HW #1

Interpreting `{ggplot2}` code

Melannie Moreno Rolón

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.
- **Questions that reference line numbers (e.g., Question #1) refer to the**

line numbers shown in the rendered code chunks. You will need to render the document to view these line numbers.

- 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

```
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 Brands
25 sysfonts::font_add(family = "fa-solid", regular = here::here("fonts", "Font Awesome 6 Free-So
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?**

- In line 15, the author defines a color palette by creating a vector and saving it as an object named `alien`. In lines 16 and 17, the background color and accent color are defined as the variables `bg` and `accent`. Defining these values early in the script is helpful because when the author later specifies theme elements or color scales, they can reference these variables instead of re-writing the hex codes each time. This practice avoids redundancy throughout the script. As a result, the script becomes easier to read and to maintain.

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 `font_add_google()` function searches for a font in the Google Fonts repository and downloads it so it can be used within the sysfonts package. The `name` argument refers to the official Google Fonts name, while the `family` argument is the custom name the user assigns so the font can be referenced later in plotting functions.

II. Data wrangling

i. Create `df_pop`

```
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 total population estimates for each state in the United States. The `state` column includes state abbreviations, and the `pop` column stores each state's corresponding total population that aggregated from the original places dataset.

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 frame contains, for each U.S. state, the total number of UFO observations, the total population estimate, the number of UFO sightings per 10,000 people, and a calculated opacity value that will later be used for visualization.

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

    - The `opacity_val` variable represents a normalized transparency value based on the number of UFO sightings per 10,000 people in each state. It is calculated so that states with higher relative ufo sightings appear more opaque in the visualization and emphasize differences across states.

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 most commonly reported UFO shapes. The `shape` column lists the shape categories, `total_sightings` contains the number of sightings for each shape, ordered from most to least frequent, and `opacity_val` stores a rescaled transparency value for use in plotting.

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

   - The `fct_reorder()` function reorders the categorical values in the shape variable based on the values in the `total_sightings` column. This ensures that the shapes appear in an order that reflects their frequency rather than alphabetical order. Without this step, the plot would display the shapes in a less meaningful order.

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

   - The purpose of rescaling `opacity_val` from 0.3 to 1 is to ensure that even the shape with the fewest ufo sightings remains visible. Setting a minimum opacity prevents categories from becoming fully transparent while still preserving visual differences in magnitude.

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 summarizes ufo sightings by day of the week and hour of the day. The `day` column represents the numeric weekday, `wday` contains the labeled weekday abbreviation, and `hour` ranges from 0 to 23. The `total_daily_obs` column records the number of sightings for each day-hour combination. The final two columns include `opacity_val`, which scales transparency by frequency, and `hour_lab`, which converts numeric hours into readable am/pm labels.

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

    - This line converts all hours after 12 into the correct "pm" label by subtracting 12 from the hour value and appending "pm" and makes sure the time labels are displayed in a familiar 12-hour format.

III. Prepare text elements

```
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?**

    - The `paste0()` function concatenates the character strings without inserting spaces and it works well for vectorized text operations. The `glue()` function allows variables to be embedded directly within strings using curly brackets and makes it more readable for constructing formatted text. The author uses `paste0()` to build the quotes vector, while `glue()` is used for the caption text because it integrates variables and HTML elements more cleanly.

IV. Build plots

i. Build `plot_shape`

```
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`).**

    - In `geom_col()`, the x aesthetic maps `total_sightings` to create horizontal bars. In the first `geom_text()`, the x value is set to 200 to position the shape labels near the start of each bar. In the second `geom_text()`, "total_sightings - 200" positions the numeric labels near the end of each bar but slightly inset. Doing this prevents overlap with the plot boundary.

ii. Build `plot_us`

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

```
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?**

    - The author wanted to place the rectangle shape as a background with geom_rect(). Then, he used the two geom texts to draw the state abbreviation and the number of sightings on top of the background created with geom_rect(). If these functions had been ordered any other way, the geom texts would not be visible.

iii. Build `plot_day`

```
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 this accomplished when week days are written using their three-letter abbreviations (e.g. `Mon`, `Tue`) in the `df_day_hour` data frame?**

    - Within the second 'geom_text()' plotting function, the author specified the location of the one-letter labels under the labels argument using the `str_sub()` function. The `str_sub()` function is used here to substitute the three-letter abbreviation in the `df_day_hour` data frame by inputting the character vector 'wday' and setting the start and end arguments to 1 to establish the number of characters to extract.

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.**

    - The `ylim()` and `xlim()` functions define the visible boundaries of the y-axis and x-axis. In this plot, adjusting these limits increases the available plotting space, allowed room for labels and improved the overall balance of the layout. Removing

these limits changes the spacing and can cause elements to appear cramped or misaligned.

iv. Build `quote*s`

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

```
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)?

```
- The author chose to convert the text objects into ggplot objects to later arrange these obj
```

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.

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

vi. Build `plot_base`

```
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)),
    # element_markdown() -> renders HTML formatting in text
    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 application of the `ggtext::element_markdown` function renders the formatted HTML tags in the `caption` text object.

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) +
  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?**

    - The plot is first assembeled by setting the base ggplot object that the author created. After the base is established, the author then inserted the plots and text objects within this base object. I thought the most challenging aspect to arranging all of the components was to determine the most appropriate numbers for the location of the objects in the 'left', 'right', 'top', and 'bottom' arguments in the `inset_element` function.

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

- One possible reason for why the author chose to do this is because it is easier to customize the location of your objects within the base and avoid extensive lines of code that would be harder to troubleshoot if an error was present in the code.

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?**

       - The viewer has to judge the position of the objects, as well as perceive angles and areas to extract the information that is presented in visualizations.

    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.**

- I think the author used the ordered bar plot to compare the categories of shapes that get reported in ufo sightings. In the plot displaying the number of ufo sightings per 10,000 inhabitants, the author might have wanted to display the spatial distribution of the number of ufo sightings and have the viewer be able to easily compare them geographically. In the last visualization, the task might have been to display the temporal pattern.

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 relies on color saturation, which ranks lower in the perceptual hierarchy. While this can reduce precision, it is used effectively here to convey magnitude while maintaining visual cohesion across the infographic.

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

- The geofaceted layout makes state-level comparisons intuitive and avoids distortion from traditional maps. The use of `coord_polar()` to represent time feels familiar, similar to reading a clock, which makes the temporal pattern easier to understand.

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

- Lower-opacity values are harder to see, requiring zooming to read details. Additionally, the final temporal visualization lacks a title or caption, which could help guide interpretation.

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 learned about the `geofacet` package and how it can be used to display geographic data without a traditional map. I also learned how multiple ggplot objects can be inserted into a base plot, which was a new organizational approach for me.

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 did not use AI tools to interpret the code itself. I relied on documentation and examples. I found the documentation to be more efficient and precise for understanding function behavior.