can be manipulated individually to fully customize plots.

```r
plot_ufo <- ggplot() +
  annotation_custom(grid::rasterGrob(ufo_image)) +
  theme_void() +
  theme(
    plot.background = element_rect(fill = bg, color = bg)
  )
```

vi. Build `plot_base`

```r
plot_base <- ggplot() +
  labs(
    title = "UFO Sightings",
    subtitle = "Summary of over 88k reported sightings across the US",
    caption = caption
    ) +
  theme_void() +
  theme(
    text = element_text(family = "orb",
                        size = 48,
                        lineheight = 0.3,
                        color = accent),
    plot.background = element_rect(fill = bg,
                                   color = bg),
    plot.title = element_text(size = 128,
                              face = "bold",
                              hjust = 0.5,
                              margin = margin(b = 10)),
    plot.subtitle = element_text(family = "bar",
                                 hjust = 0.5,
                                 margin = margin(b = 20)),
    plot.caption = ggtext::element_markdown(family = "bar",
                                            face = "italic",
                                            color = colorspace::darken(accent, 0.25),
                                            hjust = 0.5,
                                            margin = margin(t = 20)),
    plot.margin = margin(b = 20, t = 50, r = 50, l = 50)
  )
```

17. **Why does the author render `plot.caption` using `ggtext::element_markdown()`, rather than `element_text()` (like he does for rendering `plot.title` and `text`)?**

    - The author uses `ggtext::element_markdown()` instead of `element_text()` because the caption was defined using the `glue()` function, and styling is specified using HTML. `ggtext::element_markdown()` can interpret HTML, whereas `element_text()` cannot. In this case, this is important is to preserve the spacing of the caption, for example. The title and subtitle, on the other hand, are entered as simple strings, and can be rendered with `element_text`.

V. Assemble & save

```
plot_final <- plot_base +
  inset_element(plot_shape, left = 0, right = 1, top = 1, bottom = 0.66) +
  inset_element(plot_us, left = 0.42, right = 1, top = 0.74, bottom = 0.33) +
  inset_element(plot_day, left = 0, right = 0.66, top = 0.4, bottom = 0) +
  inset_element(quote1, left = 0.5, right = 1, top = 0.8, bottom = 0.72) +
```

```r
  inset_element(quote2, left = 0, right = 1, top = 0.52, bottom = 0.4) +
  inset_element(quote3, left = 0.7, right = 1, top = 0.2, bottom = 0) +
  inset_element(plot_ufo, left = 0.25, right = 0.41, top = 0.23, bottom = 0.17) +
  plot_annotation(
    theme = theme(
      plot.background = element_rect(fill = bg,
                                     color = bg)
    )
  )

ggsave(plot = plot_final,
       filename = here::here("outputs", "ufo_sightings_infographic.png"),
       height = 16,
       width = 10)
```

18. **Explain how `plot_final` is assembled. What do you think is the most challenging aspect of arranging all components into a single plot?**

    - Each element of the final visual – the three plots and three quotes – were all manually inset onto the `plot_base` (which contains the title, subtitle, and caption) by specifying the locations of the outer bounds (left, right, top, and bottom). The UFO image in the middle of `plot_day` is also manually inset. The function `inset_element()` comes from the `patchwork` package. The most challenging aspect of arranging all of the components must be figuring out the proper placement using the argument units. I imagine that it would take a lot of fiddling and re-running the code to get all of the components to be situated exactly where you want them.

19. **Can you think of one reason the author may have chosen to separate the construction of `plot_base` and `plot_final`?**

    - By creating `plot_base` first and treating it as its own separate object, the author gives themself the freedom to alter details such as text size, position, and color (of the title, subtitle, and caption), without having to rerun or be afraid of affecting `plot_final` each time. Additionally, `plot_base` is defined using many lines of code. Constructing `plot_base` and `plot_final` together in one code chunk could potentially reduce readibility and comprehension to an outside audience.

Answer some final reflective questions

20. **During week 2, we discuss Choosing the right graphic form. Refer to this lecture when answering the sub-questions, below:**

a. **What "perceptual tasks" (from Cleveland & McGill's heirarchy) must the viewer perform to extract information from these visualizations?**

   - Plot 1: Measure bar lengths relative to each other;
   - Plot 2: Interpret opacity and spatial position (states *are* labeled though, making it a bit easier);
   - Plot 3: Interpret opacity, as well as keep track of both the time and day variables (a little difficult with the radial plot)

b. **What task(s) do you think the author wanted to enable or message(s) he wanted to convey with these visualizations (see lecture 2.1, slide 16 for examples)? Be sure to note at least one task / message for each of the three data viz.**

   - Plot 1: The author wanted to convey big picture trends, such as that – regardless of time or place – the most common shape of UFO spotted is "light"
   - Plots 2 and 3: The author wanted to explore temporal and spatial patterns of UFO sightings in the US. Both of these plots also enable comparison between different places and times of day.

c. **Name at least one caveat to the "hierarchy of perceptual tasks" that the author employed to achieve a goal(s) you noted in question b?**

   - The author preferred to use shades (`opacity_val`) to convey number of sightings (and explore spatial and temporal patterns), which sits pretty low on the hierarchy.

21. **Describe two elements of this piece that you find visually-pleasing / easy to understand / intuitive. Why?**

    - I liked how the barplot labels were embedded/plotted on top of the actual bar. I also liked the use of the coord_polar/radial plot to convey time of day. Although it doesn't follow an actual clock (because it has 24 sections instead of 12), it was still inuitive, perhaps because we often think of time as the sun's rotation around the globe.

22. **Describe two elements of this piece that you feel could be better presented in a different way. Why?**

    - 1. The squares of the US map are visually appealing, but it does make it makes it more difficult to actually discern which state is which based on common knowledge of the United States map (as in, they're not exactly where they are truly located).
    - 2. The way the day of the week variable was presented in the third figure isn't very intuitive. From that plot I mostly gathered that sightings happened in the early morning, and very little about day of the week (at least at first glance).

23. **Describe two new things that you learned by interpreting / annotating this code. These could be packages, functions, or even code organizational approaches that you hadn't previously known about or considered.**

    - I really liked the opacity value approach; I didn't even know you could change alpha based on a variable.
    - I also always thought that coord_polar wasn't a very useful function, but I see now that it can be a visually-appealing and perhaps even intuitive way of depicting time. I also didn't know that you could increase the size of the hole in the middle simply by using `ylim()`.

24. **How, if at all, did you use AI tools to help you interpret this code? Describe your approach to using these tools for this assignment. In what ways was consulting the documentation more (or less) helpful than using AI?**

    - I tried to avoid AI tools while interpreting this code, though it did inadvertently come up when I Google-searched for a function, and the AI overview popped up at the top of my results. If there were sources that seemed useful (from Stack Overflow, for example) I would opt to use those instead of using the overview. I will say that checking function documentation (internally in R) was always my first step, and for most simple functions it was more clear than AI, because it clearly lists all of the possible arguments and their uses.