

Computational Thinking 2

Seleonalela Hunter - Selesitino

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

```
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 -----
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 become non-conflicting
```

```
library(here)
```

here() starts at /Users/chalchiuhlicuechipilli/Documents/BIOE 176 - Data Science for EEB/

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.

```
x <- 5  
  
# Check if the value of x is greater than 10  
if(x > 10) # if() creates the CONDITION  
{  
  # Paste takes the value stored in x and combines that with a character string  
  print(paste(x, "is greater than 10"))  
} else # else() creates the alternative output if CONDITION is not met.  
{  
  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?

Q1.1 Answer:

The output of the if else statement when $x = 11$ instead of 5 is “is greater than 10.”

```

x <- 11

# Check if the value of x is greater than 10
if(x > 10) # if() creates the CONDITION
{
  # Paste takes the value stored in x and combines that with a character string
  print(paste(x, "is greater than 10"))
} else # else() creates the alternative output if CONDITION is not met.
{
  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:

```

x <- 5 # defines x as = 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.

```
x <- 5 # I had to change the "five" back to "5"
       # or else I could not Render my pdf.

# 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

```
 traceback()
```

No traceback available

```
#Recieved these additional errors, so traceback was unable to run.
#Error during wrapup: not that many frames on the stack
#Error: no more error handlers available (recursive errors?); invoking 'abort' restart
```

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.

```
# define a variable
x <- 11
# when x = 10, output = "x is 10"
# when x = 9, output = "x is less than 10"

# 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:

```
vec <- c(9, 10, 11, 12) # created vector using c()
# named the new vector using "vec" using <-
```

Then we start the for loop:

```
# For 1 through the length of the vector "vec"
for (i in 1:length(vec)) { # run the condition starting from first value to last value in 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.

Q1.2 Answer:

```
# define Vector:  
y <- c(-2, 42, 0, 10) # creating vector y  
  
# create loop:  
for (i in 1:length(y)) {  
  
  # check the value of using nested if-else statements  
  if (y[i] < 0) {  
    # if the element is negative  
    print("value is negative")  
  } else {  
    # if the element is positive  
    if (y[i] > 0) {  
      # if the element equals 10  
      print("value is possitive")  
    } else {  
      # if the element is exactly 0  
      if(y[i]==0){  
        print("value is 0!")  
      }  
    }  
  }  
}
```

```
[1] "value is negative"  
[1] "value is possitive"  
[1] "value is 0!"  
[1] "value is possitive"
```

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:

```
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.

Q1.3 Answer: Niwot Ridge measured stress by observing the concentration of glucocorticoid metabolite present in the fecal samples collected from individual pika's. The stress variable is called “`concentration_pg_m`” and is measured in picograms GCM/gram.

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?

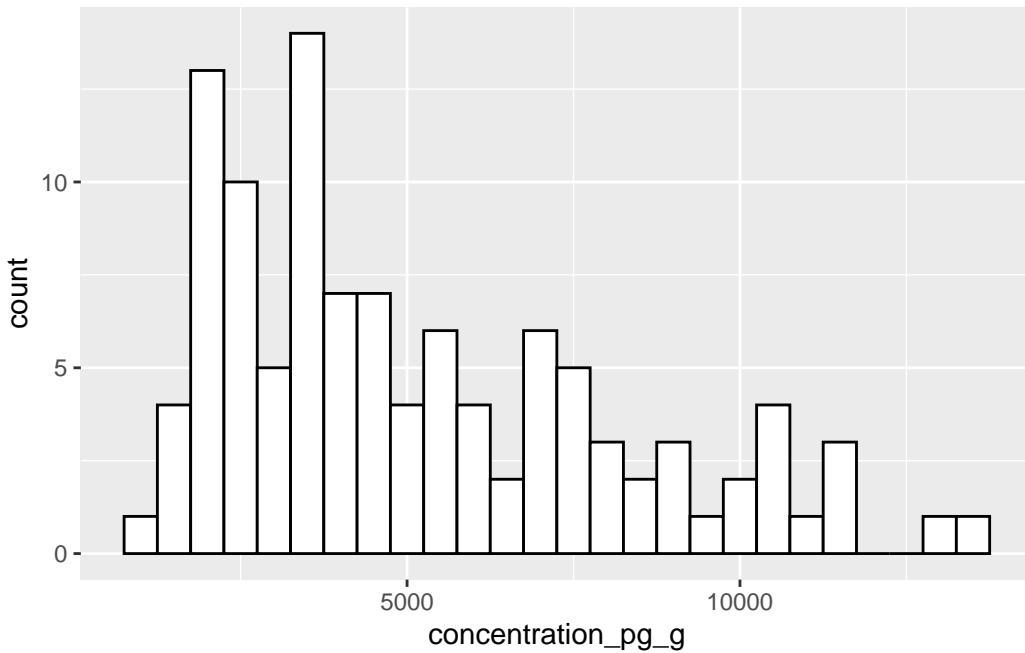
Q1.4 Answer: A row in the dataset represents 1 observation for a single pika.

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")
```

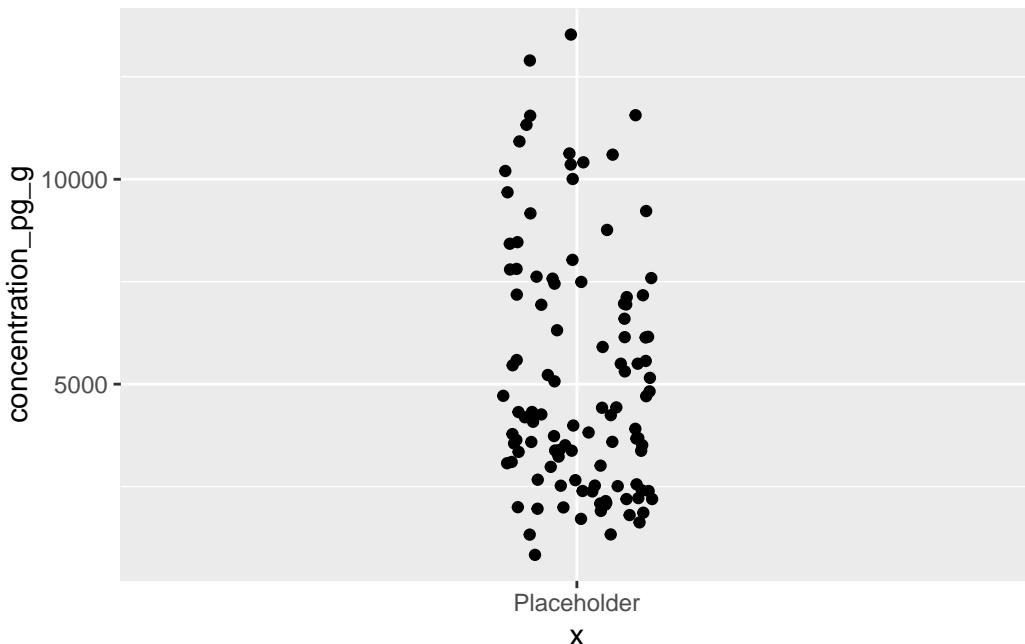


```
# Make a scatterplot with jittered points:  
nwt_pikas %>%  
  # We're adding a little placeholder axis just so we can see the point distribution  
  ggplot(aes(x ="Placeholder",
```

```

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 %>% # naming output of this code
  # 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)))

```

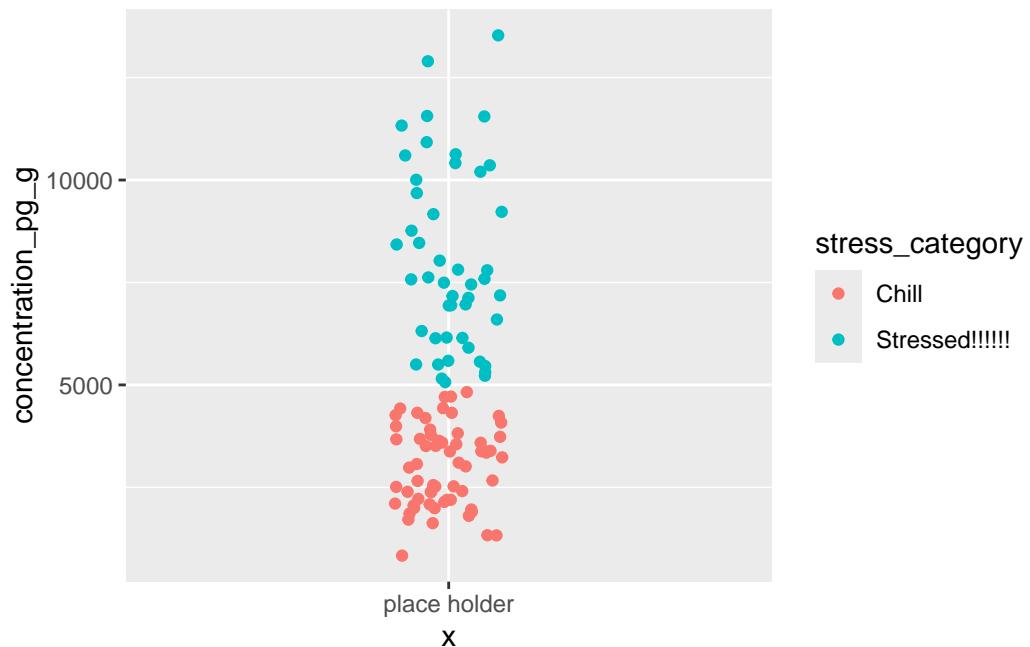
```
# A tibble: 6 x 7
  date      site    station sex concentration_pg_g elev_m stress_category
  <date>    <fct>   <fct>   <fct>        <dbl>   <dbl>   <chr>
1 2018-06-08 Cable  Gate  Cable G~ male       11563.  3343. Stressed!!!!!!
2 2018-06-08 Cable  Gate  Cable G~ male       10629.  3353. Stressed!!!!!!
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.

Q1.5 Answer:

```
nwt_pikas_categ %>%
  ggplot(aes(x = "place holder",
             y = concentration_pg_g,
             color = stress_category)) +
  geom_jitter(width = 0.1)
```



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", # In June or July
    (month == 6 | month == 7) & concentration_pg_g <= 5000 ~ "Early summer chill", # In June or July
    (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)

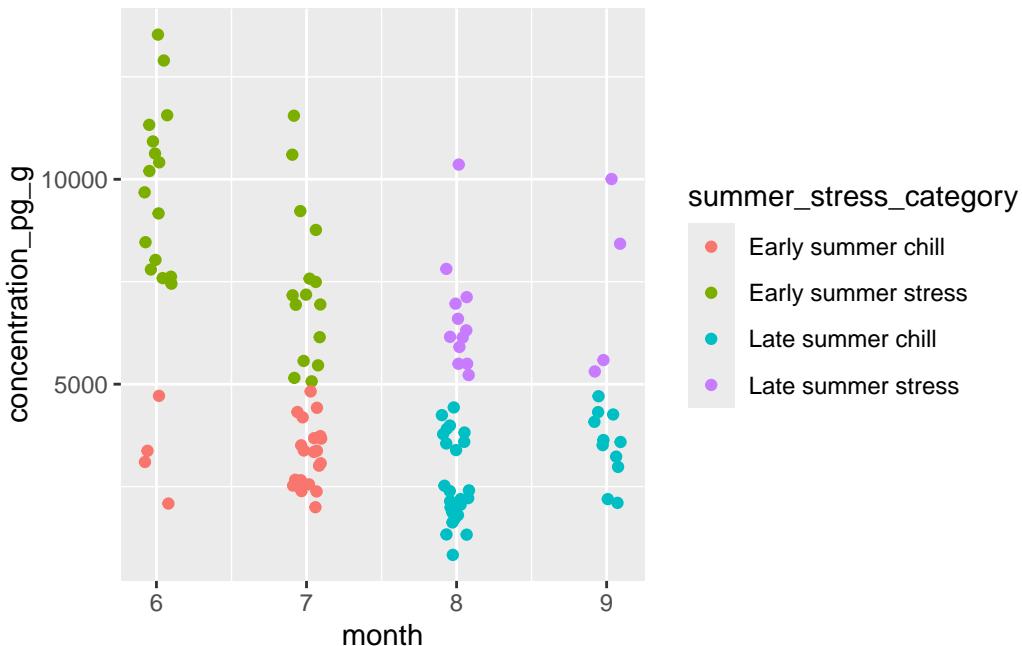
# A tibble: 6 x 11
  date      month site     station    utm_easting utm_northing sex
  <date>    <dbl> <fct>    <fct>        <dbl>          <dbl> <fct>
1 2018-06-08      6 Cable Gate Cable Gate 1      451373        4432963 male
2 2018-06-08      6 Cable Gate Cable Gate 2      451411        4432985 male
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?

Q2.1 Answer: I am using the knz_bison dataset.

```

library(lterdatasampler)
head(knz_bison)

# A tibble: 6 x 8
  data_code rec_year rec_month rec_day animal_code animal_sex animal_weight
  <chr>      <dbl>     <dbl>    <dbl> <chr>       <chr>           <dbl>
1 CBH01      1994      11        8  813         F            890
2 CBH01      1994      11        8  834         F            1074
3 CBH01      1994      11        8 B-301        F            1060
4 CBH01      1994      11        8 B-402        F            989
5 CBH01      1994      11        8 B-403        F            1062
6 CBH01      1994      11        8 B-502        F            978
# i 1 more variable: animal_yob <dbl>

# Tidying the data:

# creating column for bison age + changing weight to from units of Ib to kg:
new_bison <- knz_bison %>% # we are saving this new dataframe as new_bison.
mutate(bison_age = rec_year - animal_yob, # this code gives us a new column with the age of +
weight_kg = animal_weight*0.453592) # this code creates the new column with the weight

# Creating new column with only Female bison:
new_bison$female <- NA #creating a new column for 'female' and filling it with NA's
new_bison$female[new_bison$animal_sex == "F"] <- "F" # filling new column with F if it is true

# Filtering out any Observations of non-Female bison:
new_bisonF <- new_bison %>% # creating new data.frame for output
  filter(!is.na(female)) # filtering out any row with NA

head(new_bisonF)

# A tibble: 6 x 11
  data_code rec_year rec_month rec_day animal_code animal_sex animal_weight
  <chr>      <dbl>     <dbl>    <dbl> <chr>       <chr>           <dbl>
1 CBH01      1994      11        8  813         F            890
2 CBH01      1994      11        8  834         F            1074
3 CBH01      1994      11        8 B-301        F            1060
4 CBH01      1994      11        8 B-402        F            989
5 CBH01      1994      11        8 B-403        F            1062

```

```

6 CBH01           1994      11       8 B-502      F          978
# i 4 more variables: animal_yob <dbl>, bison_age <dbl>, weight_kg <dbl>,
#   female <chr>

unique(new_bisonF$bison_age) # shows how many unique ages are in columnn.

[1] 13 11 10 9 8 7 6 5 4 3 2 1 0 NA 14 12 15 16 17 18 19 20 21

```

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.

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.

Q2.2 Answer:

Using a for loop with an if else statement I want to loop through it the bison_age column, in the tidied version of the knz_bison data set I created. This will let me see which female bison are in their years of high fertility, and which ones are at a stage where they experience reduced fertility because they are either immature or advanced in age.

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.

Q2.3 Answer:

```

ba <- new_bisonF$bison_age #creating a vector for bison age
fertililities <- c()
# telling R to run this for the length of the ba vector.
for (i in 1:length(ba)) {
  # tells R to ignore all NA's in the following if/else() statements
  if (is.na(ba[i])) {
    next
  }
  if(ba[i] >= 2 & ba[i] <= 9) {
    # if the bison is 2:9 R will print "highly fertile"
    fertililities[i] <- "Highly Fertile"
  } else {
    # if bison is less than 2 years old R will Print "immature reduced fertility"
    if(ba[i] < 2){
      fertililities[i] <- "Immature Reduced Fertility"
    } else {
      # if bison is not less than 2 or 2:9 years but older, R will print "older reduced fertility"
      fertililities[i] <- "Older Reduced Fertility"
    }
  }
}

# Check out the first 6 values of the vector
head(fertililities)

```

```

[1] "Older Reduced Fertility" "Older Reduced Fertility"
[3] "Older Reduced Fertility" "Older Reduced Fertility"
[5] "Older Reduced Fertility" "Highly Fertile"

```

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.

Q2.4 Answer: With the if else/case_when() functions I want to create a new column called bison fertility. This column will contain values that will tell me, based off of age, which bison (in each of the 3 groups I defined in the for loop if else statements) are at an age where they are highly fertile, or at an age where they experience reduced fertility due to being too young or old. This will allow me to make a scatter plot relating fertility and age to weight.

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

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

Q2.5 Answer:

```
new_bisonF1 <- new_bisonF %>% # naming output of this code
  # Call the new column bison_fertility
  mutate(bison_fertility = case_when(
    # bison < 2 years old will have a value in the new column = "Immature Reduced Fertility"
    # bison < 2 years will be recorded as "Immature Reduced Fertility"
    bison_age < 2 ~ "Immature Reduced Fertility",
    # bison_age == 2 ~ "Highly Fertile",
    # bison_age == 3 ~ "Highly Fertile",
    # bison_age == 4 ~ "Highly Fertile",
    # bison_age == 5 ~ "Highly Fertile",
    # bison_age == 6 ~ "Highly Fertile",
    # bison_age == 7 ~ "Highly Fertile",
    # bison_age == 8 ~ "Highly Fertile",
    # bison_age == 9 ~ "Highly Fertile",
    # bison between 2 and 9 years old will be recorded as "Highly Fertile"
    bison_age >= 2 & bison_age <= 9 ~ "Highly Fertile",
    # bison older than 9 years will be recorded as "Older Reduced Fertility"
    bison_age > 9 ~ "Older Reduced Fertility",
    .default = "Unknown Age" # any age that is not listed will be inputed as "Unknown age".
  ))
head(new_bisonF1) # checking to see column was made correctly in new data.frame.

# A tibble: 6 x 12
```

```

  data_code rec_year rec_month rec_day animal_code animal_sex animal_weight
  <chr>      <dbl>     <dbl>    <dbl> <chr>      <chr>       <dbl>
1 CBH01      1994      11       8  813       F          890
2 CBH01      1994      11       8  834       F          1074
3 CBH01      1994      11       8  B-301      F          1060
4 CBH01      1994      11       8  B-402      F          989
5 CBH01      1994      11       8  B-403      F          1062
6 CBH01      1994      11       8  B-502      F          978
# i 5 more variables: animal_yob <dbl>, bison_age <dbl>, weight_kg <dbl>,
#   female <chr>, bison_fertility <chr>

```

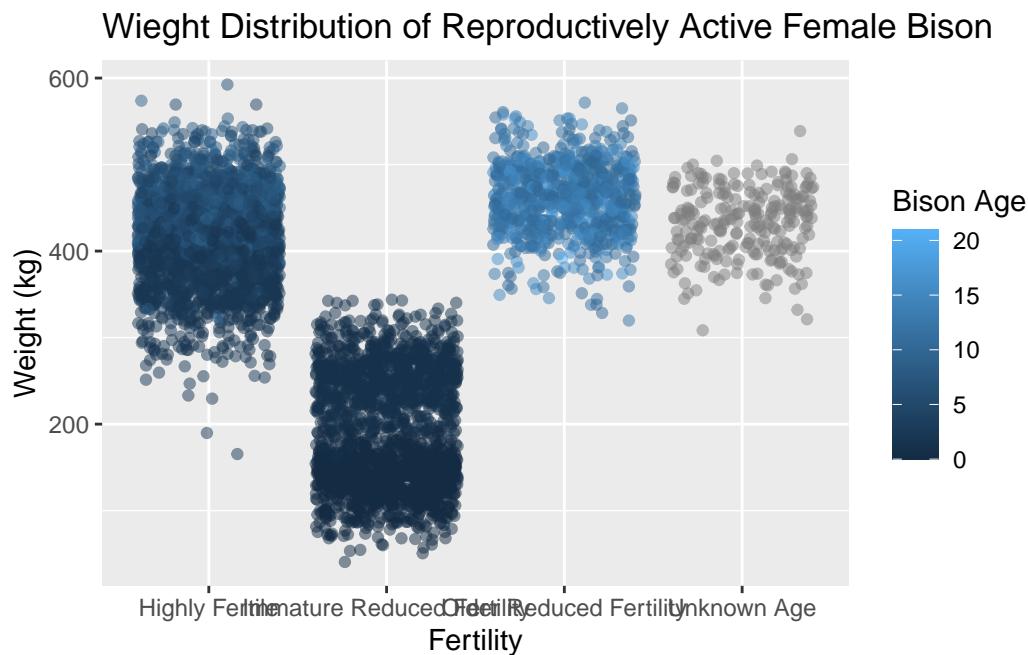
Graph:

I wanted to make some graphs to help compare fertility, age, and weight, to see if there are any noticeable patterns/trends.

```

Female_Bison_Plot1 <- ggplot(data = new_bisonF1, aes(x=bison_fertility, y=weight_kg))+  
  geom_jitter(aes(color=bison_age), alpha=0.5)+  
  labs(x="Fertility", y="Weight (kg)", title = "Wieght Distribution of Reproductively Active F")
Female_Bison_Plot1

```

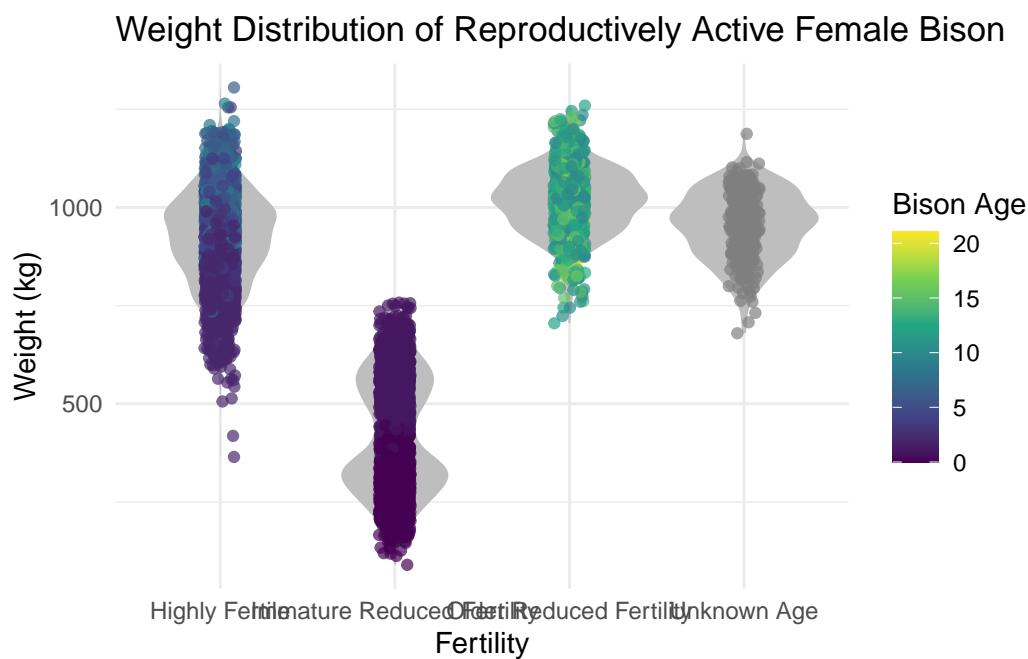


```

# for this graph I did use Chat GPT
# to help with finding what geom_plot could more effectively
# represented the patterns I saw in the first graph.

ggplot(data = new_bisonF1, aes(x = bison_fertility, y = animal_weight)) +
  geom_violin(fill = "grey", color = NA) +
  geom_jitter(aes(color = bison_age), width = 0.09, alpha = 0.7) +
  scale_color_viridis_c() +
  theme_minimal()+
  labs(x="Fertility", y="Weight (kg)", title = "Weight Distribution of Reproductively Active Female Bison")

```



```

Female_Bison_Plot2 <- ggplot(data = new_bisonF1, aes(x=bison_fertility, y=weight_kg))+  

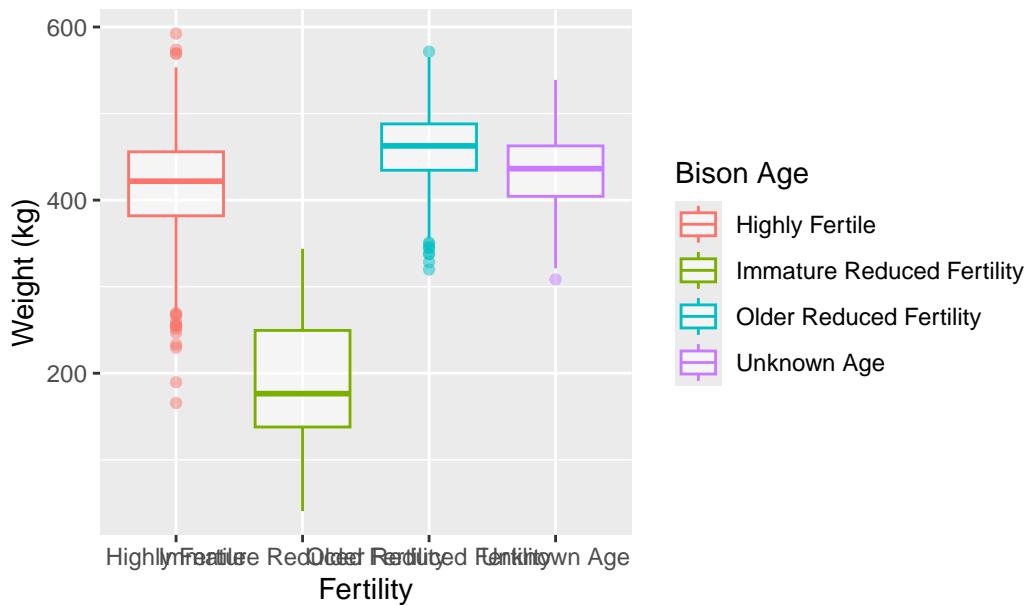
  geom_boxplot(aes(color=bison_fertility), alpha=0.5)+  

  labs(x="Fertility", y="Weight (kg)", title = "Wieght Distribution of Reproductively Active Female Bison")

```

Female_Bison_Plot2

Wieght Distribution of Reproductively Active Female Bison



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