Lab 01

ENVX2001 Applied Statistical Methods

Contents

Learning Outcomes

In this lab, we will learn how to:

1. Explain the differences between (i) samples and populations (ii) standard error and standard deviation;
2. Use R to perform basic data analysis tasks related to exploratory data analysis
3. Present their code and results using RMarkdown.

Specific goals

By the end of this lab, you should be able to:

- ☐ Calculate means, medians, and standard deviations
- ☐ Create graphs using ggplot2
- ☐ Subset and organise data
- ☐ Understand why different types of summary statistics exist, and when to use each one

Hi!

Welcome to the first lab for ENVX2001! Before we start, make sure you have access to the latest versions of R and RStudio.

Preparation

If you are attending this lab in person, your demonstrators will treat you to a short presentation. If you are completing this lab remotely, please do the following:

- Download these files: water.xlsx, browsing_data_2003_2020_2.csv
- Read the abstract of this article: MacNulty et al., 2025

A Note on Generative AI (GenAI)

GenAI is a powerful tool that can help you learn and understand the concepts we cover in ENVX2001. However, for the first six weeks of this course, we ask that you please refrain from asking GenAI for help.

There are two crucial skills we want to help you develop this semester:

- Problem solving tenacity
- Statistical intuition

In our experience, these skills are best learned without the help of AI. In weeks 7-12, when we introduce more complex statistical concepts, is where GenAI can really help.

Aspen grove in Yellowstone National Park, from Wikimedia Commons (2008), by John Fowler.

First Thoughts

To recap the story from our presentation (or article, for remote students), a study by Ripple et al. (2025) found new evidence to support the already popular idea that wolves are a keystone species in Yellowstone National Park. By modelling the rate of willow regrowth before and after wolves were reintroduced to the park, Ripple and his team found that the wolves had an enormous, positive effect on the park's ecosystem [1].

However, Ripple's study was criticized by another group of scientists; Daniel MacNulty and his team argued that Ripple's methods were flawed, and that while the wolves of Yellowstone National Park may have caused a weak trophic cascade in some areas of the park, this effect was not nearly as strong or as universal as Ripple had claimed [2].

Grey wolf in Yellowstone National Park, from Wikimedia Commons (2013), by Mike van Dalen.

i For you to consider

Based on what you saw in the presentation, or read from the article, what are your first thoughts on the situation? Do you think MacNulty was right to criticize Ripple's methods?

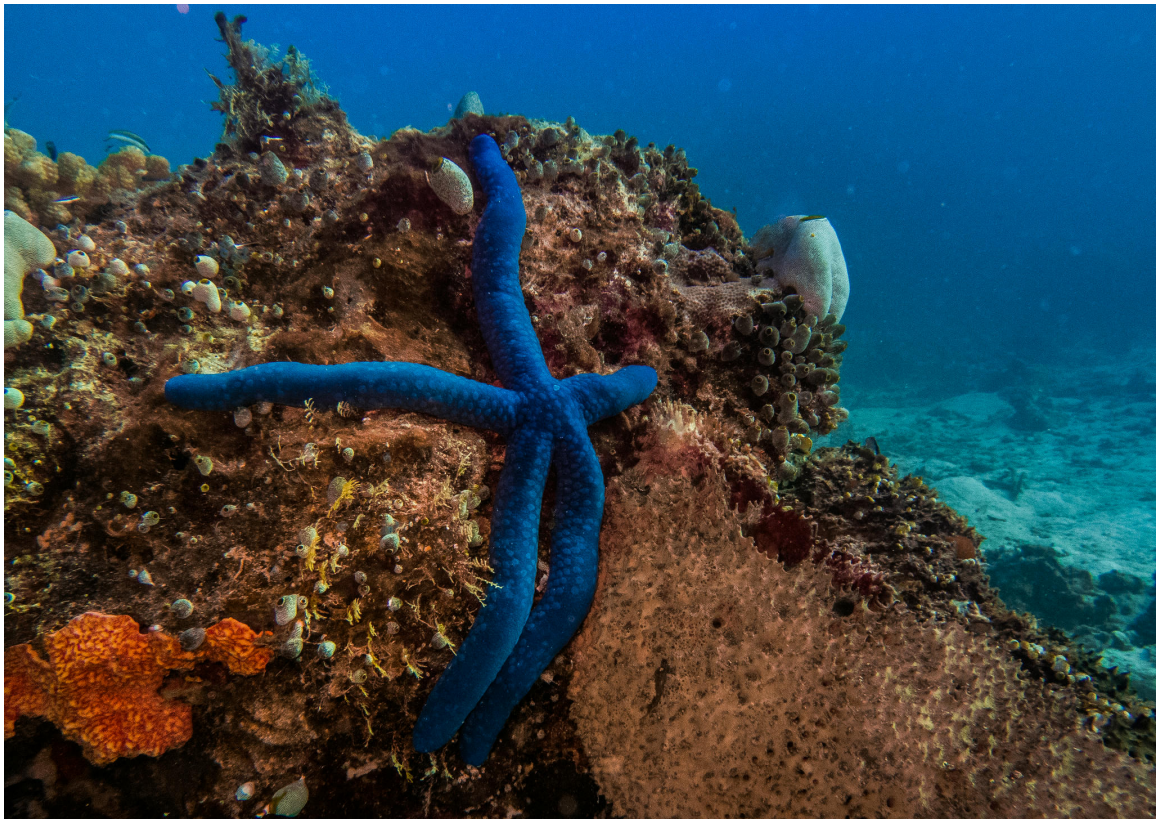
What we think

Without any idea of what Ripple's study entails, it is difficult to judge the validity of MacNulty's criticisms. However, criticisms in science should always be welcomed and taken seriously, even if they challenge established ideas. Let's run through some exercises to sharpen our statistical intuition before we return to this problem.

Part 1: Organising data

Exercise: Blue Sea Stars

The following exercise involves a fabricated story and simulated data



Blue sea star (Linckia laevigata), *from Wikimedia Commons (2017), by João D'Andretta.*

A marine scientist (we'll call her Stella) is studying benthic invertebrates on Lady Elliot Island, and notices that the blue sea stars (*Linckia laevigata*) from this island seem to be smaller than those in other parts of the Great Barrier Reef. She wonders if her eyes are deceiving her.

To get to the bottom of this, she collects 16 sea stars from Lady Elliot Island and measures one random arm from each of them to the nearest 0.1 cm. She knows that the typical length of a blue sea star's arm is around 11.5 cm [3].

i Building Habits

Find out where Lady Elliot Island is located on a map. Why might the sea stars there be different in size to sea stars from other parts of the Great Barrier Reef? It is always a good idea to think about the context behind our data before we analyse it.

Food for thought

Lady Elliot Island marks the southern end of the Great Barrier Reef. According to a study by Thompson and Thompson (1982), blue sea stars shrink noticeably within a week of low food availability. The same study also notes that larger sea stars of this species tend to live in deeper waters.

It may be the case that the reefs of Lady Elliot Island are especially shallow. It could also be that there is less food for sea stars on Lady Elliot Island because it is so isolated from the rest of the Great Barrier Reef. We will leave you to look further into this topic if you are interested.

Here is the data Stella gathers:

i Stella's data

CODE

```
stars ← c(
  10.3, 11.0, 10.5, 10.0, 11.3, 14.5, 13.0, 12.1, 12.1,
  9.4, 11.3, 12.0, 11.5, 9.3, 10.1, 7.6
)
```

Notice that we use the `c()` function to specify a data *vector*. A vector is a collection of similar objects; in this case, numbers.

To make it easier to recall this vector, we can give it the name '`stars`' using the ← symbol. Now, if we ever want to recall this list of numbers again, we can do so easily:

i Recall Stella's data

CODE

```
stars # recalls Stella's data
```

OUTPUT

```
 [1] 10.3 11.0 10.5 10.0 11.3 14.5 13.0 12.1 12.1  9.4 11.3 12.0 11.5  9.3 10.1
[16]  7.6
```

The # symbol is used to make a *comment*. Comments are very useful, and you should get into the habit of including them in your code.

> **i** Building Habits
>
> Make a code chunk. You can do this using the +C button on the top of your screen (ask a demonstrator for help if you are confused).
>
> Type in the following lines of code:
>
> mean(c(1,2,3,4,5))
>
> median(c(0,0,1,45,459,2,49,1))
>
> And describe what each of them do using comments.

Food for thought

This is how we would have done it:

```
CODE
mean(c(1,2,3,4,5)) # takes the average of 1,2,3,4,and 5
```

```
OUTPUT
[1] 3
```

```
CODE
median(c(0,0,1,45,459,2,49,1)) # finds the median in the sequence: 0,0,1,1,2,45,49,459
```

```
OUTPUT
[1] 1.5
```

Notice that the comments do not show up in your outputs. This is because the # tells R not to read them as code.

Comments are not only great to help other people understand your code, but also to remind yourself of what you did at a later date. The # key is your friend.

To find the average arm length of Stella's sea stars, we can use the `mean()` function:

```
CODE
mean(stars) # Notice that instead of re-typing our data, we can recall it using the name we gave
it earlier: 'stars'.
```

```
OUTPUT
[1] 11
```

Tip

The average is not the only summary statistic we can calculate; here are some others:

CODE

```
median(stars) # Median - the middle number in Stella's data
```

OUTPUT

```
[1] 11.15
```

CODE

```
sd(stars) # Standard deviation - the spread of Stella's data
```

OUTPUT

```
[1] 1.626038
```

The `summary()` function can give you many different summary statistics at once:

CODE

```
summary(stars) # Also gives you the 1st and 3rd quartiles, minimum value, and maximum value
```

OUTPUT

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   7.60   10.07   11.15   11.00   12.03   14.50
```

That's a lot of functions to remember! Don't worry, you can always ask R for help. Use the `?` symbol. For example, to find out more about the `mean()` function, you can type `?mean` into your console (your console is located at the bottom of your screen).

Black noddy (Anous minutus)*, a seabird found on the palm trees of Lady Elliot Island. From Wikimedia Commons (2007), by Dalgo UK*

Practice: Even More Sea Stars

Stella tells her friends about her study, and they all decide to visit Lady Elliot Island to help her collect more samples. Here are the datasets that each of Stella's friends collects; we will name them stars_1, stars_2, etc. :

i Data collected by Stella's friends

```
CODE
stars_1 ← c(
  11.3, 15.0, 9.5, 10.0, 11.0, 11.2, 12.2, 8.5, 9.1, 9.5,
  11.4, 12.4, 13.0, 8.3, 11.0, 12.5
)
stars_2 ← c(
  14.0, 11.5, 6.5, 9.1, 9.3, 15.0, 11.0, 9.2, 12.7, 8.5,
  11.8, 8.8, 8.3, 9.1, 11.6, 14.0
)
stars_3 ← c(
  9.5, 12.3, 13.6, 8.2, 15.8, 7.7, 10.1, 11.3, 11.5, 12.9,
  10.1, 8.3, 7.5, 8.9, 9.1, 10.0
)
stars_4 ← c(
  10.0, 12.1, 16.0, 8.0, 11.3, 14.0, 12.0, 13.5, 10.1,
  10.5, 10.8, 9.1, 14.3, 9.0, 15.5, 8.5
)
stars_5 ← c(
  7.0, 8.5, 10.5, 7.1, 11.3, 9.0, 9.5, 12.1, 8.0, 9.3,
  10.9, 7.3, 8.5, 9.0, 8.1, 12.4
)
```

i Sharpen your skills

Find the mean and standard deviation for each of these datasets.

Solution

We can apply the `mean()` and `sd()` functions to each dataset individually:

CODE
```
mean(stars_1) # mean of stars_1
```

OUTPUT
```
[1] 10.99375
```

CODE
```
sd(stars_1) # standard deviation of stars_1
```

OUTPUT
```
[1] 1.799803
```

CODE
```
mean(stars_2) # mean of stars_2
```

OUTPUT
```
[1] 10.65
```

CODE
```
sd(stars_2) # standard deviation of stars_2
```

OUTPUT
```
[1] 2.424321
```

CODE
```
mean(stars_3) # mean of stars_3
```

OUTPUT
```
[1] 10.425
```

CODE
```
sd(stars_3) # standard deviation of stars_3
```

OUTPUT
```
[1] 2.328233
```

CODE
```
mean(stars_4) # mean of stars_4
```

OUTPUT
```
[1] 11.54375
```

CODE
```
sd(stars_5) # standard deviation of stars_5
```

OUTPUT
```
[1] 1.671671
```

When you have many datasets, repeating this process can become tedious. We will show you shortcuts and workarounds in the coming weeks to make these types of tasks easier.

What do we have here? Another collection of numbers? Let's make a vector out of them!

Sharpen your skills

Make a vector that contains the means of each of Stella and her friends' datasets.

Name this vector `stars_means`.

Tip

Your vector should have 6 entries - one for the mean of Stella's own dataset `stars`, and one for the means of each of her friends' datasets `stars_1`, `stars_2`, etc.

Solution

We can use the `c()` function to create our vector:

```
CODE
stars_means ← c(
  mean(stars_1), mean(stars_2), mean(stars_3),
  mean(stars_4), mean(stars_5), mean(stars)
)
# The entries of this vector might look strange, but they are really just individual numbers;
mean(stars_1) is a number, and so is mean(stars_2), etc.
```

Whenever you want to store a collection of numbers, you can make them into a vector. These vectors will stay under R's 'environment' tab (at the top right of your screen).

Let's see what our new vector looks like:

```
CODE
stars_means # A vector with 6 entries.
```

```
OUTPUT
[1] 10.99375 10.65000 10.42500 11.54375  9.28125 11.00000
```

We have just created a brand new dataset out of six pre-existing ones. Let's find out more about this new dataset - what is its mean and standard deviation?

Sharpen your skills

Calculate the mean and standard deviation of `stars_means`. How do these values compare to the mean and standard deviation of Stella's original dataset, `stars`?

Solution

Because `stars_means` is a vector, we can apply the `mean()` and `sd()` functions to it:

CODE

```
mean(stars_means) # The average value of stars_means
```

OUTPUT

```
[1] 10.64896
```

CODE

```
sd(stars_means) # The standard deviation of stars_means
```

OUTPUT

```
[1] 0.7698764
```

Notice that `stars_means` has a very similar average to `stars` (10.6 vs 11), but a much smaller standard deviation (0.77 vs 1.62).

The more friends Stella invites, and the more samples they gather, the smaller the standard deviation of `stars_means` will become.

The mean of Stella's original dataset, `stars`, is called the **sample mean**, and the standard deviation of `stars` is called the **sample standard deviation**.

The mean of our new dataset, `stars_means`, is called the **average of the sample means**, and the standard deviation of `stars_means` is an estimate of the **standard error** in Stella's data.

Depending on how many friends Stella has, and how many sea stars each of them measures, the average of the sample means may serve as a good estimate of the **population mean** (i.e. the true average arm length of all the blue sea stars on Lady Elliot Island, if we could measure each and every one of them). If Stella had hundreds of friends, `stars_means` would have hundreds of entries, and its mean would approach the true population mean while its standard deviation approaches the true standard error.

Stella was very lucky. In reality, we won't always have hundreds of friends (or even five) to help us collect additional data. Because of this, we often need to *approximate* the population mean and standard error based on a limited number of samples. We can do this using the following equations:

$$\bar{X} \approx \mu$$

$$SE \approx \frac{s}{\sqrt{n}}$$

Where $\bar{X}$ is the sample mean, μ is the population mean, SE is the true standard error, s is the sample standard deviation, and n is the sample size (how many sea stars Stella measured).

Section Summary

So, what did Stella find? Were the blue sea stars of Lady Elliot Island really smaller than blue sea stars elsewhere? To answer that question, we need to carry out a statistical test.

If you already know how to perform a one-sample t-test, or a one-way ANOVA, give it a try. Otherwise, we will go through how to run both of these tests next week. Then, you can revisit this lab and apply one of them to Stella's data.

Before we move onto the next section, now is a good time to take a 5-minute break.

Part 2: Making Graphs

Exercise: Water Chemistry

The following exercise involves real data from G. M. Lovett, K. C. Weathers, and W. V. Sobczak [4]

Diamond Notch Falls in Westkill Mountain, one of the many mountains in New York's Catskill Park. From Wikimedia Commons (2021), by Daniel Case.

In the year 2000, Gary Lovett and his research team measured water chemistry in the streams of the Catskill Mountains. They were concerned that growing levels of industrial activity in the area may affect surrounding forests, and they needed a way to keep track of pollutant levels in the environment.

For this exercise, we will focus on sulphates. It is worth noting that Lovett's original study focused on nitrates instead.

Reading data

Unlike Stella's data, which we could type directly into R, Lovett's data is stored in a separate Excel file. We need to load this file into R before we can analyse the data inside.

To do this, we need to install a package. Packages are add-ons you can download from the internet, kind of like expansion packs in a video game.

The package we want to install is called "readxl". We can do this using the `install.packages()` function.

> **Tip**
>
> The code we need to execute is: `install.packages("readxl")`. However, we do not want to put this line into our coding script. Instead, we want to put it into our console. The console is the window at the bottom of the screen, where you may see lines of blue code that you previously executed.

> **Warning**
>
> Always execute one-time operations, such as installing packages and inspecting file paths, inside your console. If you include these tasks in your coding script, they will execute every time you render the document. Not only will this slow down the rendering process, it will also result in a messy html file with irrelevant outputs.

Now, we can use this brand new package to read our excel file:

```
CODE
library(readxl) # Activates the package 'readxl'
water <- read_excel("data/water.xlsx") # Reads the file 'water.xlsx' into R as a table

# Note that we renamed this table 'water' using the <- operator.
```

Tip

In the function `read_excel("data/water.xlsx")`, the `data/` part tells R which folder to search. If you do not have a folder called "data" on your computer, then you should run `read_excel("water.xlsx")` instead. To manually reset R's search location (also called its *directory*), go to the very top of your screen and find "Session -> Set Working Directory -> Choose Directory...".

Warning

A common error you may encounter is "no such file or directory". This either means you have misspelled your file name (remember to include .xlsx and use ""), or that R is searching for your file in the wrong folder. To check which folder R is searching, run the command `getwd()` in your console.

Tip

A good way to organise your data is to keep it close to your script. Wherever you save your qmd. file, make sure your data is also saved in the same folder.

Building Habits

After reading the data, it is good practice to verify that everything has been imported correctly. Use the `str()` function to check the structure of your new dataset, `water`.

Food for thought

This is what we found:

CODE

```
str(water) # Checks the structure of the dataset named 'water'
```

OUTPUT

```
tibble [39 × 5] (S3: tbl_df/tbl/data.frame)
 $ SO4                           : num [1:39] 50.6 55.4 56.5 57.5 58.3 63 66.5 64.5 63.4 58.4 ...
 $ Sampling_Elevation_Nearest_100m: num [1:39] 700 600 600 700 600 500 500 600 700 700 ...
 $ NO3                           : num [1:39] 24.2 25.4 29.7 22.1 13.1 27.5 28.1 31.2 22.6
35.9 ...
 $ Cl                            : num [1:39] 15.5 16.4 17.1 16.8 18.3 15.7 26.9 22 21.3 29.8 ...
 $ Creek_Formally_Named          : chr [1:39] "Yes" "No" "Yes" "Yes" ...
```

Note:

"tibble" means R recognises your data as a table.

"$ SO4" means SO4 is a column in this table. If there were other columns, each of them would have a "$" in front as well.

"num" means R recognises the SO4 column as numeric.

"[1:39]" means there are 39 entries in the SO4 column.

Subsetting data

Subsetting means keeping some parts of your data while excluding others. For example, we may want to keep a specific column, or remove a specific row.

The easiest way to subset data is to use the `[,]` operator. You can use the space in front of the comma `[*,]` to select rows, and the space after the comma `[,*]` to select columns.

i Building Habits

i) Use the `[,]` operator to select the third row and first column of the `water` dataset.

ii) Use the `[,]` operator to select the ninth row of the `water` dataset.

iii) Use the `[,]` operator to select the first column of the `water` dataset.

Food for thought

This is what we did:

i)

CODE

```
water[3, 1] # Third row, first column
```

OUTPUT

```
# A tibble: 1 × 1
    SO4
  <dbl>
1  56.5
```

ii)

CODE

```
water[9,] # Ninth row
```

OUTPUT

```
# A tibble: 1 × 5
    SO4 Sampling_Elevation_Nearest_100m   NO3    Cl Creek_Formally_Named
  <dbl>                           <dbl> <dbl> <dbl> <chr>
1  63.4                             700  22.6  21.3 Yes
```

iii)

CODE

```
water[,1] # First column
```

OUTPUT

```
# A tibble: 39 × 1
     SO4
   <dbl>
 1  50.6
 2  55.4
 3  56.5
 4  57.5
 5  58.3
 6  63
 7  66.5
 8  64.5
 9  63.4
10  58.4
# ℹ 29 more rows
```

Tip

You can use the operator `$` instead of `[,]` to select specific columns by name. For example:

CODE

```
water$SO4 # Selects the column named 'SO4'
```

OUTPUT

```
 [1] 50.6 55.4 56.5 57.5 58.3 63.0 66.5 64.5 63.4 58.4 70.6 56.9 56.7 56.0 60.4
[16] 67.8 70.8 58.6 59.5 55.5 63.4 57.8 55.1 65.5 62.7 72.1 63.4 68.5 65.8 69.2
[31] 66.7 59.3 61.1 62.1 70.4 62.1 64.6 61.4 56.9
```

Notice that we went from a table to a collection of numbers (recall that a collection of numbers is called a numerical *vector*).

You can apply functions to these numbers, as you would to any other numerical vector. For example, let's find out their median value:

CODE

```
median(water$SO4) # The median value of column SO4
```

OUTPUT

```
[1] 62.1
```

Building Habits

Summary statistics are a great way to quickly make sense of your data.

Use the `summary()` function on column SO4. Do you think the values in this column are symmetrically distributed?

Food for thought

CODE

```
summary(water$SO4) # summary statistics for the SO4 column.
```

OUTPUT

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  50.60   57.65   62.10   61.92   65.65   72.10
```

The median and mean are very similar, so the distribution is probably close to symmetrical.

Note that if the mean is greater than the median, the distribution would be right-skewed, and if the mean is less than the median, the distribution would be left-skewed.

Catskill Mountains in the fall. From Wikimedia Commons (2016), by Daniel Case.

Practice: Now You See Me

Graphs are a great way to visualise data. The `ggplot2` package is a popular package made for this very purpose. It is based on the grammar of graphics (as if the grammar of English was not enough), which you can look into in your own time.

Let's activate this package:

```
CODE
library(ggplot2)
```

> Tip
>
> If R cannot find `ggplot2` in your library, you may not have installed ggplot2 before. In that case, run `install.packages("ggplot2")` in your console to install it.
>
> Even better, install the package "tidyverse". This package includes ggplot2, as well as many other useful packages such as dplyr.

Of course, there are other ways to create graphs in R; but we recommend using `ggplot2` because it is flexible and intuitive.

Basically Yo-Chi

`ggplot2` is very similar to Yo-Chi. Really, it is. What is the first thing you do at a Yo-Chi? You grab a cup. Let's grab a cup:

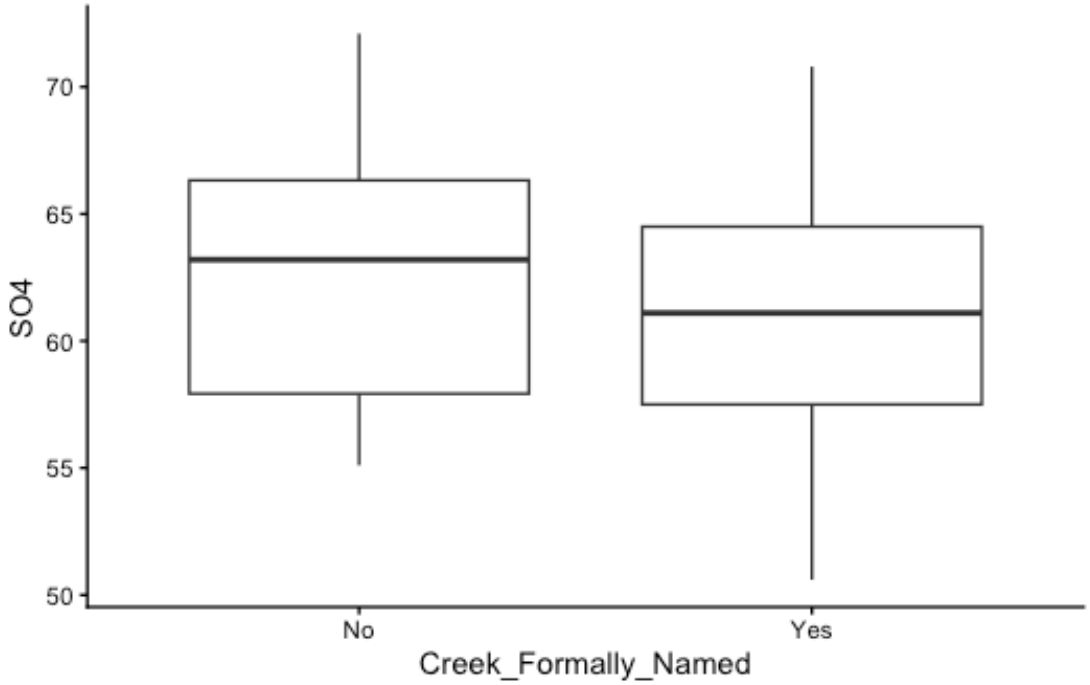
> Sharpen your skills
>
> The basic template for `ggplot2` is:
>
> `ggplot(data = _, aes(x = _, y = _)) + geom_()`.
>
> Think of this as an empty cup. You can put things into it.
>
> The argument `data =` tells R which dataset to reference (in this case, we will use `water`).
>
> The `x =` and `y =` arguments specify which column we want to use as our x values, and which column we want to use as our y values. Play around with different columns here, and see what you find.
>
> The argument `geom_()` tells R the type of graph you want. Here are some options you can try: `geom_boxplot()`, `geom_line()`, `geom_point()`, `geom_smooth()`, `geom_jitter()`.
>
> Make a few different graphs from the `water` dataset, and pick your favourite one!

Solution

These are the graphs we made; just plain yoghurt (signature tart) for now:

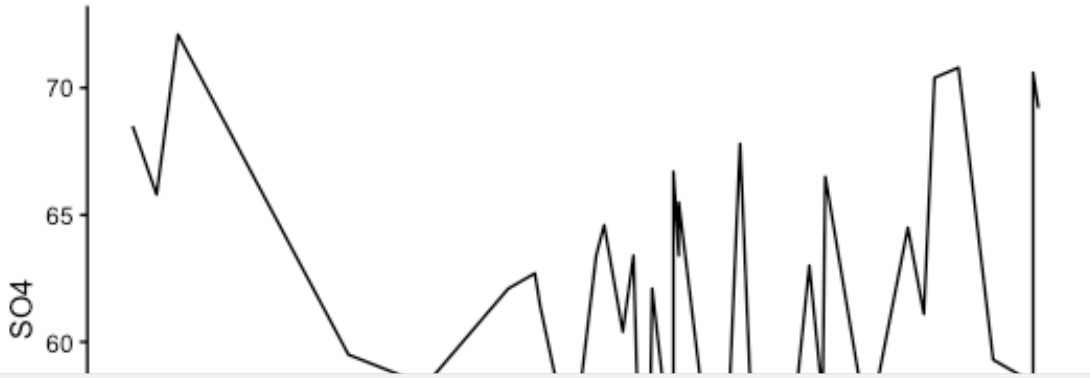
CODE

```
ggplot(data = water,
       aes(x = Creek_Formally_Named, y = SO4)) +
  geom_boxplot() +
  theme_classic() # Box plot (we added a 'classic' theme to erase the grey background)
```

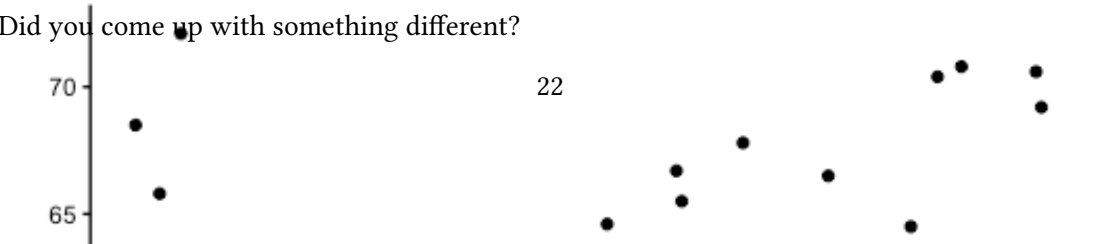


CODE

```
ggplot(data = water,
       aes(x = NO3, y = SO4)) +
  geom_line() +
  theme_classic() # Line plot (cool, but hard to interpret!)
```



Did you come up with something different?



Tip

To keep your code neat, hit 'enter' after each + sign. This starts a new, indented line and prevents overcrowding.

Choose your flavour

We can customise our Yo- I mean our graphs by colour-coding them. To do this, we take our basic template:

```
ggplot(data = _, aes(x = _, y = _)) + geom_()
```

And add the arguments `colour =` and `fill =` into the `geom_()` bracket, like this:

```
ggplot(data = _, aes(x = _, y = _)) + geom_(colour = _, fill = _)
```

Sharpen your skills

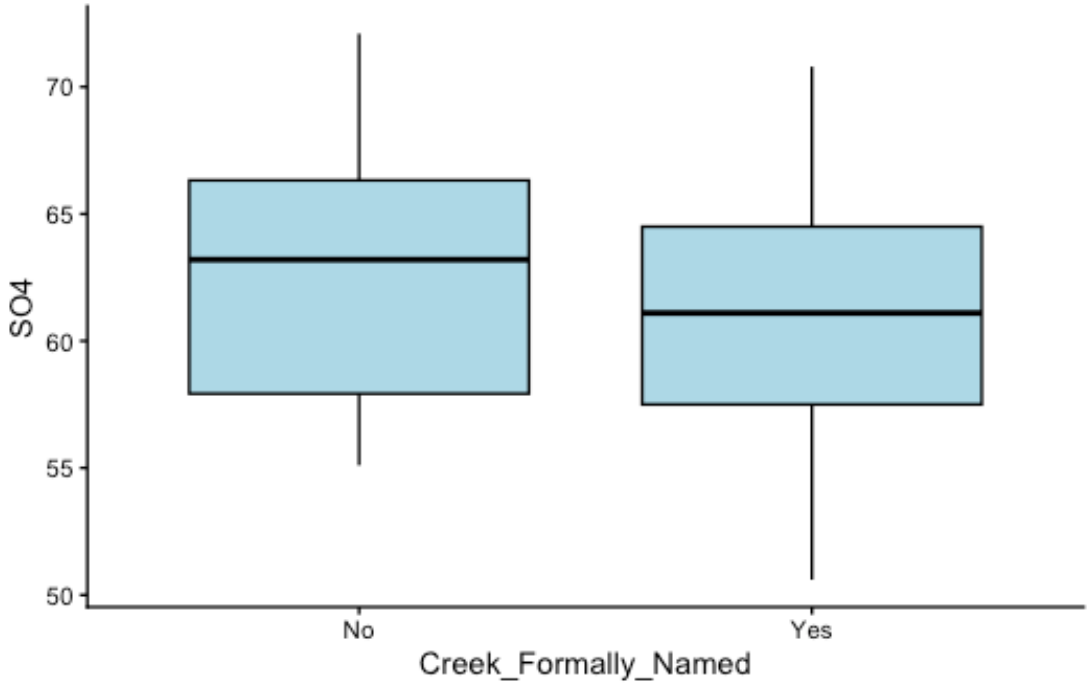
Take your favourite graph from before, and turn it into a different colour. Simple options you can try include: `colour = 'red'`,`colour = 'lightblue'`, `fill = 'grey'`, etc. Name a colour, and it probably exists. For any other colours, look up their HEX codes.

Solution

Our box plot, turned blue.

CODE

```
ggplot(data = water,
       aes(x = Creek_Formally_Named, y = SO4)) +
  geom_boxplot(colour = 'black', fill = 'lightblue') +
  theme_classic()
```



Out of all the graphs that `ggplot2` offers, histograms and bar plots are kind of special. This is because they do not take a y-argument; in fact, the y-argument for both of these graphs is by default 'count'. (If you are confused about why this is the case, reach out to one of your demonstrators.)

Let's practice making a histogram:

Sharpen your skills

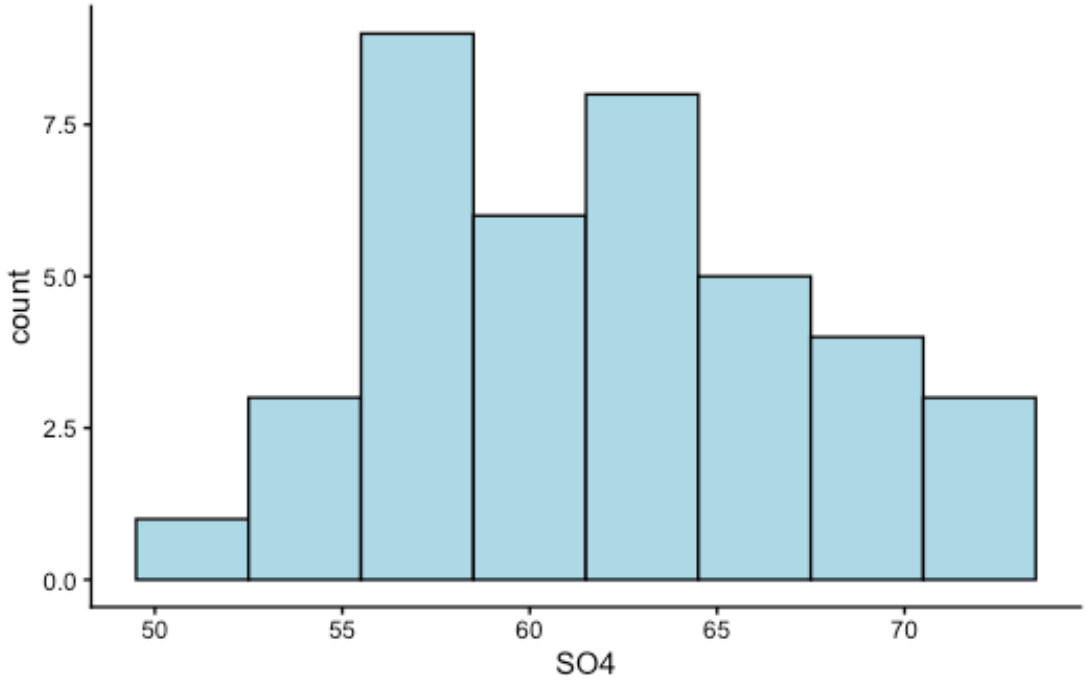
Make a histogram of the SO4 column in `water`. Use `geom_histogram()` to create a histogram, and use the argument `geom_histogram(binwidth = _)` to adjust its appearance.

Earlier, we guessed from our summary statistics that the values in column SO4 are symmetrically distributed. Is that the case?

Solution

Here is our histogram:

```
CODE
ggplot(data = water, aes(x = SO4)) +
  geom_histogram(colour = 'black', fill = 'lightblue',
                 binwidth = 3) +
  theme_classic()
```



The distribution does look reasonably symmetrical.

Tip

The table below contains heuristic guidelines on which graphical summary to use based on the number of observations. Commands refer to arguments in `ggplot2`, not base R.

observations	graphics	command
1-5	plot raw data	`geom_point()`
6-20	boxplot	`geom_boxplot()`
20 or more	histogram	`geom_histogram()`

⚠ Warning

If you prefer a more formal way to detect skewness, you may be tempted to use the `skewness()` function from the `moments` package. This function calculates the skewness coefficient of a dataset or vector.

CODE

```
library(moments) # Install this package if you haven't already by running:
`install.packages("moments")` in your console
skewness(water$SO4) # Calculates the skewness coefficient of the SO4 column
```

OUTPUT

```
[1] 0.1571807
```

A low skewness coefficient (anything < 0.5 is usually considered low) means our distribution should be reasonably symmetrical.

However, skewness coefficients become hard to interpret in the case of multi-modal distributions. In general, we recommend sticking to histograms.

Add toppings

We have our cup, we've chosen our flavours, now it's time to add our toppings.

You can add a theme to your graph using the `theme_()` argument, like this:

`ggplot(...) + geom_(...) + theme_()`

You can also change the axis labels using the `labs()` argument, like this:

`ggplot(...) + .geom_(...) + theme_() + labs(title = _, x = _, y = _)`

ℹ Sharpen your skills

Take your histogram from earlier and re-label its x-axis. Lovett's team measured sulphate concentration in micromoles per litre.

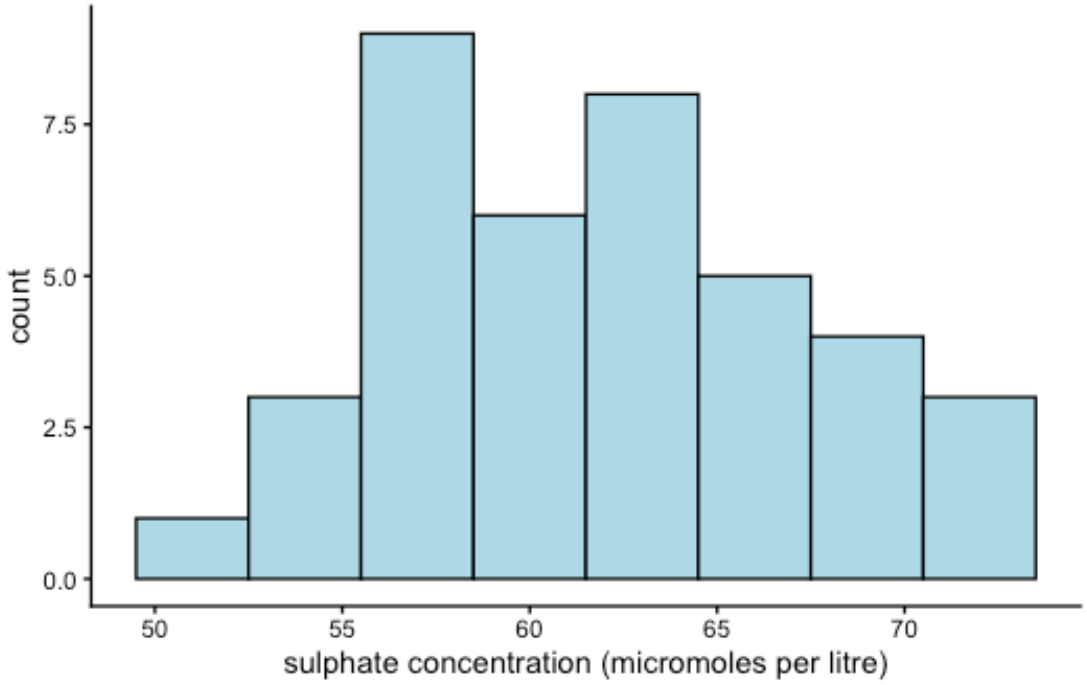
Choose a theme for your graph as well. Some cool themes to try out are: `theme_classic()`, `theme_minimal()`, `theme_bw()`, `theme_dark()`.

Solution

To change our x-axis label, we should use the argument `labs(x = _)`. We can leave `y_` and `title_` arguments out, since we are not interested in changing the y-axis label or the title.

CODE

```
ggplot(data = water, aes(x = SO4)) +
  geom_histogram(colour = 'black', fill = 'lightblue',
                 binwidth = 3) +
  theme_classic() +
  labs(x = 'sulphate concentration (micromoles per litre)')
```



For theme, I stuck with `theme_classic()` - a personal favourite.

A freshwater snail (Lymnaea stagnalis) *in algae. From Wikimedia Commons (2009), by Peter Pfeiffer.*

Section Summary

According to a study in Finland, it takes upwards of 500 micromoles/litre of sulphate to cause noticeable harm to aquatic crustaceans and molluscs [5]. The values from Lovett's study were far below this (check our histograms from earlier). So, it might seem like the freshwater ecosystems of Catskill Park are safe… for now.

However, we must take note of two important thing: Firstly, harmful pollutant levels can be very ecosystem-specific. It is entirely possible for one ecosystems to be more sensitive than another to the same level of sulphate pollution. Secondly, sulphate was not the only pollutant Lovett's team measured. In fact, their main concern was nitrate saturation (NO3).

We will leave it to you to figure out whether nitrate concentrations in the creeks of Catskill Park were above or below environmentally accepted levels at the time of Lovett's study.

For now, let's take a 5-minute break, and then we will revisit the Yellowstone controversy one last time.

Part 3: Handling Complexity

Case Study: Back to Yellowstone

The following case study involves real data from N. T. Hobbs, D. B. Johnston, K. N. Marshall, E. C. Wolf, and D. J. Cooper [6], which was used by Ripple et al. (2025) in their analysis

Lion geyser and Heart Spring in Yellowstone National Park. From Wikimedia Commons (2008), by Brocken Inaglory.

Read in data

The data from Hobbs et al. (2024) comes as a csv file - which means `read_xlsx` will not work this time, so we need to install the `readr` package instead. Remember to do this in your console, not your coding script.

Once you have installed `readr`, we can summon it from the R library.

```
CODE
library(readr) # activates the readr package
```

> **Tip**
>
> Just like `ggplot2`,`readr` is also included in the `tidyverse` package.

Now, we can read in our data using the function `read.csv()`:

```
CODE
Yellowstone ← read.csv('data/browsing_data_2003_2020_2.csv') # we will name this dataset
'Yellowstone'
```

> **Building Habits**
>
> Check the structure of the `Yellowstone` dataset.

Food for thought

This is what we did:

```
CODE
str(Yellowstone)
```

```
OUTPUT
'data.frame':   3840 obs. of  21 variables:
 $ site_full     : chr  "crescent-obs" "crescent-obs" "crescent-obs" "crescent-obs" ...
 $ willid        : int  86 86 86 86 86 86 86 86 89 ...
 $ year          : int  2010 2014 2013 2011 2009 2017 2012 2018 2015 2013 ...
 $ willid_full   : chr  "crescent-obs-86" "crescent-obs-86" "crescent-obs-86" "crescent-
obs-86" ...
 $ site_id       : chr  "crescent" "crescent" "crescent" "crescent" ...
 $ treat         : chr  "obs" "obs" "obs" "obs" ...
 $ exp           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ plantht       : int  49 65 66 45 42 56 81 57 66 77 ...
 $ N_shoots      : int  95 84 134 170 54 75 78 27 124 69 ...
 $ N_browsed     : int  60 3 74 43 29 14 22 8 13 31 ...
 $ N_unbrowsed   : int  33 76 55 120 25 54 56 19 107 32 ...
 $ N_deep_browsed: int  2 5 5 7 0 7 0 0 4 6 ...
 $ p_browsed     : num  0.6316 0.0357 0.5522 0.2529 0.537 ...
 $ p_deep_browsed: num  0.0211 0.0595 0.0373 0.0412 0 ...
 $ fence         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ dam           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ browse        : int  1 1 1 1 1 1 1 1 1 1 ...
 $ n.plants      : int  9 9 9 9 9 9 9 9 9 8 ...
 $ n.years       : int  10 10 10 10 10 10 10 10 10 9 ...
 $ min_yr        : int  2009 2009 2009 2009 2009 2009 2009 2009 2009 2009 ...
 $ max_yr        : int  2018 2018 2018 2018 2018 2018 2018 2018 2018 2017 ...
```

Notice that words come up as `chr`, which stands for 'character'. Characters cannot be analysed statistically - they must first be converted into either numbers or factors.

In this case, we are not interested in running statistics; so we are fine to leave the characters as they are.

This dataset is the original one produced by Hobbs and his research team in 2024 from 21 control sites and 16 experimental sites.

However, when Ripple's team re-analysed the same dataset one year later, they did not include all 37 sites. Instead, for unknown reasons, they only chose 4 out of 16 experimental sites to study.

One of the major criticisms leveled at Ripple by MacNulty et al. (2025) was that such an odd choice of study sites jeopardised the validity of the rest of the study.

To see why MacNulty thought this, let's try to replicate Ripple's study design with our own dataset.

First, we have to remove all the sites in our dataset that Ripple excluded from his study. The code for this is a little bit tricky, so we will go through it step-by-step.

Here is a list of all the sites that Ripple excluded:

i Sites that Ripple excluded

```
CODE
sites_excluded ← c(
  'wb-dx','wb-dc','wb-cx',
  'elk-dx','elk-dc','elk-cx',
  'eb2-dx','eb2-dc','eb2-cx',
  'eb1-dx','eb1-dc','eb1-cx'
) # Notice that this is a vector. We are used to seeing vectors with numbers by now, but we
are also allowed to make vectors with characters.
```

The challenge is to remove all of these sites from our dataset.

We can turn to the `[,]` function for this. Remember that to remove rows from our dataset, we need to specify them in front of the comma `[*,]`.

i Sharpen your skills

i) Select the first row of the `Yellowstone` dataset.

ii) Select the first five rows of the `Yellowstone` dataset.

Solution

For part i), apply the `[,]` function directly:

```
CODE
Yellowstone[1,] # selects the first row in the dataset
```

```
OUTPUT
```

```
     site_full willid year     willid_full  site_id treat exp plantht N_shoots
1 crescent-obs     86 2010 crescent-obs-86 crescent   obs   0      49       95
  N_browsed N_unbrowsed N_deep_browsed p_browsed p_deep_browsed fence dam
1        60          33              2 0.6315789     0.02105263     0   0
  browse n.plants n.years min_yr max_yr
1      1        9      10   2009   2018
```

Part ii) is a bit more difficult. We need to use the : operator to select rows 1 to 5 before applying `[,]`:

CODE

```
Yellowstone[1:5,] # selects rows 1 to 5 in the dataset
```

OUTPUT

```
     site_full willid year     willid_full  site_id treat exp plantht N_shoots
1 crescent-obs     86 2010 crescent-obs-86 crescent   obs   0      49       95
2 crescent-obs     86 2014 crescent-obs-86 crescent   obs   0      65       84
3 crescent-obs     86 2013 crescent-obs-86 crescent   obs   0      66      134
4 crescent-obs     86 2011 crescent-obs-86 crescent   obs   0      45      170
5 crescent-obs     86 2009 crescent-obs-86 crescent   obs   0      42       54
  N_browsed N_unbrowsed N_deep_browsed  p_browsed p_deep_browsed fence dam
1        60          33              2 0.63157895     0.02105263     0   0
2         3          76              5 0.03571429     0.05952381     0   0
3        74          55              5 0.55223881     0.03731343     0   0
4        43         120              7 0.25294118     0.04117647     0   0
5        29          25              0 0.53703704     0.00000000     0   0
  browse n.plants n.years min_yr max_yr
1      1        9      10   2009   2018
2      1        9      10   2009   2018
3      1        9      10   2009   2018
4      1        9      10   2009   2018
5      1        9      10   2009   2018
```

Tip

To select rows by name, first use $ to specify the column that lists all the site names, then use == to match a specific name. For example, to select all the rows from the site "crescent-obs":

CODE

```
head(Yellowstone[Yellowstone$site_full =="crescent-obs",])
```

OUTPUT

```
     site_full willid year     willid_full  site_id treat exp plantht N_shoots
1 crescent-obs     86 2010 crescent-obs-86 crescent   obs   0      49       95
2 crescent-obs     86 2014 crescent-obs-86 crescent   obs   0      65       84
3 crescent-obs     86 2013 crescent-obs-86 crescent   obs   0      66      134
4 crescent-obs     86 2011 crescent-obs-86 crescent   obs   0      45      170
5 crescent-obs     86 2009 crescent-obs-86 crescent   obs   0      42       54
6 crescent-obs     86 2017 crescent-obs-86 crescent   obs   0      56       75
  N_browsed N_unbrowsed N_deep_browsed  p_browsed p_deep_browsed fence dam
1        60          33              2 0.63157895     0.02105263     0   0
2         3          76              5 0.03571429     0.05952381     0   0
3        74          55              5 0.55223881     0.03731343     0   0
4        43         120              7 0.25294118     0.04117647     0   0
5        29          25              0 0.53703704     0.00000000     0   0
6        14          54              7 0.18666667     0.09333333     0   0
  browse n.plants n.years min_yr max_yr
1      1        9      10   2009   2018
2      1        9      10   2009   2018
3      1        9      10   2009   2018
4      1        9      10   2009   2018
5      1        9      10   2009   2018
6      1        9      10   2009   2018
```

We first have to specify `Yellowstone$site_full`, because `site_full` is the column that lists all the site names. Then, we use == to match the name `crescent-obs`.

The `head` argument at the beginning is just to prevent R from printing a long table. You can omit it if you want to see the full list of results.

Now, we do a little bit of coding magic and invoke the `%in%` function to pick out multiple row names at once.

Selecting all excluded sites

CODE

```
head(Yellowstone[Yellowstone$site_full %in% sites_excluded,])
```

OUTPUT

```
    site_full willid year willid_full site_id treat exp plantht N_shoots
191    eb1-cx    637 2011  eb1-cx-637     eb1    cx   1     146      139
192    eb1-cx    637 2013  eb1-cx-637     eb1    cx   1     162      166
193    eb1-cx    637 2017  eb1-cx-637     eb1    cx   1     167       63
194    eb1-cx    637 2010  eb1-cx-637     eb1    cx   1     165       84
195    eb1-cx    637 2012  eb1-cx-637     eb1    cx   1     120      238
196    eb1-cx    637 2018  eb1-cx-637     eb1    cx   1     195       41
    N_browsed N_unbrowsed N_deep_browsed p_browsed p_deep_browsed fence dam
191         0         139              0         0              0     1   0
192         0         166              0         0              0     1   0
193         0          63              0         0              0     1   0
194         0          84              0         0              0     1   0
195         0         238              0         0              0     1   0
196         0          41              0         0              0     1   0
    browse n.plants n.years min_yr max_yr
191      0       12      16   2003   2018
192      0       12      16   2003   2018
193      0       12      16   2003   2018
194      0       12      16   2003   2018
195      0       12      16   2003   2018
196      0       12      16   2003   2018
```

The `%in% sites_excluded` part picks out every site whose name matches the `sites_excluded` vector we made earlier.

Again, `head` is just to limit the number of rows R displays.

Phew! That's the hard part done. All that is left is to use the `!` operator to tell R that we want to *exclude* these sites, not include them, and then give our new dataset a name.

CODE

```
Yellowstone_1 ← Yellowstone[!Yellowstone$site_full %in% sites_excluded,] # remove rows and rename
as 'Yellowstone_1'
```

Ripple claims that his study occurred across 25 sites from 2001 to 2020. Let's make a line graph to see how often each of these 25 sites were actually surveyed:

Sharpen your skills

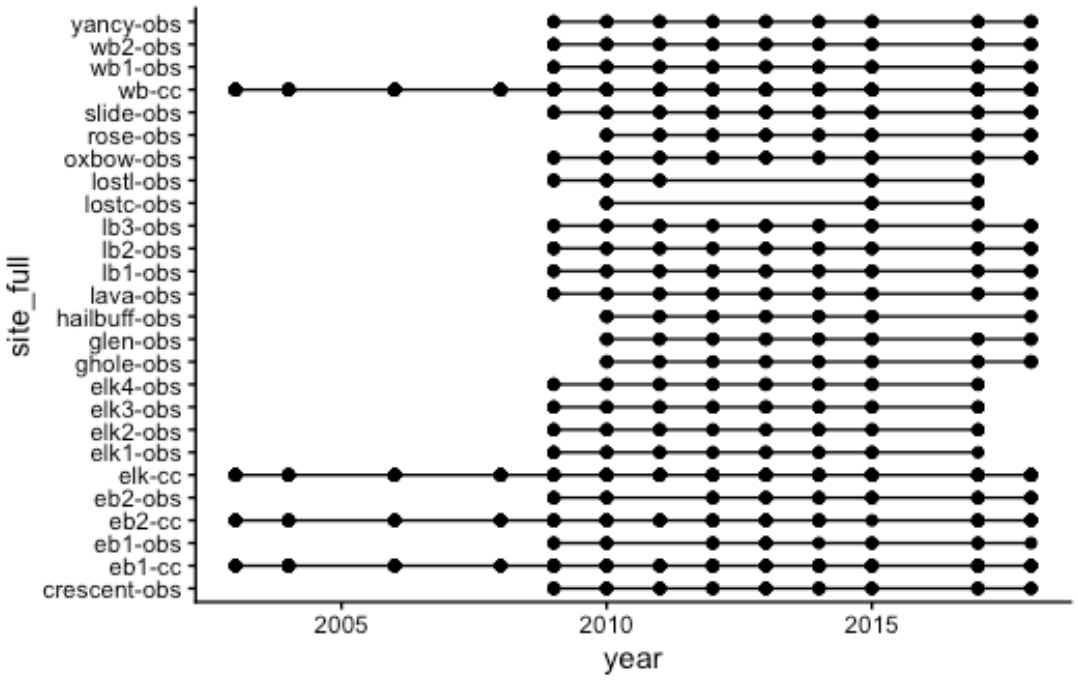
Use `geom_line()` to make a line graph with `year` on the x-axis, and `sites_full` on the y-axis.

What do you notice about the times each of these sites were surveyed?

Solution

Here's what we did:

```
CODE
ggplot(Yellowstone_1, aes(x = year, y = site_full))+
  geom_line()+
  geom_point()+
  theme_classic() # Note that we added a scatter plot using geom_point() to see the timing of
the surveys even more clearly. Each point is one survey.
```



While it is true that Ripple used survey data from 25 sites, only 4 of these sites had records tracing back to 2001. This means that most sites in Ripple's study did not have a reliable baseline to compare against. For a supposed 'before-after' study, Ripple was missing a lot of 'before' sites.

Ripple describes how willows in 2020 were, on average, twice as tall as they were in 2001; but the willows from 2020 were not the same as the ones from 2001, because new sites were added in between! While Ripple's study spanned 20 years in principle, the bulk of his evidence really only spanned 13 years in practice (from 2008 to 2020).

Conclusion

Closing Thoughts

Is this all to say that wolves had no effect on the ecology of Yellowstone National Park? No. In fact, MacNulty himself believes that wolves did in fact cause a trophic cascade, but a much weaker one than what Ripple proposed.

Is it fair to say that Ripple was a charlatan? Certainly not. Dr William Ripple is a distinguished professor at Oregon State University, and has published many groundbreaking papers on complex ecological processes, including trophic cascades.

What this case study really shows is that even experienced researchers working on high-profile experiments can make mistakes. That's got to take some pressure off the rest of us, right?

The most important thing is to listen to the criticisms of others without taking it too personally. That way, we can help each other avoid costly pitfalls through collaboration.

> **i** For you to consider
>
> Let's revisit the debate between MacNulty and Ripple with a better grasp of the situation. What are your thoughts now? Was MacNulty fair in his criticism of Ripple's study?

What we think

While we disagree with MacNulty's claim that Ripple's study was 'invalid', we do agree that Ripple's survey methods could have used some improvements. In particular, we think each survey site should have been fixed through time, so that 'before' and 'after' measurements came from the same set of trees (i.e. a paired study design).

Otherwise, as MacNulty points out, it is difficult to say whether wolves had a positive impact on tree growth, or whether some trees were simply taller than others to begin with.

We will learn more about paired study designs next week.

Thanks!

That's all for today. If you have any questions, please approach your demonstrators. Don't forget to save your Quarto document for future reference.

Bibliography

[1] W. J. Ripple, R. L. Beschta, C. Wolf, L. E. Painter, and A. J. Wirsing, "The Strength of the Yellowstone Trophic Cascade after Wolf Reintroduction," *Global Ecology and Conservation*, vol. 58, 2025, doi: 10.1016/j.gecco.2025.e03428.

[2] D. R. MacNulty, D. Cooper, M. Procko, and T. J. Clark-Wolf, "Flawed Analysis Invalidates Claim of a Strong Yellowstone Trophic Cascade after Wolf Reintroduction: A Comment on Ripple et al. (2025)," *Global Ecology and Conservation*, vol. 63, 2025, doi: 10.1016/j.gecco.2025.e03899.

[3] G. Thomson and C. Thompson, "Movement and Size Structure in a Population of the Blue Starfish \textit{Linckia laevigata} (L.) at Lizard Island, Great Barrier Reef," *Marine and Freshwater Research*, vol. 33, no. 3, p. 561, 1982, doi: 10.1071/mf9820561.

[4] G. M. Lovett, K. C. Weathers, and W. V. Sobczak, "Nitrogen Saturation and Retention in Forested Watersheds of the Catskill Mountains, New York," *Ecological Applications*, vol. 10, pp. 73–84, 2000.

[5] J. Karjalainen *et al.*, "Sulfate Sensitivity of Aquatic Organisms in Soft Freshwaters Explored by Toxicity Tests and Species Sensitivity Distribution," *Ecotoxicology and Environmental Safety*, vol. 258, p. 114984, 2023, doi: 10.1016/j.ecoenv.2023.114984.

[6] N. T. Hobbs, D. B. Johnston, K. N. Marshall, E. C. Wolf, and D. J. Cooper, "Does Restoring Apex Predators to Food Webs Restore Ecosystems? Large Carnivores in Yellowstone as a Model System," *Ecological Monographs*, vol. 94, no. 2, 2024, doi: 10.1002/ecm.1598.