ComputationalThinking2

Overview

Welcome! This is the second "computational thinking" lesson, which continues work on for loops and revisits conditional statements.

You will submit one output for this activity:

1. A **PDF** of a rendered Quarto document with all of your R code. Please create a new Quarto document (e.g. don't use this `README.qmd`), include all of the code that appears in this document, in addition to adding your own code and **answers to all of the questions** in the "Q#" sections. Submit this through Gradescope.

- For this activity, we want you to get more GitHub practice: copy this repository to your own GitHub as in the second activity (if you haven't already been doing this). At the end, you will practice pushing your changes back to the repository.

If you have trouble submitting as a PDF, please ask Calvin or Malin for help. If we still can't solve it, you can submit the .qmd file instead.

A reminder: **Please label the code** in your final submission in two ways: 1) denote your answers to each question using headers that correspond to the question you're answering and 2) thoroughly "comment" your code: remember, this means annotating your code directly by typing descriptions of what each line does after a `#`. This will help future you!

Let's start by reading in the relevant packages

```r
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   3.5.1     v tibble    3.3.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
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 becom
```

```r
library(here)
```

```
here() starts at /Users/waltvance/Library/Mobile Documents/com~apple~CloudDocs/School/UCSC/Bl
```

---

1. Conditionals

We've already worked with conditionals during one of the seaside chats, which used `case_when()`, `dplyr's` version of an `if_else` statement. Today we'll walk through using `if_else` statements, the precursor to `case_when()`, as well as get more practice with `case_when()`.

Remember the conditional statements from when we used `filter()`:

- Greater than: `>`
- Greater than or equal to: `>=`
- Less than: `<`
- Less than or equal to: `<=`
- Exactly equal to: `==`
- Not equal to: `!=`

1.1 if else statements

`if else` statements take the form of:

```
if (condition){
code to execute if condition is TRUE
} else {
code to execute if condition is FALSE
}
```

For example, below we define x as 5, then write a statement that checks whether or not a statement, `x > 10`, is TRUE. If it's true, it will print out the first part. If not, it will print the second.

```r
x <- 5

# Check if the value of x is greater than 10
if(x > 10)
{
  # Paste takes the value stored in x and combines that with a character string
  print(paste(x, "is greater than 10"))
} else {
  print(paste(x, "is less than or equal to 10"))
}
```

```
[1] "5 is less than or equal to 10"
```

Q1.1: Modify the value of x

Copy/paste this example but change x to be 11. What is the output of the if else statement?

```r
#changed x to equal 11
x <- 11

# Check if the value of x is greater than 10
if(x > 10)
{
  # Paste takes the value stored in x and combines that with a character string
  print(paste(x, "is greater than 10"))
} else {
  print(paste(x, "is less than or equal to 10"))
}
```

```
[1] "11 is greater than 10"
```

Using `traceback()` for errors

There are many ways to get help with errors in R. The `traceback()` function is one, and it prints out a summary of how your code arrived at the error you received.

Consider this if else statement:

```r
x <- 5

# Check if the value of x is greater than 10
if(x > 10)
{
  # If x is > 10, multiple x by 2
  print(x*2)
} else
{
  # If x is not > 10, divide x by 2
  print(x/2)
}
```

```
[1] 2.5
```

Let's change the `x <- 5` to `x <- "five"`. Run this code, which attempts to do math on a character string, and immediately run `traceback()` afterwards.

Remember, this will give you an error if you try to render the document, so substitute the `{r}` for `{r eval = FALSE}` when in the source editor.

```r
x <- "five"

# Check if the value of x is greater than 10
if(x > 10)
{
  # If x is > 10, multiple x by 2
  print(x*2)
} else
{
  # If x is not > 10, divide x by 2
  print(x/2)
}
```

In this example, `traceback()` prints `1: print(x * 2) at #4`. `1:` tells you that this is the first thing it tried to do. The `print(x*2)` tells you what it tried to do that didn't work, and `at #4` means the error is at line #4. This can be particularly helpful if you are running a big function, loop, series of functions, etc, and need to figure out exactly when and where that pesky error started.

You can read more about debugging and the `traceback()` function in particular at this link: https://adv-r.hadley.nz/debugging.html (`traceback()` in section 22.3)

Adding another condition

Let's make it a little more complex by adding a third conditional. The if else statements in R can be nested together to form multiple statements and will evaluate expressions based on the conditions one by one, beginning from the outer condition to the inner one.

Here, we will make R print one thing if x is less than 10, something else if x is equal to 10, and a third thing if x is greater than 10.

```r
# define a variable
x <- 11

# check the value of x using nested if-else statements
if (x < 10) {
  # if x is less than 10
  print("x is less than 10")
} else {
  # if x is exactly equal to 10
  if (x == 10) {
    print("x is 10!!!")
  } else {
    # if x is greater than 10
    print("x is greater than 10")
  }
}
```

```
[1] "x is greater than 10"
```

Run the statement above with `x = 9`, `x = 10`, and `x = 11` to see what happens. No need to re-paste the code; just change the x value and rerun the if else statement below it.

Let's combine what we learned about for loops with an if else statement. The goal here is to take a vector of numbers, loop through it, and apply our if else statement to each element in the vector, telling us if it's less than, equal to, or greater than 10.

First define a vector to loop through:

```r
vec <- c(9, 10, 11, 12)
```

Then we start the for loop:

```r
# For 1 through the length of the vector "vec"
for (i in 1:length(vec)) {

  # check the value of using nested if-else statements
  if (vec[i] < 10) {
    # if the element is less than 10
    print("value is less than 10")
  } else {
    # if the element is exactly equal to 10
    if (vec[i] == 10) {
      # if the element equals 10
      print("value is 10!!!")
    } else {
      # if the element is greater than 10
      print("value is greater than 10")
    }
  }

}
```

```
[1] "value is less than 10"
[1] "value is 10!!!"
[1] "value is greater than 10"
[1] "value is greater than 10"
```

The for loop iterates through the vector and applies the if else statement separately for each vector element, printing an output for each number.

Q1.2: Create a new for loop + if else statement

Remake a similar for loop / if else statement as above. The for loop should loop through a vector called `y` that has the values `-2, 42, 0, 10` and should assess whether or not the values are negative, positive, or equal to zero.

First define a vector to loop through, then run the for loop / if else statement.

```r
vec <- c(-2, 42, 0, 10)
```

```r
# For 1 through the length of the vector "vec"
for (i in 1:length(vec)) {

  # check the value of using nested if-else statements
  if (vec[i] < 0) {
    # if the element is less than 0
    print("value is less than 0")
  } else {
    # if the element is exactly equal to 0
    if (vec[i] == 0) {
      # if the element equals 10
      print("value is 0!!!")
    } else {
      # if the element is greater than 0
      print("value is greater than 0")
    }
  }

}
```

```
[1] "value is less than 0"
[1] "value is greater than 0"
[1] "value is 0!!!"
[1] "value is greater than 0"
```

1.2 `case_when()` and pikas

Let's return to the `tidyverse` and revisit `case_when()`, `dplyr`'s version of the if else statement. We will work with the `lterdatasampler` data again, so let's load in those two packages:

```r
library(lterdatasampler)
library(tidyverse)
```

Figure 1: LTER researcher releasing a pika at the Niwot site, LTER CC BY-SA 4.0

Today we will work with the pika data, `nwt_pikas`! From the vignette on the `lterdatasampler` package:

"*Researchers at the Niwot Ridge Long Term Ecological Research Site (NWT LTER) seek to study and monitor the health of the Colorado Rockies over time. Because of external factors like climate change, it's more important than ever for scientists to understand how and why the Rockies are changing.*"

"*The American pika (*Ochotona princeps*) is a key species present at the NWT LTER. Despite their small size, pikas can be very informative about the health of the ecosystem. If pikas are more stressed, it can suggest that their habitat has declined in quality. As a result, the study of pikas is critical to the Colorado Rockies ecosystem.*"

Also, these study sites are very close to where Calvin grew up :)

Q1.3: How do the researchers measure pika stress?

Navigate to the help page of this dataset and read up on the dataset. In a couple sentences, describe how the researchers measured pika stress, including how they did it, what the "stress" variable is called, and what the units are.

A1.3 data collected between jun-sept of 2018. stress measured by observing glucocorticoid metabolite present in pika feces. stress variable = concentration_pg_m, units = picogram GCM/gram dry pika feces

---

Q1.4: What does a row represent in this data?

What does a row represent in this dataset? Is it an average value from a site? From a station? From an individual pika? From an individual pika poop? From a subsample of a pika poop?

A1.4 each row represents an data from a sample from an individual pika and its poop

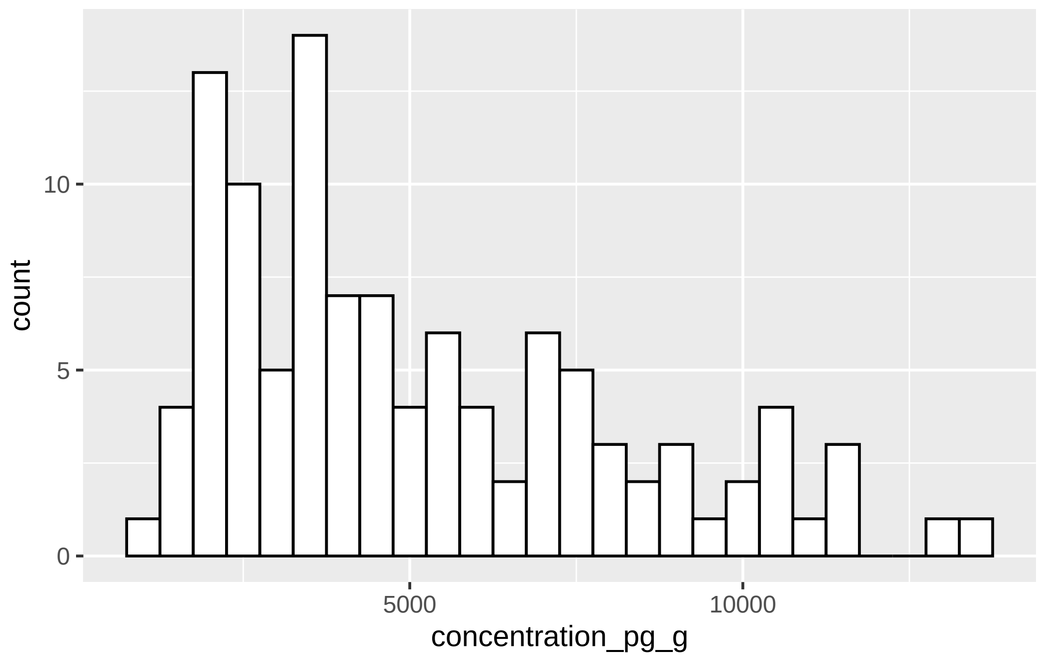
---

Categorize stress

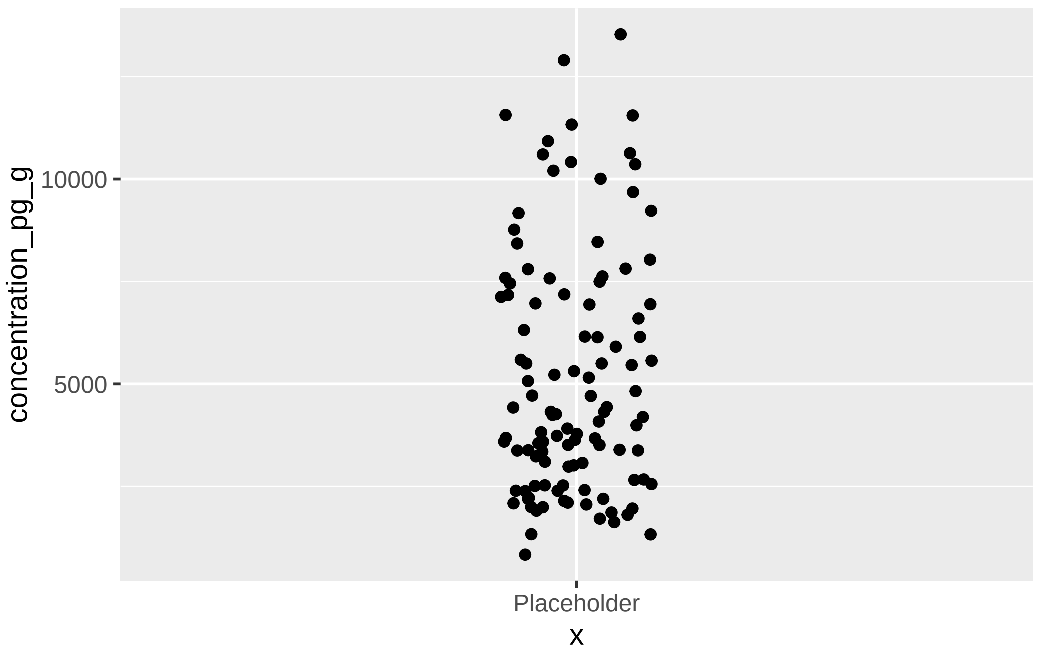
The researchers measured stress on a continuous scale in the `concentration_pg_g` column. To practice using `case_when()`, let's turn this into a categorical variable by assigning an arbitrary threshold after which we call a pika "stressed" (warning: this is not endorsed by Pika Science).

First let's look at a distribution of the values in the `concentration_pg_g` column in two ways to see if there are any natural breakpoints:

```
# Make a histogram
nwt_pikas %>%
  ggplot(aes(x = concentration_pg_g)) +
  # Add the histogram geom, which only needs an x-axis
  # Choose a binwidth of 500 picogram GCM/gram
  geom_histogram(binwidth = 500,
                 fill = "white",
                 color = "black")
```



```
# Make a scatterplot with jittered points
nwt_pikas %>%
  # We're adding a little placeholder axis just so we can see the point distribution
  ggplot(aes(x ="Placeholder",
               y = concentration_pg_g)) +
  # Add the geom_jitter geom
  geom_jitter(width = 0.1)
```



It looks like a lot of the data points are concentrated before 5000 or 6000, after which there's a long tail of data. Let's arbitrarily choose 5000 as the threshold after which pikas become stressed (again, NOT endorsed by Big Pika Science).

Let's create a new column with a `mutate` and a `case_when`. Let's call the new column `stress_category` and have it equal "Stressed!!!!!!" when the `concentration_pg_g` column is greater than 5000, and "Chill" when it's less than or equal to that value.

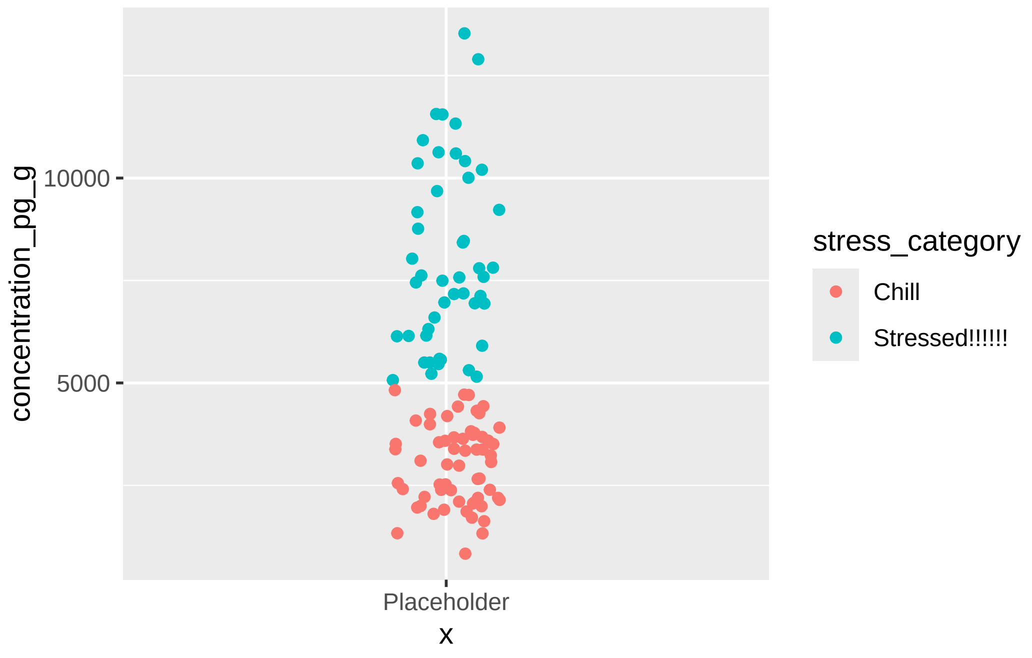
```
nwt_pikas_categ <- nwt_pikas %>%
  # Call the new column stress_category
  mutate(stress_category = case_when(
    # When the value is > 5000, make the new column's value "Stressed!!!!"
    concentration_pg_g > 5000 ~ "Stressed!!!!!!",
    # Otherwise, make the new column's value "Chill"
    .default = "Chill"
  ))

# Check out the first 6 rows, but remove the utm columns just for visibility
head(nwt_pikas_categ %>% select(-c(utm_easting, utm_northing)))
```

```
# A tibble: 6 x 7
  date       site       station   sex   concentration_pg_g elev_m stress_category
  <date>     <fct>      <fct>     <fct>              <dbl>  <dbl> <chr>
1 2018-06-08 Cable Gate Cable G~ male              11563.  3343. Stressed!!!!!!
2 2018-06-08 Cable Gate Cable G~ male              10629.  3353. Stressed!!!!!!
3 2018-06-08 Cable Gate Cable G~ male              10924.  3358. Stressed!!!!!!
4 2018-06-13 West Knoll West Kn~ male              10414.  3578. Stressed!!!!!!
5 2018-06-13 West Knoll West Kn~ male              13531.  3584. Stressed!!!!!!
6 2018-06-13 West Knoll West Kn~ <NA>               7799.  3595. Stressed!!!!!!
```

Q1.5: Remake the scatterplot, but color the points by the new stress category

Remake our scatterplot from above, but this time color the points by the new category we just made.

```
# Make a scatterplot with jittered points
nwt_pikas_categ %>%
  # We're adding a little placeholder axis just so we can see the point distribution
  ggplot(aes(x ="Placeholder",
             y = concentration_pg_g)) +
  # Add the geom_jitter geom and color it by stress_category
  geom_jitter(width = 0.1,aes(color = stress_category))
```



Hopefully you can now visualize what that `case_when()` statement did with the information we gave it.

Let's add in another categorization, this time based on the time of year. First, we'll create a new "month" column based on the date column. The date column is in `date` format, which R recognizes as such, so functions in the `lubridate` package (already loaded within `tidyverse`) can help us easily

```
nwt_pikas_categ2 <- nwt_pikas_categ %>%
  # Create a new column called month
  # then, extract the month from the date using the month() function
  mutate(month = month(date)) %>%
  # Lastly, relocate the month column after the date column so it's more easily visible to us
  relocate(month, .after = date)

head(nwt_pikas_categ2)
```

```
# A tibble: 6 x 10
  date       month site       station      utm_easting utm_northing sex
  <date>     <dbl> <fct>      <fct>              <dbl>        <dbl> <fct>
1 2018-06-08     6 Cable Gate Cable Gate 1      451373      4432963 male
2 2018-06-08     6 Cable Gate Cable Gate 2      451411      4432985 male
3 2018-06-08     6 Cable Gate Cable Gate 3      451462      4432991 male
4 2018-06-13     6 West Knoll West Knoll 3      449317      4434093 male
5 2018-06-13     6 West Knoll West Knoll 4      449342      4434141 male
6 2018-06-13     6 West Knoll West Knoll 5      449323      4434273 <NA>
# i 3 more variables: concentration_pg_g <dbl>, elev_m <dbl>,
#   stress_category <chr>
```

Now, let's change the categories so that it also includes some info about the time of the summer that these stress values were sampled. If the month is June or July (6 or 7) AND the GCM concentration is > 5000, let's call that "Early summer stress", vs "Early summer chill" if it's less than or equal to 5000. Let's also repeat that with the late summer, August and September (8 or 9).å

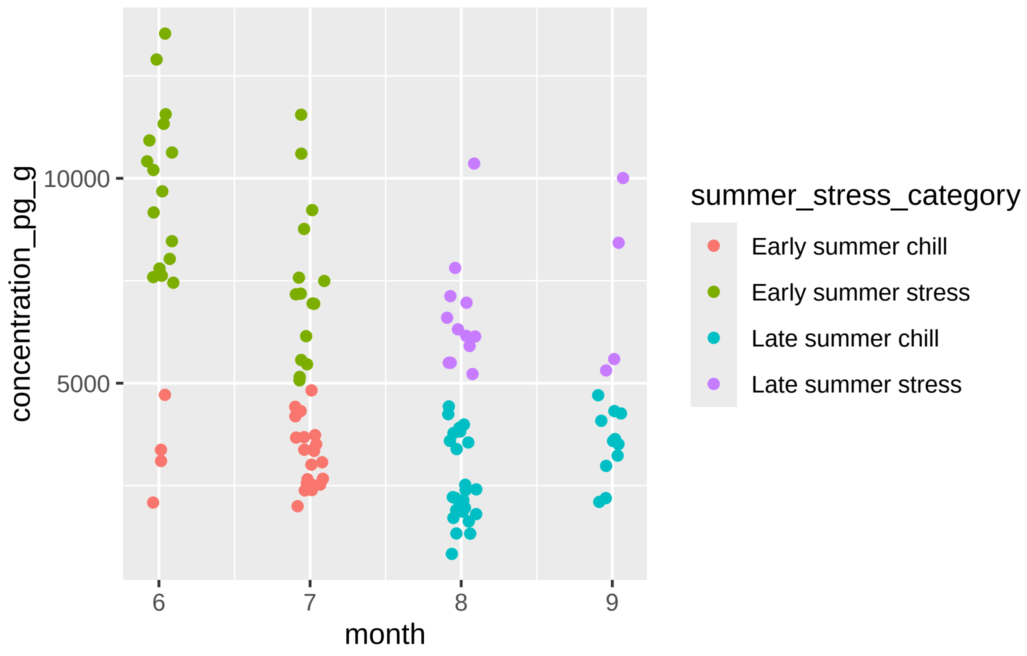
```
nwt_pikas_summerstress <- nwt_pikas_categ2 %>%
  mutate(summer_stress_category = case_when(
    (month == 6 | month == 7) & concentration_pg_g > 5000 ~ "Early summer stress",
    (month == 6 | month == 7) & concentration_pg_g <= 5000 ~ "Early summer chill",
    (month == 8 | month == 9) & concentration_pg_g > 5000 ~ "Late summer stress",
    (month == 8 | month == 9) & concentration_pg_g <= 5000 ~ "Late summer chill",
    .default = "NA"
  ))

head(nwt_pikas_summerstress)
```

```
# A tibble: 6 x 11
  date       month site       station      utm_easting utm_northing sex
  <date>     <dbl> <fct>      <fct>              <dbl>        <dbl> <fct>
1 2018-06-08     6 Cable Gate Cable Gate 1      451373      4432963 male
2 2018-06-08     6 Cable Gate Cable Gate 2      451411      4432985 male
3 2018-06-08     6 Cable Gate Cable Gate 3      451462      4432991 male
4 2018-06-13     6 West Knoll West Knoll 3      449317      4434093 male
5 2018-06-13     6 West Knoll West Knoll 4      449342      4434141 male
6 2018-06-13     6 West Knoll West Knoll 5      449323      4434273 <NA>
# i 4 more variables: concentration_pg_g <dbl>, elev_m <dbl>,
#   stress_category <chr>, summer_stress_category <chr>
```

Note the use of parentheses around the "or" statements, e.g. `(month == 6 | month == 7)`. This tells `case_when()` to first evaluate whether the month column is 6 or 7, and THEN evaluate whether the GCM concentration is > 5000. If I left out the parentheses (which I mistakenly did when creating this assignment), `case_when()` would look at whether then month is 6 OR whether the GCM concentration is > 5000 with the month == 7. This would mean that regardless of GMC concentration, any row with month == 6 would be assigned "Early summer stress", regardless of GMC concentration. If you want to visualize this, remove the parentheses and run the graph below, then compare that graph with that happens when you include the parentheses.

Let's visualize this with a scatterplot again to see what happened, this time with month on the x axis

```
# Make a scatterplot with jittered points
nwt_pikas_summerstress %>%
  # We're adding a little placeholder axis just so we can see the point distribution
  ggplot(aes(x = month,
             y = concentration_pg_g,
             color = summer_stress_category)) +
  # Add the geom_jitter geom
  geom_jitter(width = 0.1)
```



---

2. DIY a for loop and an if else statement / case when

Now it's your turn: Choose a dataset that we've used in this class before and perform two actions on it:

1. Write a for loop to iterate through the dataset in some way
2. Apply an if else OR a case_when() statement to the dataset in some way

This is purposefully vague: it is your choice about what to do. Revisit both computational thinking activities to get some ideas. Neither has to be complicated, but make sure that you explain what you are doing and annotate your code.

Q2.1 What dataset are you using?

A2.1 Palmerpenguins!

```r
library(palmerpenguins)
# Read in the palmer penguins data
penguins <- penguins
```

For loop

Q2.2 Write a couple sentences describing what you want to do with the for loop

Before executing the for loop, write a couple sentences clearly describing what it is that you are trying to accomplish.

A2.2 I want this loop to tell me the ratio of bill length to bill depth

---

Q2.3 Apply the for loop to this dataset

Apply a for loop in some way to this dataset. Describe what you are doing in comments on each line.

```r
#creates a vector that is based on a for loop that iritates the amount of times there are ro
bill.ratio <- for(i in 1:length(penguins$bill_length_mm)) {
#printing each ratio as it comes along
  print(penguins$bill_length_mm[i]/penguins$bill_depth_mm[i])

}
```

```
[1] 2.090909
[1] 2.270115
[1] 2.238889
[1] NA
[1] 1.901554
[1] 1.907767
[1] 2.185393
[1] 2
[1] 1.883978
[1] 2.079208
```

```
[1] 2.210526
[1] 2.184971
[1] 2.335227
[1] 1.820755
[1] 1.63981
[1] 2.05618
[1] 2.036842
[1] 2.05314
[1] 1.869565
[1] 2.139535
[1] 2.065574
[1] 2.016043
[1] 1.869792
[1] 2.110497
[1] 2.255814
[1] 1.867725
[1] 2.182796
[1] 2.26257
[1] 2.037634
[1] 2.142857
[1] 2.365269
[1] 2.055249
[1] 2.219101
[1] 2.164021
[1] 2.141176
[1] 1.85782
[1] 1.94
[1] 2.281081
[1] 1.948187
[1] 2.08377
[1] 2.027778
[1] 2.217391
[1] 1.945946
[1] 2.238579
[1] 2.189349
[1] 2.106383
[1] 2.163158
[1] 1.984127
[1] 2.011173
[1] 1.995283
[1] 2.237288
[1] 2.121693
[1] 1.955307
```

```
[1] 2.153846
[1] 1.906077
[1] 2.225806
[1] 2.228571
[1] 2.159574
[1] 2.198795
[1] 1.968586
[1] 2.112426
[1] 1.957346
[1] 2.211765
[1] 2.258242
[1] 2.128655
[1] 2.311111
[1] 2.191358
[1] 2.151832
[1] 2.162651
[1] 2.154639
[1] 1.763158
[1] 2.157609
[1] 2.302326
[1] 2.42328
[1] 2.028571
[1] 2.313514
[1] 2.434524
[1] 1.917526
[1] 2.248447
[1] 2.204188
[1] 2.011628
[1] 2.4375
[1] 1.952128
[1] 1.809278
[1] 2.095506
[1] 2.034483
[1] 1.861538
[1] 1.983871
[1] 1.994792
[1] 2.069149
[1] 1.983333
[1] 2.270718
[1] 1.988304
[1] 2.187845
[1] 2.092486
[1] 2.15873
```

```
[1] 2.048387
[1] 2.178378
[1] 2.055901
[1] 2.335135
[1] 1.955307
[1] 2.05
[1] 2.35625
[1] 1.89
[1] 2.037634
[1] 2.100529
[1] 2.244186
[1] 1.91
[1] 2.241176
[1] 2.273684
[1] 2.309091
[1] 2.246305
[1] 2.242938
[1] 2.164103
[1] 1.913043
[1] 2.333333
[1] 2.270588
[1] 1.819512
[1] 2.1
[1] 2.209677
[1] 2.104651
[1] 1.90404
[1] 2.364706
[1] 2.237838
[1] 2.213836
[1] 2.136842
[1] 2.204545
[1] 2.26776
[1] 2.280702
[1] 2.45
[1] 2.150838
[1] 2.244792
[1] 1.989189
[1] 2.027027
[1] 2.164773
[1] 2.348571
[1] 2.034286
[1] 2
[1] 2.242424
```

```
[1] 2.217877
[1] 2.350877
[1] 2.360465
[1] 2.070968
[1] 2.394118
[1] 2.220238
[1] 2.085561
[1] 2.107527
[1] 1.98913
[1] 2.022472
[1] 2.088398
[1] 2.105263
[1] 2.243243
[1] 3.492424
[1] 3.067485
[1] 3.453901
[1] 3.289474
[1] 3.282759
[1] 3.444444
[1] 3.109589
[1] 3.052288
[1] 3.231343
[1] 3.038961
[1] 2.985401
[1] 3.043478
[1] 3.321168
[1] 3.315068
[1] 3.136986
[1] 3.140127
[1] 3.111111
[1] 3.236842
[1] 3.186207
[1] 3.225166
[1] 3.51049
[1] 3.110345
[1] 3.206897
[1] 2.93038
[1] 3.274809
[1] 3.05298
[1] 3.111888
[1] 3.186667
[1] 3.370629
[1] 3.267974
```

```
[1] 3.091503
[1] 3.014085
[1] 3.110345
[1] 3.505882
[1] 3.317568
[1] 2.969325
[1] 3.109489
[1] 2.566474
[1] 3.235294
[1] 3.101911
[1] 3.116788
[1] 3.1
[1] 3.306569
[1] 3.306667
[1] 3.176101
[1] 3.136691
[1] 3.273381
[1] 3.176101
[1] 3.37594
[1] 2.860759
[1] 3.28169
[1] 3.439716
[1] 3.131944
[1] 3.34
[1] 3.229167
[1] 2.922078
[1] 3.151079
[1] 3.033333
[1] 2.97931
[1] 3.294118
[1] 3.282609
[1] 3.100671
[1] 3.28777
[1] 3.458599
[1] 3.225352
[1] 2.964286
[1] 3.208333
[1] 3.055556
[1] 3.06338
[1] 3.38
[1] 3.18
[1] 2.974359
[1] 3.089744
```

```
[1] 3.141892
[1] 3.093333
[1] 3.0375
[1] 3.34507
[1] 3.134969
[1] 3.275362
[1] 2.756098
[1] 3.386207
[1] 3.365385
[1] 3.246575
[1] 3.144654
[1] 3.253623
[1] 2.936416
[1] 3.013889
[1] 3.612676
[1] 3.392857
[1] 3.064706
[1] 3.166667
[1] 3.052632
[1] 3.137931
[1] 3.074534
[1] 3.027211
[1] 3.235669
[1] 3.126582
[1] 3.212329
[1] 3.361111
[1] 3.09697
[1] 3.233333
[1] 3.288235
[1] 3.045161
[1] 3.273333
[1] 3.427536
[1] 2.906832
[1] 2.836735
[1] 3.379747
[1] 3.092857
[1] 3.18543
[1] 3.322368
[1] 3.132075
[1] 2.861842
[1] 3.159509
[1] 3.276596
[1] 3.44375
```

```
[1] 2.834395
[1] 3.012346
[1] 3.445255
[1] NA
[1] 3.272727
[1] 3.210191
[1] 3.054054
[1] 3.099379
[1] 2.597765
[1] 2.564103
[1] 2.671875
[1] 2.427807
[1] 2.661616
[1] 2.539326
[1] 2.532967
[1] 2.818681
[1] 2.433862
[1] 2.577889
[1] 2.617978
[1] 2.546798
[1] 2.716763
[1] 2.872928
[1] 2.684211
[1] 2.576531
[1] 2.515
[1] 3.258427
[1] 2.494624
[1] 2.703297
[1] 2.450867
[1] 2.771429
[1] 2.60241
[1] 2.608247
[1] 2.608939
[1] 2.736842
[1] 2.744565
[1] 2.605263
[1] 2.606742
[1] 2.64
[1] 2.463855
[1] 2.605769
[1] 2.54491
[1] 2.712766
[1] 2.672043
```

```
[1] 2.827381
[1] 2.601093
[1] 2.512077
[1] 2.825301
[1] 2.688442
[1] 2.512821
[1] 2.64
[1] 2.664921
[1] 2.676471
[1] 2.843575
[1] 2.745946
[1] 2.798883
[1] 2.5
[1] 2.754011
[1] 2.878613
[1] 2.932927
[1] 2.705263
[1] 2.641618
[1] 2.573604
[1] 2.456647
[1] 2.776596
[1] 2.722892
[1] 2.477387
[1] 2.670213
[1] 2.350515
[1] 2.661538
[1] 2.836364
[1] 2.688235
[1] 2.818182
[1] 2.403315
[1] 2.725275
[1] 2.673684
[1] 2.684492
```

If else / case_when

Q2.4 Write a couple sentences describing what you want to do with the if else/case_when

Before executing the if else or case_when, write a couple sentences clearly describing what it is that you are trying to accomplish.

A2.4 I want to make a new column that qualitatively tell us how large the penguin is based on body mass

---

Q2.5 Apply an if else or case_when to this dataset

Apply an if else or case_when in some way to this dataset. Describe what you are doing in comments on each line.

```r
penguins.size <- penguins %>%
  # Call the new column size
  mutate(body_size = case_when(
    # When the value is > x then it will output the different categories
    body_mass_g > 5000 ~ "Very Large!!!!!!",
    body_mass_g > 4000 ~ "Large",
    body_mass_g > 2500 ~ "Medium",
    body_mass_g > 0 ~ "Small",
    # Otherwise, make the new column's value "NA"
    .default = "NA"
  ))

head(penguins.size)
```

```
# A tibble: 6 x 9
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
4 Adelie  Torgersen           NA            NA                  NA          NA
5 Adelie  Torgersen           36.7          19.3               193        3450
```

```
6 Adelie  Torgersen           39.3          20.6                 190        3650
# i 3 more variables: sex <fct>, year <int>, body_size <chr>
```

When you have finished, practice the GitHub push/pull:

- Pull to check for updates to the remote branch
- Stage your edits (after saving your document!) by checking the documents you'd like to push
- Commit your changes with a commit message
- Push your changes to the remote branch