HW #1

Interpreting `{ggplot2}` code

Richard Montes Lemus

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

```
library(colorspace)
library(geofacet)
library(ggtext)
library(glue)
library(grid)
library(magick)
```

```
Linking to ImageMagick 6.9.13.29
Enabled features: cairo, fontconfig, freetype, heic, lcms, pango, raw, rsvg, webp
Disabled features: fftw, ghostscript, x11
```

```
library(patchwork)
library(scales)
library(showtext)
```

```
Loading required package: sysfonts
```

```
Loading required package: showtextdb
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.1.0
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x readr::col_factor() masks scales::col_factor()
x purrr::discard()    masks scales::discard()
x dplyr::filter()     masks stats::filter()
x dplyr::lag()        masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```r
# Read in Data
ufo_sightings <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/mai
```

```
Rows: 96429 Columns: 12
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (7): city, state, country_code, shape, reported_duration, summary, day_...
dbl  (1): duration_seconds
lgl  (1): has_images
dttm (2): reported_date_time, reported_date_time_utc
date (1): posted_date

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
places <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/main/data/
```

```
Rows: 14417 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (6): city, alternate_city_names, state, country, country_code, timezone
dbl (4): latitude, longitude, population, elevation_m

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#Save different colors for future assignments
alien <- c("#101319", "#28ee85")
bg <- alien[1]
accent <- alien[2]

# UFO Image used in final render
ufo_image <- magick::image_read(path = here::here("images", "ufo.png"))

# Download custom fonts
sysfonts::font_add_google(name = "Orbitron", family = "orb")
sysfonts::font_add_google(name = "Barlow", family = "bar")

# Load in custom fonts
sysfonts::font_add(family = "fa-brands", regular = here::here("fonts", "Font Awesome 6 Brands
```

```
15 sysfonts::font_add(family = "fa-solid", regular = here::here("fonts", "Font Awesome 6 Free-So
16
17 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 hex codes that correspond to specific colors as variables. These variables will be used several times throughout the script for assigning a color, fill, or color for text. It is advantageous to define it here so you dont have to type out that hex code several other times throughout the script. The hex code is long, complex and susceptible to typos. Therefore, assigning hex codes to simple variables like "bg" and "accent" simplifes using this color and helps avoids typos.

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() searches through a google font data base and downloads them through sys::fonts. The input "name" corresponds to the font's actual name in the data base and allows it to be retrived correctly, the input "family" corresponds to the name you're giving it and will be referencing it by in future plots.

II. Data wrangling

i. Create `df_pop`

```
# Create clean data frame containing total population of unique cities that experienced a UFO
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 the total population of unique cities that experienced a UFO sighting per US state. Some population estimates, however, dont make sense. For example, I dont understand why California contains a population of over 58 million when at its peak it's only been in the high 30 millions.

ii. Create `df_us`

```
# Create data frame for US states containing UFO sighting information
df_us <- ufo_sightings |>
  filter(country_code == "US") |>
  mutate(state = str_replace(string = state,
                             pattern = "Fl",
                             replacement = "FL")) |>
  count(state) |> # Create column containing number of sightings per state
  left_join(df_pop, by = "state") |> # Join total state populations to new data frame
  rename(num_obs = n) |>
  mutate(
    num_obs_per10k = num_obs / pop * 10000, # Calculate number of sightings per 10,000 people
    opacity_val = num_obs_per10k / max(num_obs_per10k) # Create value between 0 - 1 that will
    )
```

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

   - This data frame contains the number of UFO sightings per US state, the state's total population, and the number of UFO sightings per 10,000 people in each state. It does this in order to allow for comparisons across states. Lastly, it contains an opacity value which will be used to color each state according to relative sighting frequency.

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

   - "opacity_val" is a number between 0 - 1 and its relative to the maximum proportion of UFO sightings per 10,000 people. The maximum proportion of UFO sightings per 10,000 people is set at 1 for a US state with the highest proportion and every other state is relative to it. It is calculated so we can color each US state according to this proportion and visualize how their proportions compare to each other with a color gradient.

iii. Create `df_shape`

```
# Create dataframe for top 10 UFO shapes reported
df_shape <- ufo_sightings |>
  filter(!shape %in% c("unknown", "other")) |> # Remove observations where shape was unknown
  count(shape) |> # Total total number of observations per shape
  rename(total_sightings = n) |>
  arrange(desc(total_sightings)) |>
  slice_head(n = 10) |> # Collect top 10 shapes reported
  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 dataframe contains the top ten UFO shapes reported and the number of times each shape was reported.

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 re-orders shapes in the shape column to ascending order based on the number of observations corresponding to each shape. This numer of observations comes from the total_sightings column. Without this step, bar plots would be ordered based on alphabetical order or ordered in some other non-informative way.

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

   - Rescaling the opacity value is helpful here because we have a large gap between the number of observations for the most common shape and the rest. If we didnt rescale the opacity value, we'd have a strong color for 1 (our maximum), then a big gap where middle colors arent used, and then the rest of shapes observed with a washed out color. We need to rescale to 0.3 - 1 to close that gap and have a more gradual color gradient going from the least observed shape in the top 10 and most observed shape in the top 10.

iv. Create `df_day_hour`

```
df_day_hour <- ufo_sightings |>
  mutate(
    day = wday(reported_date_time), # Create column for number corresponding to weekday
    hour = hour(reported_date_time), # Create column for hour of the day
    wday = wday(reported_date_time, label = TRUE) # Create column with name of the weekday
  ) |>
  count(day, wday, hour) |> # Calculate number of times each unique weekday and hour combinat
  rename(total_daily_obs = n) |>
  mutate(
    opacity_val = total_daily_obs / max(total_daily_obs),
    hour_lab = case_when( # Convert hour of the day from military to standard am/pm time
      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 contains every unique combination of weekday and hour for UFO sightings. In other words, the number of times a UFO sighting occured for every weekday's hour - if theres any record for it. It has hour both in military and standard time, represents the weekday with both a unique number and its actual name, and lastly an opacity value which will be used to visually compare the frequency of UFO sighting times to each other.

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

    - This last line is a catch-all for any values in the hour column that did not meet the previous conditions. It is essentially saying that if this value didnt meet the other conditions, it must be higher than 12. Therefore in order to convert it into standard time - 12 must be subtracted from it. For example, the hour 14 would reach this catch all and would be converted into 2pm.

III. Prepare text elements

```
# Paste ... before ufo sighting summary sentences
quotes <- paste0('"...', str_to_sentence(ufo_sightings$summary[c(47816, 6795, 93833)]), '...'

# Create notes on the bottom of graph giving credit to authors
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() is used to dynamically include some text iteratively across some other text. In this case, it iteratively adds "... and ..." to all of the dataset sentences passed through it. So despite the data set text changing, "... and ..." are pasted in front of it and the new sentence is saved that way. glue() on the other hand, just puts different pieces of text together. It doesnt dynamically change the text like paste0(). They used paste0() for quotes because those same quotes are added across all the sentences used. Therefore, it saves time to just specify it once. glue(), on the other hand, is dealing with unique sentences that dont have shared elements with the rest of the variables using glue(), so there is no need for iteratively pasting anything to them.

IV. Build plots

i. Build `plot_shape`

```
# Create horizontal bar plot for UFO shape observations
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)), # Add text to bar for 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)) + # Remove adding between plots and background
  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`).**

    - This x aesthetic is placing the text on this plot using the x axis. For 'shape', it places the shape text on the bar at the x axis location for 200. For 'total_sightings', it places the the total sightings text number at the x axis location for the total sightings minus 200.

ii. Build `plot_us`

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

```
# Create a grid with state and sightings per 10k population
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 authors ordered it this way because they wanted to center the text and sighting number in each plot before faceting them out. This ensures the text was consistently placed at the center across all grids.

iii. Build `plot_day`

```
# Create donut plot for sightings per day of the week and time
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?**

    - This is accomplished by specifying a start and end for a string within str_sub(). We had the string extraction of 'wday' start at 1 and end at 1, so it only got the first letter in the string.

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.**

    - ylim() and xlim() add space between the graph and the graph background. ylim() is helpful for this visualization because it adds what looks like a hole at the center of our graph. This makes it look less cluttered and allows us to follow time and day of the week more easily as we move through it. xlim() adds space between each tile corresponding to an hour, this helps us distinguish the shading that corresponds to each hour more easily.

iv. Build `quote*s`

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

```
# Create ggplot objects for quotes
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 author converted these text elements into ggplot objects because they can be manipulated individually more easily this way. This allows us to create a fully customized plot with ggplot objects at different placements. This is necessary for us because we want to have the quotes and images placed in very specific places.

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.

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

vi. Build `plot_base`

```
#| eval: false
#| echo: true

# Create base plot that will contain all of the plots previously created
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 renders "plot.caption" this way because caption contains custom text and images. For example it has the github logo on one line. Because of this, it needs to use ggtext and element_markdown() in order to support the use of that.

V. Assemble & save

```
# Put all plots on final base plot
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?**

    - 'plot_final' is assembled by placing all the plots we previously created onto our empty base plot. This placement comes from assigning left, right, top, and bottom arguments to a number corresponding to a location on the base plot. I think the single most challenging part of this is getting those numbers assigned to top, bottom, etc. just right. It probably took a lot of trial and error to fine tune that number and get the plots placed exactly where they needed to be.

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

    - The author probably wanted to make this script more modular. With a static base plot, the author could test out different plot placements without needed to rewrite the entire code every time something wasnt placed correctly.

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 must perform position along common scales, length, and shading tasks to extract information from these 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.**

       - For the shapes plot, the author wanted the viewer to be able to see that light shapes were the most commonly sighted.
       - For the sightings per 10k population plot, the author wanted viewers to be able to see which states proportionally have the highest UFO sightings.

- For the time plot, the author wanted the viewer to be able to see when most sightings occur for both the day of the week and time.

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 used shading to represent spatial and temporal patterns related to UFO sightings. Shading is low on the hierarchy because it is difficult for humans to accurately relate numbers to different shades. It is effective here, however, because the point of this graphic is not for the accurate extraction of specific numbers - its to visually represent sighting patterns.

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

- I think the consistent green color of the text and shading is visually-pleasing. Also, this color makes sense, it is an alien-like green so it makes this visualization fit the UFO sighting topic. Also, I think the quotes on the plot are visually pleasing and add another element of data communication to this graphic. It's visually pleasing because it fills in the blank space on this graph. Also, it allows the graphic to communicate qualitative data as opposed to just quantitative.

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

- I think the shading for the bar graphs at the top should be re-adjusted. The last bar is very dark so it's difficult to read the shape name and sighting number on it.
- I also think the bar plots could be a bubble plot with the same shading instead. I dont think it would make it easier for the viewer to identify which shape is the most common since it's hard for humans to relate numbers to area, but I do think it would look pretty cool and fit the alien theme.

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 that I can make a customized plot containing graphs and text in specific locations by turning those graphs and text into ggplot objects that can be manipulated.
- I also learned I could create captions with custom text and figures by using ggtext.

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 used AI tools to interpret specific lines of code that I could not understand after consulting with documentation and thinking about it critically. My approach was to first run seperate chunks of code before and after confusing lines of code. This helped me hone in on what the code was changing. Then, I consulted with documentation by looking up specific lines of code or functions. If I continued to struggled after this, I pasted it into an AI tool and asked it to explain the code to me and give me examples of its use. Consulting with documentation first was really helpful because it gave me information that was helpful and broad enough to forced me to critically think about its behavior. This helped prime my brain with a potential answer before consulting with AI. Thanks to this, I could do a sanity check on AI's answer.