HW #1

Interpreting `{ggplot2}` code

Sarayu Ramnath

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

```
#loading library
library(colorspace)
library(geofacet)
library(ggtext)
library(glue)
library(grid)
library(magick)
library(patchwork)
library(scales)
library(showtext)
library(tidyverse)

#loading two data sets online using read_csv: sightings and places
ufo_sightings <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/mai
places <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/main/data/

#defining color variables: bg is background color, accent is accent color
alien <- c("#101319", "#28ee85")
bg <- alien[1]
accent <- alien[2]

#reading in an ufo image using magick package
ufo_image <- magick::image_read(path = here::here("images", "ufo.png"))

#adding google fonts using showtext package
sysfonts::font_add_google(name = "Orbitron", family = "orb")
sysfonts::font_add_google(name = "Barlow", family = "bar")

#adding font awesome fonts using showtext package
sysfonts::font_add(family = "fa-brands", regular = here::here("fonts", "Font Awesome 6 Brands
sysfonts::font_add(family = "fa-solid", regular = here::here("fonts", "Font Awesome 6 Free-S

#using showtext for font rendering from now on
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?

- Oehm is defining color variables that will be used throughout the script.
  - "alien" hosts the color pallette, which is those two colors.
  - "bg" is the background color (1st alien color listed)
  - "accent" is the highlight or accent color (2nd alien color listed).
- It looks like we will see this variables as these will be called for the plot background, theme settings, and the aes settings in ggplot.
- Defining the values as variables in the beginning makes sure there is consistency across the plots because using the same variable makes sure the same colors are applied everywhere. You are less likely to make typos or have inconsistencies than if you were manually inputting the hex codes. That means too, it is easy to update. You can update the colors values in one place (beginning of script) instead of searching and replacing throughout the script.

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

- font_add_google() function will search the google fonts repository and load it into R so the fonts can be used in plots and graphics. We can then use cool fonts for the titles, labels, and annotations.
- "Name" functions identifies the official name of the font to download, but "family" is the nickname you choose to refer to the font in your R code. Like if you downloaded "Orbitron" from google fonts, that's the name. I won't type orbitron every time when I refer to the font in the theme, I might call it "orb" for short, and that would be the family name, the alias.

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.**

    - The data frame describes U.S. states by abbreviation and lists the population size of each state.

ii. Create `df_us`

```
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 begins to summarize the number of reported UFO sightings in each state, and it looks like it is being standardized by population size as it is categorized by sightings per 10k people.

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

    - "opacity_val" would show the transparency in the abundance visualization. So the sightings are standardized per 10k people by dividing each values by the maximum. That gives a scale of 0-1 which would control transparency in the visualization and highlight states with relatively higher sighting rates. States with higher sighting rates would be more opaque and states with lower sightings are more transparent.

iii. Create `df_shape`

```
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 summarized total sightings by UFO shapes. We also get an opacity value based on the total sightings.

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

    - "fct_reorder" reorders the levels of the factor shape based on the total_sightings. It reordered the UFO shape categories from most frequently reported to least.
    - if fct_reorder() was not used the shapes would appear on the plot in the order they would appear in the data frame. It would likely be harder to see which shapes are most or least common.

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

    - rescaling opacity_val would rescale the sighting frequency of the shapes to a more visible range, so we would still be able to interpret less frequent shapes but emphasizing more common ones.

iv. Create `df_day_hour`

```
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.**

    - df_day_hour data frame summarized the UFO sightings by the day of the week and hour of the day so we can see when sightings occur.
    - opacity_val shows relative frequency of sightings on a given day and hour. More frequent times are opaque, less frequent transparent. So we can see peak UFO activity times.

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

    - The time was 24 hour numeric hour and this conditional statement turns the numeric hour into an am/pm label.

III. Prepare text elements

Preparing text elements by building formatted text for captions and annotations, including font icons, line breaks and colors.

```
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() sticks the pieces of text together (concatenating the strings in technical terms). It's very manual, no added formatting, spacing, or evaluating variables unless you tell it to. It is literally just joining strings of text end of end, which is fine if it is as simple as that.
- glue() allows variables to be embedded directly in the string, so we can get more complex and multi-line formatting.
- Oehm used paste0() for the quotes because it was stringing together text and glue() for the captions because there were html icons, and variables that needed to be inserted 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 the first text geom, x= 200 is a manual value to place the shape label at a fixed position to the left of the bars.
    - In the second text geom, x= total_sightings - 200 puts the numeric counts by the end of each bar.
    - This is a manual distinction to make the labels align alongside the bars.

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) + #background
  geom_rect(aes(xmin = 0, xmax = 1, ymin = 0, ymax = 1, alpha = opacity_val),
            fill = accent) +
  #this draws a rectangle that fills the facet space for each state. Looks like the backgrou
  geom_text(aes(x = 0.5, y = 0.7, label = state), #labels
            family = "orb",
            fontface = "bold",
            size = 9,
            color = bg) +
  #puts state abbreviation on top of rectangle. Place after rectangle shows it is visible ab
  geom_text(aes(x = 0.5, y = 0.3, label = round(num_obs_per10k, 1)),
            family = "orb",
            fontface = "bold",
            size = 8,
            color = bg) +
  #seeing the numeric values of sightings per 10k. it needs to come after the rectangle so i
  geofacet::facet_geo(~state) + #creates the layout by state, so each rectangle and text lab
  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 order of geom_*() in ggplot matters because the layers are built in the order they are written, with the earlier layers being under the later layers. The order is from background to labels to numeric labels.
      - We get background first (geom_rect) sets up the canvas.
      - Labels come next to be on the top and readable so it is not hidden behind the rectangle.
      - Numeric labels are last and the counts can appear clearly over the background.

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?**

    - Line used: label = str_sub(string = wday, start = 1, end = 1)
    - The plot uses str_sub() to take the first character of the three-letter weekday abbreviation (wday) so that the first letter of each weekday is kept.

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.**

    - Without setting the ylim() and xlim() a lot of the elements in the plot would overlap or get cut off, especially because this is a polar plot and all the elements are arranged in a circle.
    - Setting the ylim() and xlim() makes sure all the text and tiles fit within the circular layout without overlapping.

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)?**

    - I think the quote and also the graphic below is turned into its own ggplot object so the text can be treated like any other plot element and then can be formatted easier in its positioning and layout.

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`)?**

- I still don't fully understand. But what I can get the gist of is that element_markdown() is used because the caption contains HTML and markdown that needs to be parsed before rendering. But the title and subtitle are plain text. There isn't any icons, or line breaks or inline styling like in the caption.

V. Assemble & save

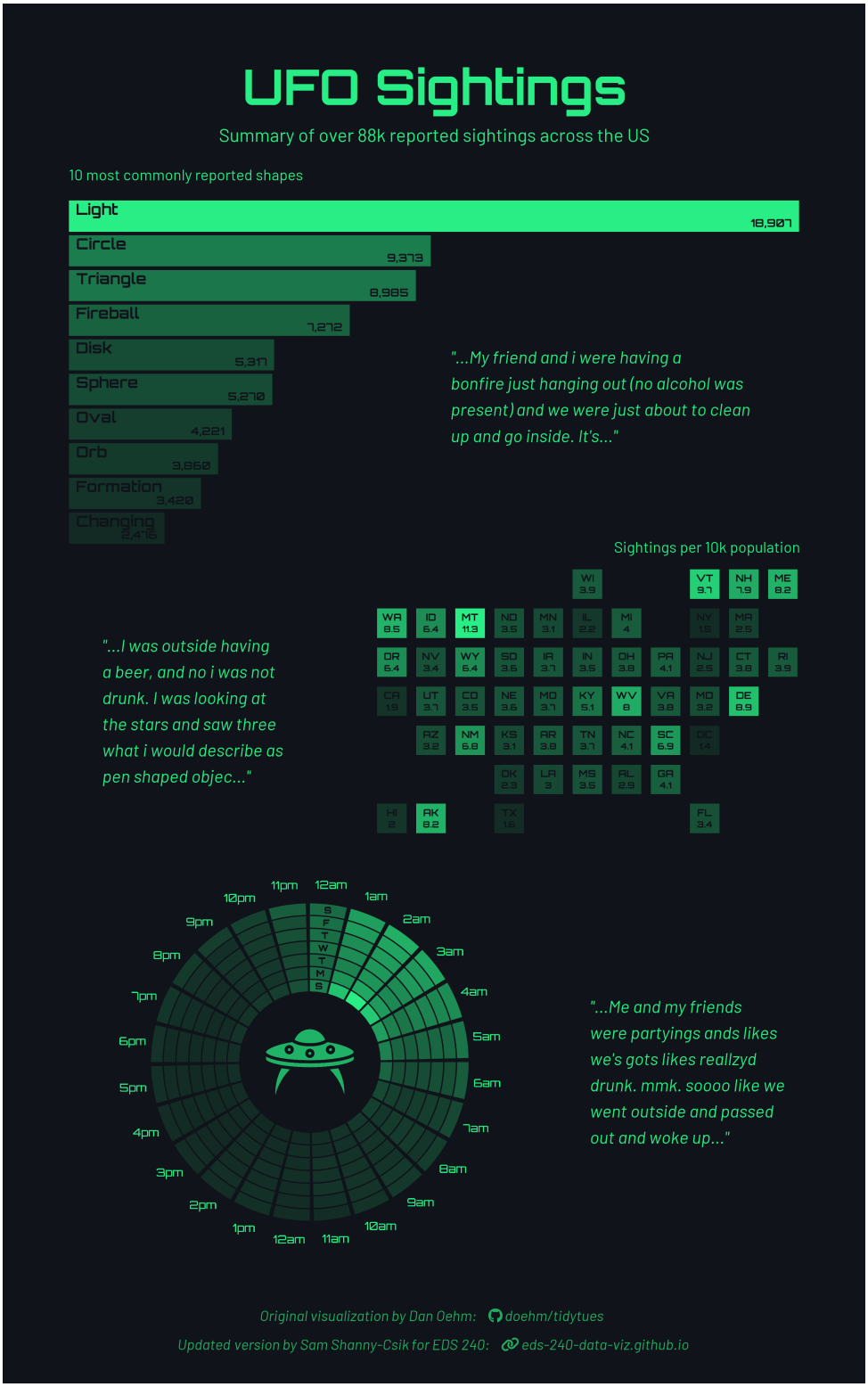
This is basically the layout step where everything finally gets assembled into one infographic.

```
#plot_final is not creating new data it is putting together multiple already built ggplots in

#plot base is the blank poster background.
#inset_element() call is like placing a sticker on that poster at a specific location (coming
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)

plot_final
```


UFO Sightings
Summary of over 88k reported sightings across the US
10 most commonly reported shapes
Light 18,907
Circle 9,373
Triangle 8,985
Fireball 7,272
Disk 5,317
Sphere 5,270
Oval 4,221
Orb 3,860
Formation 3,420
Changing 2,416
"...My friend and i were having a bonfire just hanging out (no alcohol was present) and we were just about to clean up and go inside. It's..."
Sightings per 10k population
"...I was outside having a beer, and no i was not drunk. I was looking at the stars and saw three what i would describe as pen shaped objec..."
"...Me and my friends were partyings ands likes we's gots likes reallzyd drunk. mmk. soooo like we went outside and passed out and woke up..."
Original visualization by Dan Oehm: doehm/tidytues
Updated version by Sam Shanny-Csik for EDS 240: eds-240-data-viz.github.io

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

    - plot_final is assembled by layering all of the multiple pre-built ggplots onto a single base canvas (plot_base), which has the background, title, subtitle, caption, format, and theme. Then all the multiple ggplots are layered on top. each inset_element adds a ggplot onto a specific region of the base.
    - I think the most challenging part is the manual spatial coordination. There isn't a designated layout or you can't see plots snapping into place like you can do on canvas. Everything is interdependent, the numbers are found by iteration, aspect ratio can also affect place.
    - This seems like a very tedious and time consuming process in the end because you would be manually updating all of the ggplots to find visual balance.

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

    - plot_base looks like it is the foundation which doesn't have data, or charts and is meant to stay stable throughout the design process after you choose a theme.
    - plot_final is all the assembly that has all of the visual components and the layout and formatting.
    - I think this makes the code easier to read and modify without redoing many things.

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?**

        - There are several perceptual tasks the view has to to from the UFO vizs to gain info.
        - In the bar chart of UFO shapes they look at the length of the bars to compare frequencies.
        - In the circular day/hour plot they need to look at the opacity of certain angles to see patterns of sightings.
        - From here we see the prioritized data which are placed higher (bar chart) and the secondary information showing frequency by opacity.

    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.**

- Each form and layout was chosen for specific tasks.
  - Bar chart (plot_shape) was a precise comparison of UFO shape frequencies using bar lengths.
  - Map (plot_us) shows the map of the US utilizing spatial patterns, which shows geographic hotspots by using opacity to highlight frequency.
  - Circular plot (plot_day) shows temporal/ cyclic patterns of sightings across hours and days, which is really cool because it showing peaks and daily/ weekly trends through angles and opacity.

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?**

- There is sometimes a choice of the designers to show less precise encodings. Like the opacity of the map shows general patterns but the reader can't actually denote the sightings per 10k population with opacity, they can't tell the actually number. Just a general trend. And that is a trade off of balancing clarity and aesthetics even though it is lower on the accuracy scale.

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

- The bar chart was nice because the clean horizontal bars makes it easy to compare the UFO shape frequencies with a single glance.
- The circular plot is so clever and engaging and fun to read. The polar layout and opacity shows the temporal patterns in sightings easily.

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

- plot_us shows a US map, but breaks up the familiar outline which made it harder to recognize the county. A traditional map outline would have worked fine.
- plot_shape was a good simple bar chart but only shows text labels. Adding small icons or images of each shape could make it easier for readers to visualize and more engaging.

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.**

- Inset_element() from patchwork
  - let's you place a ggplot inside another and you can build a multi-plot infographic that way. I am unsure how often I would use this because I don't have great visual perspective and benefit from grid layouts more., but I appreciate this approach for how flexible you can be with positioning, texts, icons and plots.

- scale::rescale() was super interesting! You can encode values with opacity and rescale them by converting numeric data into 0-1 range and it can highlight magnitude over exact numbers. This is a really cool visual approach to make patterns intuitive and engaging when precise comparisons aren't needed.

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?**

    - In this assignment I would look through the code and write down a general idea of what I thought the code chunk was doing. For the lines I didn't understand, I would ask chatgpt for clarification and ask questions about what the code was doing or why certain functions were used. At the same time I looked up specific details on CRAN or online when I needed precise syntax about a function's argument.
    - I would say using AI was helpful for connecting the technical details to the overall visualization and for answering a lot of the "why" questions about using specific functions for design choices over others. It helped me understand the code better and relate it to the infographic.