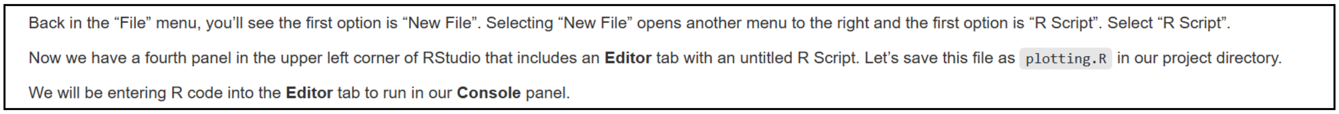
R Lesson Feedback!

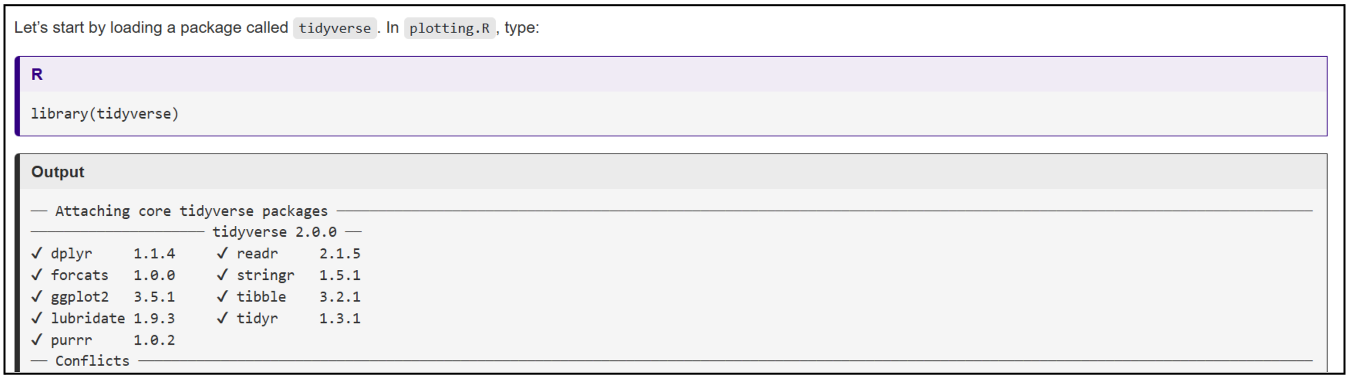


Back in the "File" menu, you'll see the first option is "New File". Selecting "New File" opens another menu to the right and the first option is "R Script". Select "R Script".

Now we have a fourth panel in the upper left corner of RStudio that includes an **Editor** tab with an untitled R Script. Let's save this file as `plotting.R` in our project directory.

We will be entering R code into the **Editor** tab to run in our **Console** panel.

Would it be clearer to word this as go to "File" -> "Save As" -> and name the file "plotting.R"? I had to look this up.



Let's start by loading a package called `tidyverse`. In `plotting.R`, type:

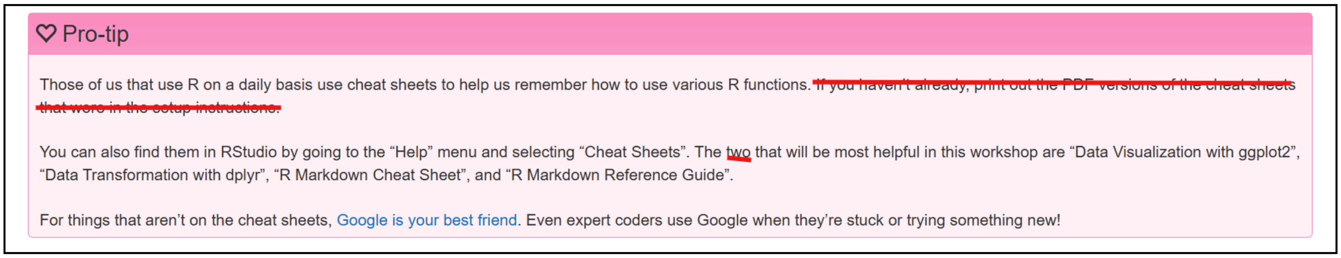
R

```
library(tidyverse)
```

Output

```
── Attaching core tidyverse packages ──────────────────────────────────────────────────────────────────
──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.1     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
✓ purrr     1.0.2
── Conflicts ──────────────────────────────────────────────────────────────────────────────────────────
```

Missing a step description. "In plotting.R, type: … [Hit "Run".]



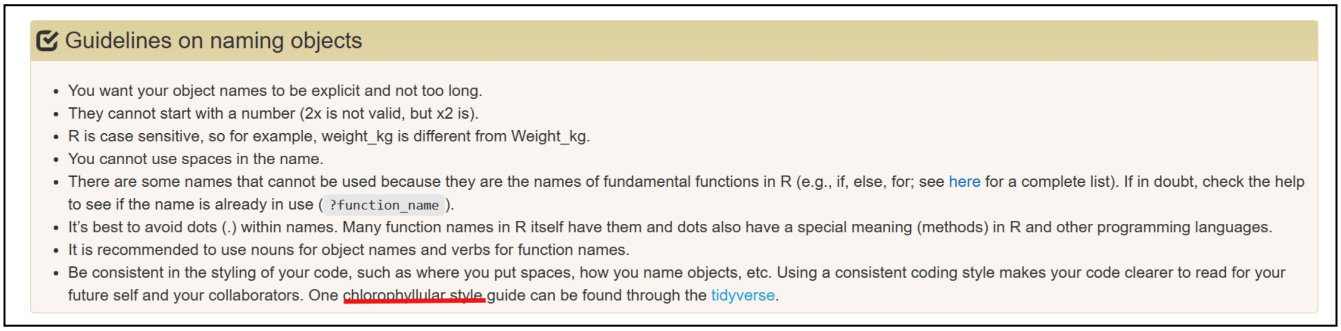
♡ Pro-tip

Those of us that use R on a daily basis use cheat sheets to help us remember how to use various R functions. ~~If you haven't already, print out the PDF versions of the cheat sheets that were in the setup instructions.~~

You can also find them in RStudio by going to the "Help" menu and selecting "Cheat Sheets". The two that will be most helpful in this workshop are "Data Visualization with ggplot2", "Data Transformation with dplyr", "R Markdown Cheat Sheet", and "R Markdown Reference Guide".

For things that aren't on the cheat sheets, Google is your best friend. Even expert coders use Google when they're stuck or trying something new!

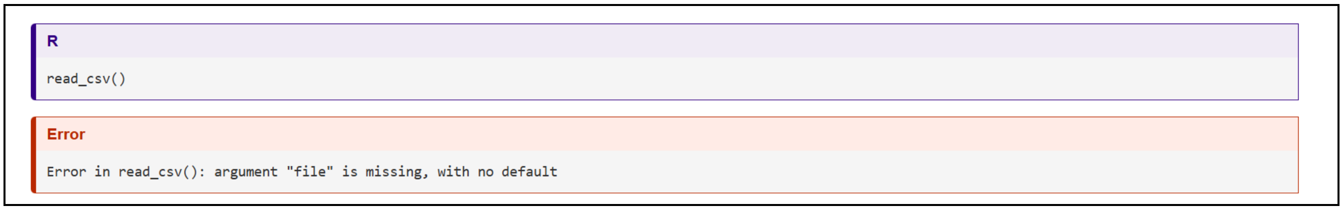
1. We talked about the part that is crossed out. :)
2. There are four things here, not two.



☑ Guidelines on naming objects

- You want your object names to be explicit and not too long.
- They cannot start with a number (2x is not valid, but x2 is).
- R is case sensitive, so for example, weight_kg is different from Weight_kg.
- You cannot use spaces in the name.
- There are some names that cannot be used because they are the names of fundamental functions in R (e.g., if, else, for; see here for a complete list). If in doubt, check the help to see if the name is already in use (`?function_name`).
- It's best to avoid dots (.) within names. Many function names in R itself have them and dots also have a special meaning (methods) in R and other programming languages.
- It is recommended to use nouns for object names and verbs for function names.
- Be consistent in the styling of your code, such as where you put spaces, how you name objects, etc. Using a consistent coding style makes your code clearer to read for your future self and your collaborators. One chlorophyllular style guide can be found through the tidyverse.

"Clorophyllular" -> "popular" :)



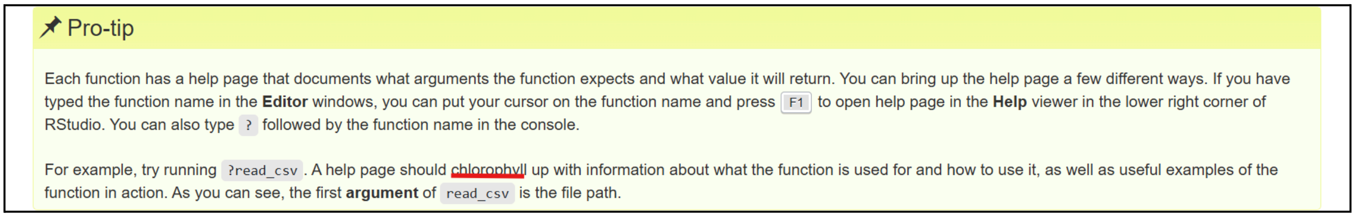
R

```
read_csv()
```

Error

```
Error in read_csv(): argument "file" is missing, with no default
```

This is the error message I get when I write read.csv(). When I type “read_csv()” I get “Error in read_csv() : could not find function ‘read_csv’”. You go on to use “read_csv()” a lot. Should it be “read.csv()”? I don’t know how other computers work, but mine at least needed to use a period instead of an underscore.

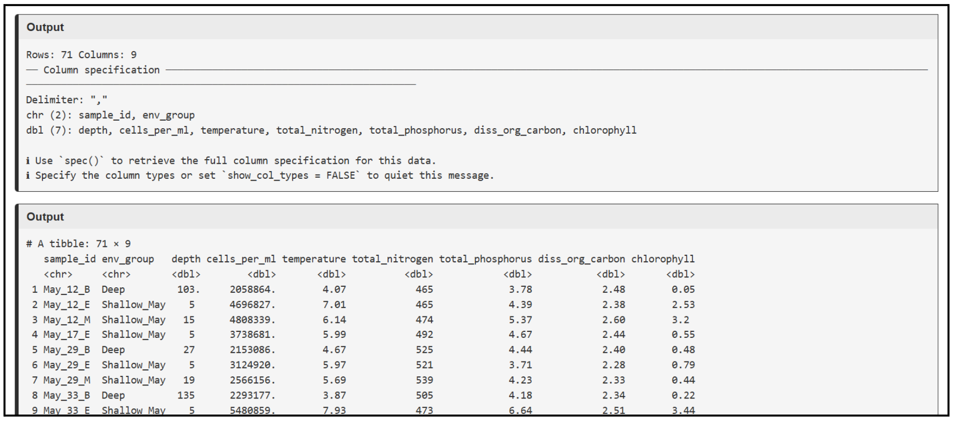


📌 Pro-tip

Each function has a help page that documents what arguments the function expects and what value it will return. You can bring up the help page a few different ways. If you have typed the function name in the **Editor** windows, you can put your cursor on the function name and press `F1` to open help page in the **Help** viewer in the lower right corner of RStudio. You can also type `?` followed by the function name in the console.

For example, try running `?read_csv`. A help page should chlorophyll up with information about what the function is used for and how to use it, as well as useful examples of the function in action. As you can see, the first **argument** of `read_csv` is the file path.

Oh Gus and your silly little chlorophylls! :) Also, when I hit F1 it just mutes my computer?



Output

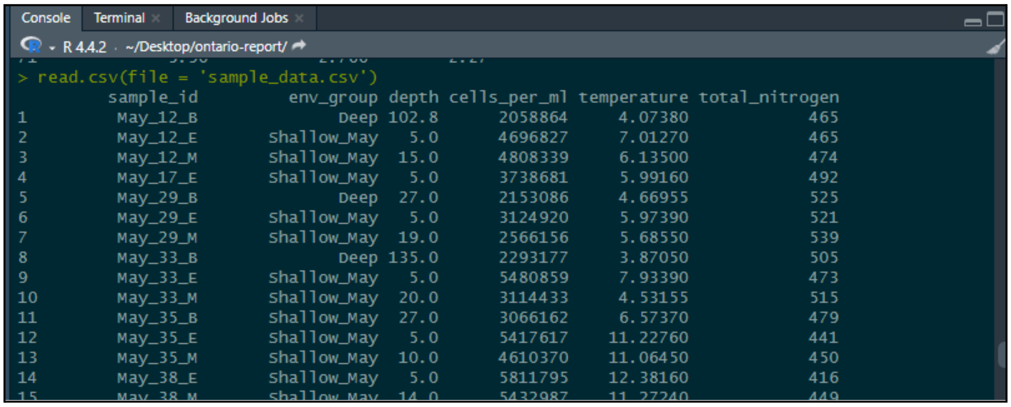
```
Rows: 71 Columns: 9
── Column specification ────────────────────────────────────────────────────────
────────────────────────────────────────────
Delimiter: ","
chr (2): sample_id, env_group
dbl (7): depth, cells_per_ml, temperature, total_nitrogen, total_phosphorus, diss_org_carbon, chlorophyll

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.
```

Output

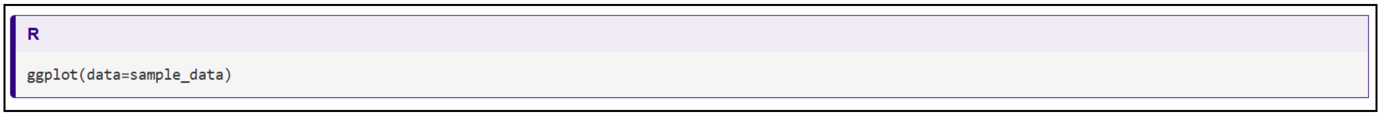
```
# A tibble: 71 × 9
   sample_id env_group   depth cells_per_ml temperature total_nitrogen total_phosphorus diss_org_carbon chlorophyll
   <chr>     <chr>       <dbl>        <dbl>       <dbl>          <dbl>            <dbl>           <dbl>       <dbl>
 1 May_12_B  Deep         103.     2058864.        4.07            465             3.78            2.48        0.05
 2 May_12_E  Shallow_May    5      4696827.        7.01            465             4.39            2.38        2.53
 3 May_12_M  Shallow_May   15      4808339.        6.14            474             5.37            2.60        3.2
 4 May_17_E  Shallow_May    5      3738681.        5.99            492             4.67            2.44        0.55
 5 May_29_B  Deep          27      2153086.        4.67            525             4.44            2.40        0.48
 6 May_29_E  Shallow_May    5      3124920.        5.97            521             3.71            2.28        0.79
 7 May_29_M  Shallow_May   19      2566156.        5.69            539             4.23            2.33        0.44
 8 May_33_B  Deep         135      2293177.        3.87            505             4.18            2.34        0.22
 9 May_33_E  Shallow_May    5      5480859.        7.93            473             6.64            2.51        3.44
```

Didn’t get the first part of this output. Here’s what I got:



```
> read.csv(file = 'sample_data.csv')
     sample_id      env_group depth cells_per_ml temperature total_nitrogen
1     May_12_B           Deep 102.8      2058864     4.07380            465
2     May_12_E    Shallow_May   5.0      4696827     7.01270            465
3     May_12_M    Shallow_May  15.0      4808339     6.13500            474
4     May_17_E    Shallow_May   5.0      3738681     5.99160            492
5     May_29_B           Deep  27.0      2153086     4.66955            525
6     May_29_E    Shallow_May   5.0      3124920     5.97390            521
7     May_29_M    Shallow_May  19.0      2566156     5.68550            539
8     May_33_B           Deep 135.0      2293177     3.87050            505
9     May_33_E    Shallow_May   5.0      5480859     7.93390            473
10    May_33_M    Shallow_May  20.0      3114433     4.53155            515
11    May_35_B    Shallow_May  27.0      3066162     6.57370            479
12    May_35_E    Shallow_May   5.0      5417617    11.22760            441
13    May_35_M    Shallow_May  10.0      4610370    11.06450            450
14    May_38_E    Shallow_May   5.0      5811795    12.38160            416
15    May_38_M    Shallow_May  14.0      5432987    11.27240            449
```

“Creating Our First Plot”

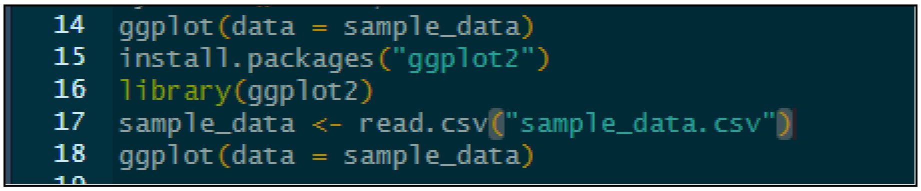


R

```
ggplot(data=sample_data)
```

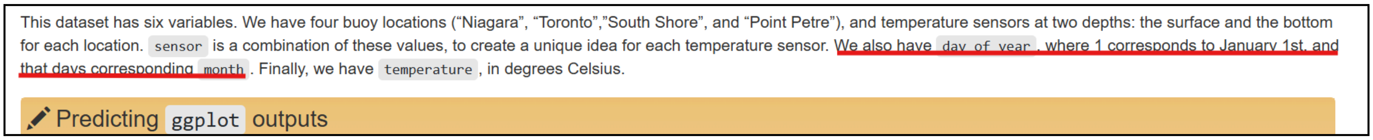
Here, I had to install ggplot2? Thought it happened earlier but maybe we hadn’t opened this project yet so it didn’t work? Might be on me. Note 12/12/24: Disregard. I now understand that packages need to be loaded in for each session.

Here’s what I ran:



```
ggplot(data = sample_data)
install.packages("ggplot2")
library(ggplot2)
sample_data <- read.csv("sample_data.csv")
ggplot(data = sample_data)
```

(Had to re-do the “sample_data <- read.csv("sample_data.csv")” because it had been written as “read_csv” earlier so it hadn’t worked for me.)

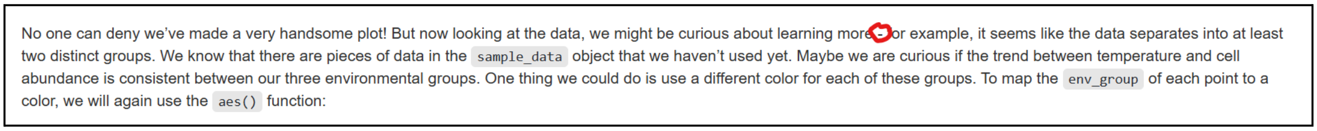


This dataset has six variables. We have four buoy locations (“Niagara”, “Toronto”,”South Shore”, and “Point Petre”), and temperature sensors at two depths: the surface and the bottom for each location. `sensor` is a combination of these values, to create a unique idea for each temperature sensor. We also have `day_of_year`, where 1 corresponds to January 1st, and that days corresponding `month`. Finally, we have `temperature`, in degrees Celsius.

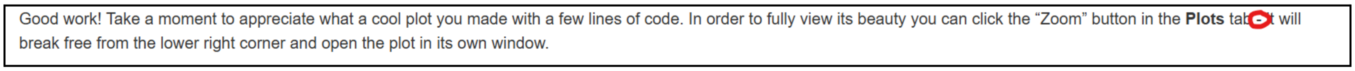
Predicting `ggplot` outputs

Okay. This is a little more nitpicky, but what are you saying here? Maybe, “We also have “day of year”, where 1 corresponds to Janary 1st. The next column lists the “month” in which the sample was collected.”

There are a few places where you use an en dash (-), but I think you mean to use an em dash (-- or —). Or, maybe it’s just the font that makes the dash seem short. You have put a lot of work into creating an accessible coding lesson and this does not hinder readability. You can ignore this comment if you want to.

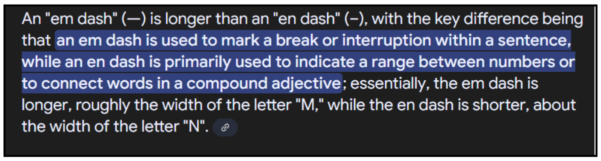


No one can deny we’ve made a very handsome plot! But now looking at the data, we might be curious about learning more - for example, it seems like the data separates into at least two distinct groups. We know that there are pieces of data in the `sample_data` object that we haven’t used yet. Maybe we are curious if the trend between temperature and cell abundance is consistent between our three environmental groups. One thing we could do is use a different color for each of these groups. To map the `env_group` of each point to a color, we will again use the `aes()` function:

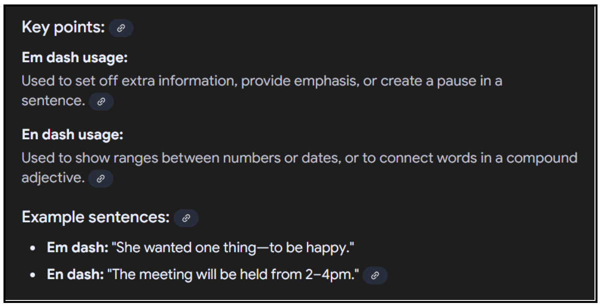


Good work! Take a moment to appreciate what a cool plot you made with a few lines of code. In order to fully view its beauty you can click the “Zoom” button in the **Plots** tab - it will break free from the lower right corner and open the plot in its own window.

**both in “Creating our first plot” section



An "em dash" (—) is longer than an "en dash" (–), with the key difference being that an em dash is used to mark a break or interruption within a sentence, while an en dash is primarily used to indicate a range between numbers or to connect words in a compound adjective; essentially, the em dash is longer, roughly the width of the letter "M," while the en dash is shorter, about the width of the letter "N".



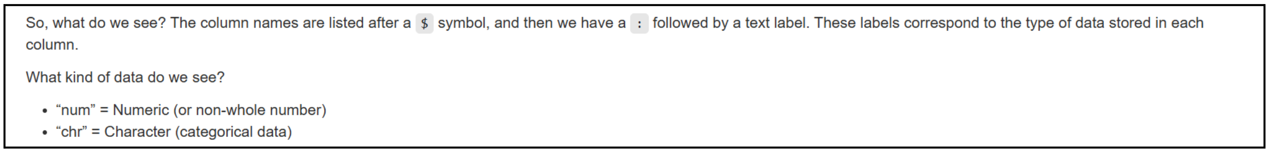
Key points:

Em dash usage:
Used to set off extra information, provide emphasis, or create a pause in a sentence.

En dash usage:
Used to show ranges between numbers or dates, or to connect words in a compound adjective.

Example sentences:

- **Em dash:** "She wanted one thing—to be happy."
- **En dash:** "The meeting will be held from 2–4pm."

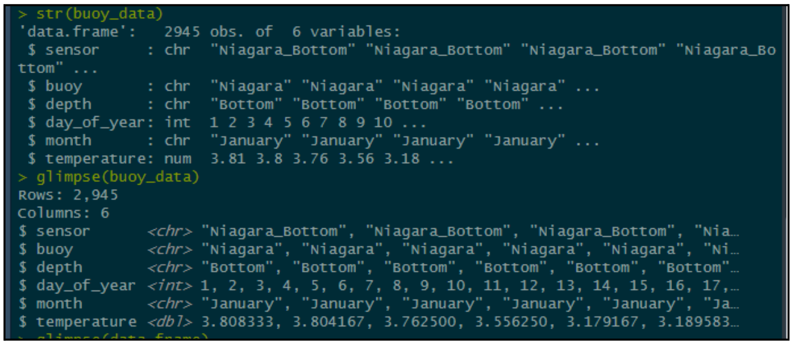


So, what do we see? The column names are listed after a `$` symbol, and then we have a `:` followed by a text label. These labels correspond to the type of data stored in each column.

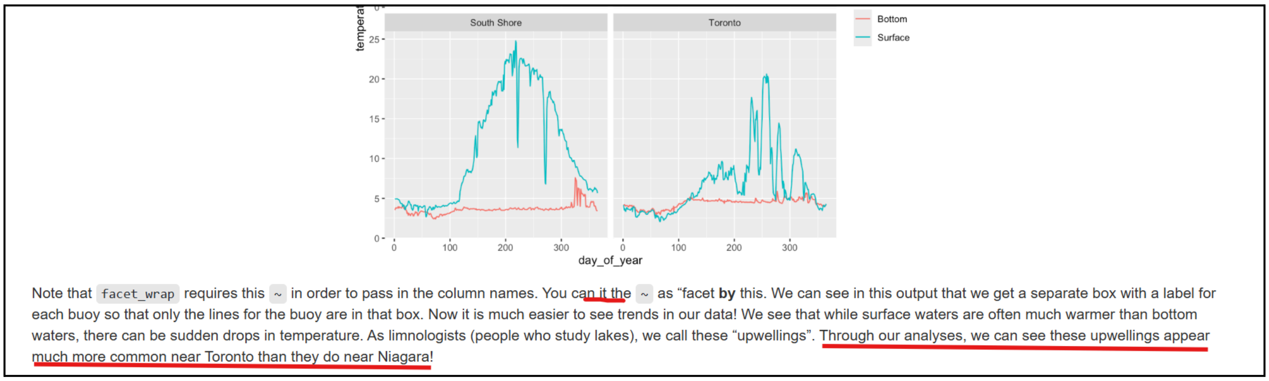
What kind of data do we see?

- “num” = Numeric (or non-whole number)
- “chr” = Character (categorical data)

I also got “int” and “dbl.” I looked it up though and it seems like those are both “num”?

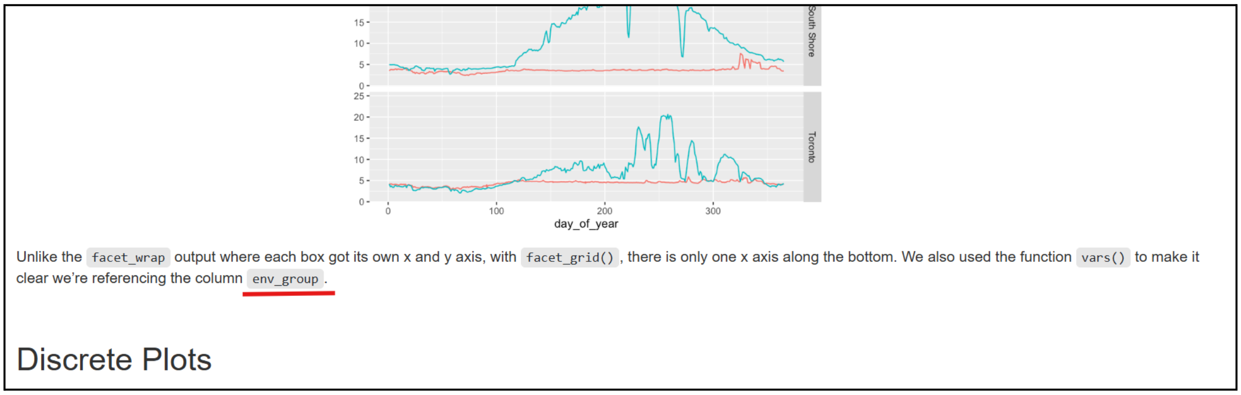


```
> str(buoy_data)
'data.frame':   2945 obs. of  6 variables:
 $ sensor     : chr  "Niagara_Bottom" "Niagara_Bottom" "Niagara_Bottom" "Niagara_Bo
ttom" ...
 $ buoy       : chr  "Niagara" "Niagara" "Niagara" "Niagara" ...
 $ depth      : chr  "Bottom" "Bottom" "Bottom" "Bottom" ...
 $ day_of_year: int  1 2 3 4 5 6 7 8 9 10 ...
 $ month      : chr  "January" "January" "January" "January" ...
 $ temperature: num  3.81 3.8 3.76 3.56 3.18 ...
> glimpse(buoy_data)
Rows: 2,945
Columns: 6
$ sensor      <chr> "Niagara_Bottom", "Niagara_Bottom", "Niagara_Bottom", "Nia…
$ buoy        <chr> "Niagara", "Niagara", "Niagara", "Niagara", "Niagara", "Ni…
$ depth       <chr> "Bottom", "Bottom", "Bottom", "Bottom", "Bottom", "Bottom"…
$ day_of_year <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,…
$ month       <chr> "January", "January", "January", "January", "January", "Ja…
$ temperature <dbl> 3.808333, 3.804167, 3.762500, 3.556250, 3.179167, 3.189583…
> glimpse(data.frame)
```



Note that `facet_wrap` requires this `~` in order to pass in the column names. You can it the `~` as “facet **by** this. We can see in this output that we get a separate box with a label for each buoy so that only the lines for the buoy are in that box. Now it is much easier to see trends in our data! We see that while surface waters are often much warmer than bottom waters, there can be sudden drops in temperature. As limnologists (people who study lakes), we call these “upwellings”. Through our analyses, we can see these upwellings appear much more common near Toronto than they do near Niagara!

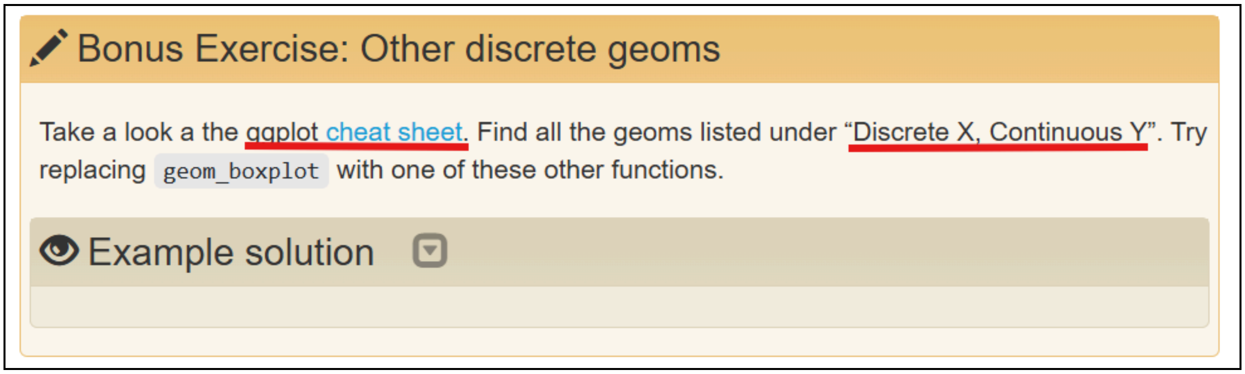
1. “You can interpret the ~ as…”?
2. “Through our analyses, we can see these upwellings appear more frequently near Toronto than they do near Niagra”?
   or “Through our analyses, we can see these upwellings appear to be much more common near Toronto than they are near Niagra”?



Unlike the `facet_wrap` output where each box got its own x and y axis, with `facet_grid()`, there is only one x axis along the bottom. We also used the function `vars()` to make it clear we’re referencing the column `env_group`.

Discrete Plots

Isn’t the column we’re referencing called “buoy”?

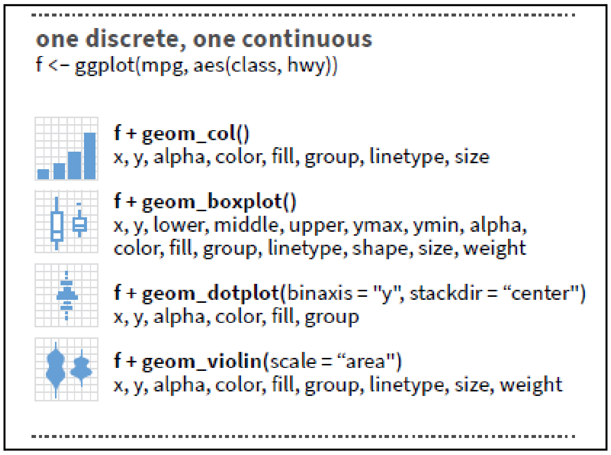


Bonus Exercise: Other discrete geoms

Take a look a the ggplot cheat sheet. Find all the geoms listed under “Discrete X, Continuous Y”. Try replacing `geom_boxplot` with one of these other functions.

Example solution

1. The ggplot cheat sheet is called “data visualization,” if people downloaded it as a PDF per earlier instructions and didn’t rename it.
2. It’s just called “one discrete, one continuous” if I’m looking at the right thing.



one discrete, one continuous

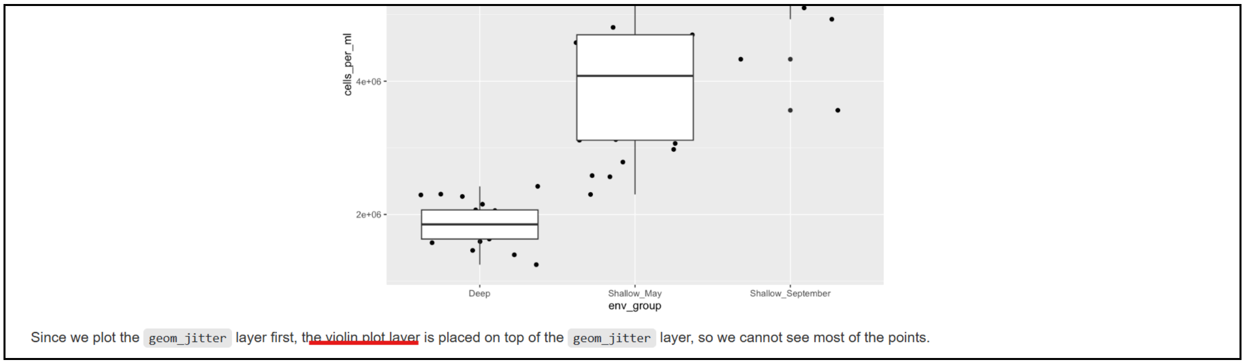
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

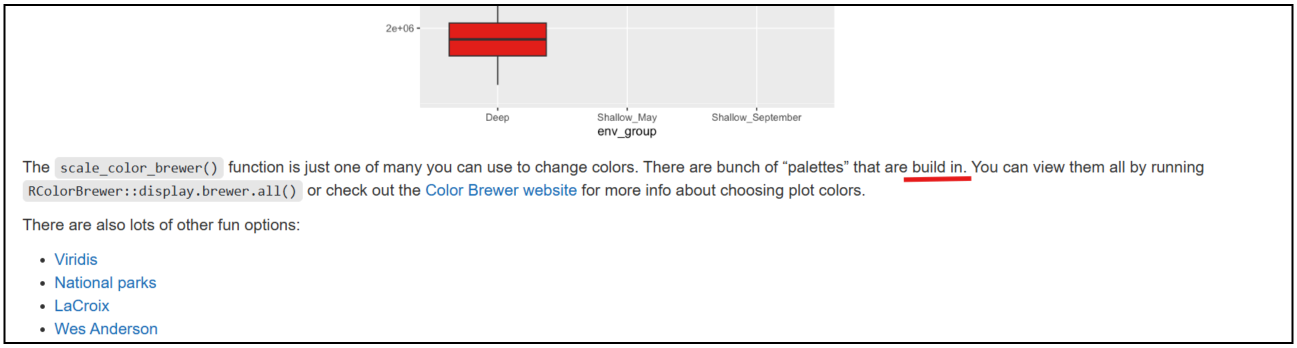
f + geom_dotplot(binaxis = "y", stackdir = “center")
x, y, alpha, color, fill, group

f + geom_violin(scale = “area")
x, y, alpha, color, fill, group, linetype, size, weight



Since we plot the `geom_jitter` layer first, the violin plot layer is placed on top of the `geom_jitter` layer, so we cannot see most of the points.

This is a boxplot, not a violin plot



The `scale_color_brewer()` function is just one of many you can use to change colors. There are bunch of "palettes" that are build in. You can view them all by running `RColorBrewer::display.brewer.all()` or check out the Color Brewer website for more info about choosing plot colors.

There are also lots of other fun options:

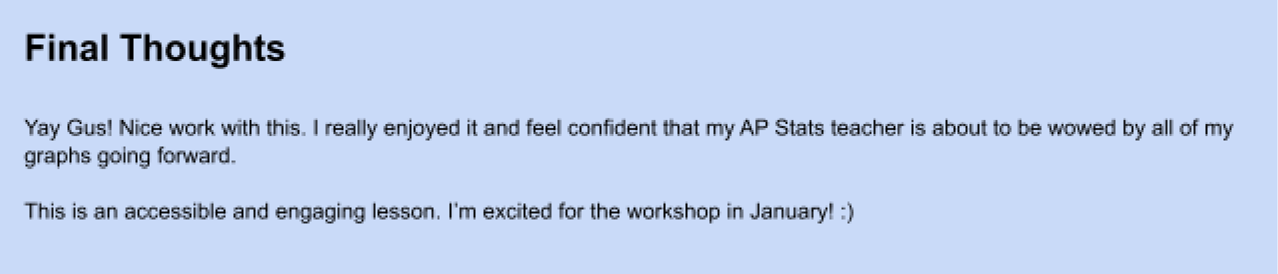
- Viridis
- National parks
- LaCroix
- Wes Anderson

"Built-in"

!!!

I got a little bit lost in the section about downloading packages for color palettes that are more fun. There's a lot in the lesson on selecting color palettes, and by the end I was feeling a little oversaturated and confused. One area of confusion: how do I see the options of palettes in the Wes Anderson package?

That said, I like that at this point in the lesson we go back to downloading packages so that people can get comfortable with that in a self-guided way. Also, all of the color options are definitely fun, and I think the grad students might really enjoy getting into the aesthetics.



Final Thoughts

Yay Gus! Nice work with this. I really enjoyed it and feel confident that my AP Stats teacher is about to be wowed by all of my graphs going forward.

This is an accessible and engaging lesson. I'm excited for the workshop in January! :)