Welcome to our workshop!

We are so glad you are here. Please call over a helper or instructor if you have any questions or concerns before we begin the workshop. Also, confirm that you have all the necessary set-up completed.

Survey Link: https://carpentries.typeform.com/to/wi32rS?slug=2025-01-14-Cornell-Micro

POST WORKSHOP SURVEY LINK :https://carpentries.typeform.com/to/UgVdRQ?slug=2025-01-14-Cornell-Micro&typeform-source=marschmilab.github.io

Links to your reports and repos:

https://github.com/rharding17/ontario-report/tree/main/Reports

Lesson #4: RMarkdown

Link to data analysis lesson: https://marschmilab.github.io/Cornell_Carpentries_Jan2025/05-r-markdown/index.html

ACTIVITY: How do you combine analyses, plots, citations, and writing?
Google docs, adobe illustrator for graphs, but screenshot and paste into docs):
Microsoft Word w/ track changes, Zotero plugin for citations, export figures/statistics from R Studio, Upload to Box for PI
make plots is Graphpad Prism, use power point to make figures with multiple panels and word doc for main manuscript, zotero for citations included in main manuscript doc and then email all files to collaborator with track changes turned on(same)
Ive only used excel for graphs and docs to combine analyses with graphs.
in the past, I have used docx with tracking edits , zotero for citations, Python for graphs
Microsoft word -> tracking changes or comments
box and zotero
Digital Lab notebook. shared documents on Box. email drafts/data/figures back and forth. Zotero for citations
Microsoft Word + Zotero （+1）
Microsoft Word/Zotero/Box
Teams folders full of Word docs and images made in R and then touched up in Illustrator or made in biorender
Microsoft word/google docs and Zotero. Putting things on Box.
Make graphs in Excel or Tableau, paste image or screenshot into Word with rest of report. Sometimes edit graphs with Paint to highlight certain areas.
Tables: Excel, texts: Word, Figures:PowerPoint, Statistics: Rstudio separatly

---

Before we begin please open up terminal (mac)/GitBash (windows) and navigate to the ontario-report directory! We will then back up to Git. To do this, type:

- pwd
- cd Desktop/ontario-report
- git status
- git add code/data_analysis.R
- git add data/sample_and_taxon.csv
- git status
    - we have now staged these two new files!
- git commit -m "Joined sample and taxon data"
- git status
- git push origin main
    - Error! We did not pull the most recent version!
- git pull origin main
    - ctrl+X to get out of the pop up pane
- git push origin main

Creating an R Markdown file:

- Navigate back to RStudio
- Go to "Session" and click "Restart R"
    - Environment should be empty, no plots should be visible.
    - Ensure you are still in ontario-report!
- Create a new directory within ontario-report called "reports"
    - This is where we will be saving our markdown documents
- Go to File --> New File --> R Markdown
    - Title document something like: "A report on Lake Ontario's Microbes"
    - Keep default format as HTML
    - Note: R Markdown files always end in ".Rmd"

Basic components of R Markdown:

- The first part is a "header" at the top of the file between the lines of ---
- The next section is a "code chunk" aka embedded R code that sets up options for the code chunks
    - denoted by 3 backticks "```" and a curly bracket "{}"
    - we end coding chunk with another 3 backticks "```"
- There is also text - this is written prose, just like the text we would find in a google doc or word document

Starting the report:

- highlight everything from line 12 down to the bottom and hit delete
    - note: the first code chunk is the "setup code chunk"
- we will save this file by hitting "floppy disc"
    - can title the Markdown document somehting like "ontario-report"
    - should now be able to see this document in the reports directory
    - remember: R Markdown files always end in ".Rmd"
- With the "Knit" button

    - navigate to "Knit Directory" and click "Project Directory"
- click the knit button
    - we have now made a new document with a ".html" ending (also visible in the "reports" directory). So cool!
- Now in your Markdown document, type:
    - ```{r packages}
    - library(tidyverse)
    - ```
        - Now click "knit"
        - Here we added in R code! This code loads tidyverse!
        - Note: if you click gear symbol and change to "Chunk output in console"
            - this sends the output into the console instead of being stuck down into the R Markdown document
- To read in our data, type:
    - ```{r data}
    - sample_and_taxon <- read_csv("data/sample_and_taxon.csv")
    - ```
- To add in a cells vs temp plot, type:
    - ```{cells_vs_temp}
    - sample_and_taxon %>%
    - ggplot() +
    - aes(x = temperature, y = cells_per_ml/1000000, color=env_group) +
    - geom_point() +
    - labs(x = "Temperature (C)", y = "Cells(million/ml)",
    - title= "Are temperature and cell abundance linked?")
    - ```
        - Now click "knit"
        - In your knitted document you should be able to see your plot!

- Add in some sort of text to your report to explain what is happening in each chunk, to help understand the code/plots in the document. You can add the text in the white space below the code chunks. Example, you could type in the following:
    - The above plot shows the relationship between cell abundance and temperature for a total of 'r nSamples' samples.
- To get rid of features we don't need, we need to make some changes in the Markdown document. We will include different arguments in the curly brackets: Play around with the following: "warning", "message", "echo", "include" and "eval". For example, in the curly brackets try adding "include=FALSE" like so:
    - ```{r packages, include=FALSE}
    - ```{r cell_vs_temp, echo = FALSE}
    - ```{r data_summary, include=FALSE}
        - Note: echo hides the code if set to "echo=FALSE"
            - "include=FALSE" we don't see code or output (we ran it, just silently)
            - "warning=FALSE" we no longer see warning sections
            - "message=FALSE" we no longer see messages (for example like the one we saw when we loaded tidyverse)
            - "eval=FALSE" we still include the code, but won't run the code in

the chunk

- Adjust by typing:
    - ```{r packages, include=FALSE}
    - library(tidyverse)
    - ```
- We want to calculate three summary statistics. Below our plot, add a new R chunk by typing:
    - ```{r data_summary, include=FALSE}
    - nSamples <- nrow(sample_and_taxon)
    - minTemp <- sample_and_taxon %>%
    - pull(temperature) %>%
        - min() %>%
        - round()
    - maxTemp <- sample_and_taxon %>%
        - pull(temperature) %>%
        - max() %>%
        - round()
    - ```
    - pull = pulls/selects a column from a dataframe and transforms it into a vector!
    - round = rounds a decimal
- Now writing an "in-line r chunk". Knit when finished. In the white below, type:
    - The above plot shows `r nSamples`. For these samples, the minimum temp was `r minTemp `°C and the maximum was `r maxtemp`°C.
- Use "headers" to break up your document:
    - outside of an R chunk the pound sign "#" indicates a header
        - can nest headers with multiple pound signs! ex: "##", the more pound signs the smaller your header is going to be
            - 6 #'s is the maximum for a header
        - feel free to play around with adding headers within your R Markdown document
        - Knit this document to visualize the headers you have set up. :)
        - All of the labelling with headers also helps us organize and navigate our document
            - click the "outline" button to see an outline of all the headers used
            - also can click the white space to see a menu of your document organized by the headers. Click on the headers to navigate to different parts of your document.
- Note about the "Visual" tab. This tab allows us to insert images, code, citations and adjust our formatting, similar to how we would do it in a Google/Word document.
- To insert tables, your best bet is to go to visual tab and hit "Table". To practice, insert a table with parameters of your choosing.
    - Gus used two columns: "Statistic" and "Value" and three rows: "Number of Samples", "Minimum Temp" and "Maximum Temp".
        - In the value column he inserted `r nSamples`, `r minTemp` and `r maxTemp` for each coordinating row.
    - Navigate back to the "Source" tab to view the markdown syntax RStudio has automatically created for you.
- Now, try to use kable. Type:

- ```{r kable}
- library(knitr)
- sample_and_taxon %>%
- filter(env_group == "Deep") %>%
- select(cells_per_ml, temperature, Chloroflexi) %>%
- kable()
    - Note: Kable = type of table. We can give the kable() function a data table and it will format it to a nice looking table in the report.
    - Knit your document
    - Your Kable will likely look a tad nicer than the default table we previously made.

Challenge

- Make a bullet point list. First line is your favourite coding language we've learned so far (in italics). The second line is your favourite function we've learned so far (in bold). The third line is the next programming thing you want to learn (in italics and bold).
    - If complete, switch to a numbered list. You can also find a markdown formatting tutorial and insert it as a hyperlink.
- To complete the challenges, type:
    - -*Markdown*
    - - facet_wrap**
    - - ***SQL***
        - Now knit to confirm we have done it correctly!
        - To change this bullet list to numbered list, hold down option (mac)/alt (windows) and drag cursor, can do backspace and add numbers with a period following.
            - ex: 1. *Markdown*
- To make a hyperlink, type:
    - [Markdown Tutorial](INSERT LINK)
        - Don't forget to knit!
        - This will insert the link, which will show up as "Markdown Tutorial" in our document.

Citation Demo

- Tools --> global options --> R Markdown (add your local)
    - hit "@" to search within your Zotero library
    - once you knit you will have a bibliography automatically produced and you will also have your inline citations.
    - can change format of citations by making specifications in your YAML header

General Notes:

- everytime we knit, we save our Markdown document
- can add/commit/push with the "Git" button
    - click "Git"
    - check boxes of files you would like to add
    - write a commit message in the white box
    - hit the commit button to commit

        - hit "push" to push to GitHub
            - note:
                - the 'm' means modified
                - the two yellow question marks represent untracked files, files that have not been added
                - the 'd' means deleted
- can insert images multiple ways
            - Often the simplest way:
                - copy the image
                - navigate to visual pane
                - you can paste image directly into Markdown document
- If you ran across issues pushing/pulling from the Git tab in Rstudio, in your terminal (mac)/ GitBash (windows) type:
    - git push -u origin main
        - Note that if you are a mac user, you are able to do this from your terminal within RStudio. If you are a windows user, you must do this through GitBash.

Make your own Markdown document!

- 1. Make a new Markdown file
- 2. Give it an informative title
- 3. Delete all unnecessary Markdown and R code (everything below the setup R chunk)
- 4. Save it to the "reports" directory as LASTNAME_Ontario_Report.Rmd

Lesson #3: Dataframe Management with dplyr and tidyr (R for Data Management)

Link to data analysis lesson: https://marschmilab.github.io/Cornell_Carpentries_Jan2025/04-r-data-analysis/index.html

Open up yesterday's R project (.Rproj) from your ontario-report folder. Check that you see "ontario-report" in your top right corner to maek sure that you're in the right R project.

- create new Rscript for today (FIle --> New File --> Rscript). Call this script data_analysis.R and save it to your code file in your ontario-report directory.
- #Loading packages
    - library(tidyverse)
- #Reading in data
    - sample_data <- read_csv("data/sample_data.csv")
    - summarize(sample_data, average_cells = mean(cells_per_ml))
    - sample_data %>% summarize(average_cells = mean(cells_per_ml))
    - sample_data %>%
        - summarize(average_cells = mean(cells_per_ml))
- #Filtering rows
    - sample_data %>%
        - filter(env_group == "Deep")

    - sample_data %>%
        - filter(env_group == "Deep") %>%
        - summarize(average_cells = mean(cells_per_ml))
- quick notes on logical operators
    - == is used for naming
    - == means "is equal to"
    - != means "'is not equal to"
    - < and > mean is less than and is greater than, respectively
    - %in% allows you to select multiple types of rows when filtering
- what using the string package in R would look like-- for your information!
    - sample_data %>%
        - filter(str_detect(env_group, "Shallow"))
- #calculate the average chlorophyll in the entire dataset
    - sample_data %>%
        - summarize(avg_chl = mean(chlorophyll))
- #calculate the average chlorophyll in only Shallow September
    - sample_data %>%
        - filter(env_group == "Shallow September") %>%
        - summarize(avg_chl = mean(chlorophyll))
- #looking for just the samples in September, not jsut the Shallow September ones
    - sample_data %>%
        - filter(str_detect(env_group, "September")) %>%
        - summarize(avg_chl = mean(chlorophyll))
- #learning about the group_by function
    - sample_data %>%
        - group_by(env_group)
    - sample_data %>%
        - group_by(env_group) %>%
        - summarize(average_cells = mean(cells_per_ml),
            - min_cells = min(cells_per_ml))
- #calculate the average temperature per env_group
    - sample_cata %>%
        - group_by(env_group) %>%
        - summarize(average_temperature = mean(temperature))
- #Learning about the mutate function-- let's calculate the TN:TP for all of our data
    - sample_data %>%
        - mutate(tn_tp_ratio = total_nitrogen / total_phosphorus)
    - sample_data %>%
        - mutate(tn_tp_ratio = total_nitrogen / total_phosphorus) %>% view #this is if you want to pipe your results into a temporary View tab to see all your data

- #what if we want to group by multiple columns?
    - sample_data %>%
        - mutate(temp_is_hot = temperature > 8)
    - sample_data %>%
        - mutate(temp_is_hot = temperature > 8) %>% view #this will output a logical column that tells you if the temperature is greater than 8 degrees, by using TRUE or FALSE
    - sample_data %>%
        - mutate(temp_is_hot = temperature > 8) %>%
        - group_by(env_group, temp_is_hot)
    - sample_data %>%
        - mutate(temp_is_hot = temperature > 8) %>%
        - group_by(env_group, temp_is_hot) %>%
        - summarize(avg_temp = mean(temperature))
    - sample_data %>%
        - mutate(temp_is_hot = temperature > 8) %>%
        - group_by(env_group, temp_is_hot) %>%
        - summarize(avg_temp = mean(temperature),
            - avg_cells = mean(cells_per_ml))
- #selecting columns with select function
    - sample_data %>%
        - select(sample_id, depth)
- #we can use the select function to get rid of columns we don't want!
    - sample_data %>%
        - select(-env_group)
- #learning to use the colon to select ranges in your dataset
    - sample_data %>%
        - select(sample_id:temperature)
- #learning about some select helper functions!
    - sample_data %>%
        - select(starts_with("total"))
- #create a dataframe with only sample_id, env_group, depth, temperature and cells_per_ml (there are lots of ways of doing this! check out a few of them below)
    - sample_data %>%
        - select(sample_id: temperature)
    - sample_data %>%
        - select(sample_id, env_group, depth, temperature, cells_per_ml) # this will produce a dataframe with the columns in the same order that they've been inputted into the select function, meaning that you can use this function to reorder your columns!

- sample_data %>%
    - select(1:5)
- sample_data %>%
    - select(-starts_with("total"), -chlorophyll, -diss_org_carbon)
- sample_data %>%
    - select(-(total_nitrogen:chlorophyll))

SAVE YOUR SCRIPT!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

- #cleaning data
    - read_csv("data/taxon_abundance.csv")
    - read_csv("data/taxon_abundance.csv", skip = 2)
    - read_csv("data/taxon_abundance.csv", skip = 2) %>%
        - select(-...10)
    - read_csv("data/taxon_abundance.csv", skip = 2) %>%
        - select(-...10) %>%
        - rename(sequencer = ...9)
    - read_csv("data/taxon_abundance.csv", skip = 2) %>%
        - select(-...10) %>%
        - rename(sequencer = ...9) #remember that you can pipe this into a view command by adding %>% view() if you feel like you're not getting enough information from the tibble outputted to your console
- #remove the lot number and sequencer columns and assign all of the cleaned data to an object called "taxon_clean"
    - read_csv("data/taxon_abundance.csv", skip = 2) %>%
        - select(-...10) %>%
        - rename(sequencer = ...9) %>%
        - select(-Lot_Number, -sequencer)
    - taxon_clean <- read_csv("data/taxon_abundance.csv", skip = 2) %>%
        - select(-...10) %>%
        - rename(sequencer = ...9) %>%
        - select(-Lot_Number, -sequencer)
- #learning about long data vs wide data
    - taxon_clean %>%
        - pivot_longer(cols = Proteobacteria:Cyanobacteria,
            - names_to = "Phylum",
            - values_to = "Abundance")
    - taxon_long <- taxon_clean %>%
        - pivot_longer(cols = Proteobacteria:Cyanobacteria,
            - names_to = "Phylum",
            - values_to = "Abundance")
    - taxon_long %>%

        - group_by(Phylum) %>%
        - summarize(avg_abund = mean(Abundance))

- #let's check out what these data look like by making a stacked bar plot!
    - taxon_long %>%
        - ggplot() +
        - aes(x = sample_id,
            - y = Abundance,
            - fill = Phylum) +
        - geom_col() +
        - theme(axis.text.x = element_text(angle = 90))

- #Making long data wide
    - taxon_logn %>%
        - pivot_wider(names_from = "Phylum",
            - values_from = "Abundance")

- #Joining dataframes
    - head(sample_data)
    - head(taxon_clean)
- **Interested in learning more about join functions? Check out this useful link: https://dplyr.tidyverse.org/reference/mutate-joins.html**
- #Inner join
    - inner_join (sample_data, taxon_clean)
    - inner_join (sample_data, taxon_clean, by = "sample_id") #it's better to be more specific about what column you want to join with
    - anti_join(sample_data, taxon_clean, by = "sample_id") #this will allow us to determine which rows we lost when we ran the inner_join function in the previous step!
    - #if we want to look at the values from just one column, we can use the $ symbol, from the base R functions
        - sample_data$sample_id
        - taxon_clean$sample_id
    - #we need to fix the September taxon IDs so that they match!
        - taxon_clean %>%
            - mutate(sample_id = str_replace(sample_id, pattern = "Sep", repalcement = "September"))
    - taxon_clean_goodSep <- taxon_clean %>%
            - mutate(sample_id = str_replace(sample_id, pattern = "Sep", repalcement = "September"))
    - inner_join (sample_data, taxon_clean_goodSep, by = "sample_id") #now we have the right number of rows! yay!

SAVE YOUR SCRIPT!!!!!!!!!!!!!!!!!!!!!!!!

- #let's save our work to a .csv file!
    - sample_and_taxon <- inner_join (sample_data, taxon_clean_goodSep, by = "sample_id")
    - write_csv(sample_and_taxon, file = "data/sample_and_taxon.csv")
- #let's make a plot!!!!!!!!!!!!!!!!!!!! :)
- #first question we're asking: where does Chloroflexi like to live?

- sample_and_taxon %>%
    - ggplot() +
    - aes(x = depth,
        - y = Chloroflexi) +
    - geom_point()
- sample_and_taxon %>%
    - ggplot() +
    - aes(x = depth,
        - y = Chloroflexi) +
    - geom_point() +
    - labs(x = "Depth(m)",
        - y = "Chloroflexi relative abundance")
- sample_and_taxon %>%
    - ggplot() +
    - aes(x = depth,
        - y = Chloroflexi) +
    - geom_point() +
    - labs(x = "Depth(m)",
        - y = "Chloroflexi relative abundance") +
    - geom_smooth()
- sample_and_taxon %>%
    - ggplot() +
    - aes(x = depth,
        - y = Chloroflexi) +
    - geom_point() +
    - labs(x = "Depth(m)",
        - y = "Chloroflexi relative abundance") +
    - geom_smooth(method = "lm")
- #let's make this even prettier by adding our equation of the line!
    - install.packages("ggpubr")
    - library(ggpubr)
        - sample_and_taxon %>%
            - ggplot() +
            - aes(x = depth,
                - y = Chloroflexi) +
            - geom_point() +
            - labs(x = "Depth(m)",
                - y = "Chloroflexi relative abundance") +
            - geom_smooth(method = "lm") +
            - stat_regline_equation()
        - sample_and_taxon %>%
            - ggplot() +
            - aes(x = depth,
                - y = Chloroflexi) +

- geom_point() +
- labs(x = "Depth(m)",
    - y = "Chloroflexi relative abundance") +
- geom_smooth(method = "lm") +
- stat_cor() #runnning a correlation

- sample_and_taxon %>%
    - ggplot() +
    - aes(x = depth,
        - y = Chloroflexi) +
    - geom_point() +
    - labs(x = "Depth(m)",
        - y = "Chloroflexi relative abundance") +
    - geom_smooth(method = "lm") +
    - stat_cor() +
    - annotate(geom = "text",
        - x = 25, y = .3,
        - label = "This is a text label")

- #what is the average abundance and standard deviation of Chloroflexi in our three env_groups
    - sample_and_taxon %>%
        - group_by(env_group) %>%
        - summarize(avg_chloro = mean(Chloroflexi),
            - sd_chloro = sd(Chloroflexi))

SAVE YOUR SCRIPT!!

Lesson #2: Unix and Git
Link to Unix lesson: https://marschmilab.github.io/Cornell_Carpentries_Jan2025/02-unix-shell/index.html
Link to Git lesson: https://marschmilab.github.io/Cornell_Carpentries_Jan2025/03-intro-git-github/index.html
Link to BioHPC if you want to learn more: https://biohpc.cornell.edu/Default.aspx

Introducing the Shell

- Unix is a type of operating system for computers
    - Mac = Terminal
    - Windows = GitBash
- The first line of the shell shows a prompt... the shell is waiting for an input!
- If you see "command not found" look for typos!!

Navigating the Shell

- To get the "$" prompt, type:
    - `PS1='$ '`
    - Directory: synonymous with a folder in a computer
    - Note: directories can be nested within other directories. We call the way that different directories are nested the "directory tree".
- To see where you are located, type:
    - pwd
- To see what files are in the directory, type:
    - ls

Man and Help

- Windows command: ls --help
- Mac command: man ls
    - use "q" to exit the manual
- Note: man is short for manual!
- Flags, similar to arguments in R, let us customize how "ls" runs
- Testing different flags:
    - man ls
    - ls -F
    - ls -a
    - ls -F -a
        - Can also use "ls -Fa" as a more compact version of this command
- Challenge: find the ls option that prints everything in order of size
    - ls -S
- Note: beware of capital/lowercase letters, case sensitivity is important here!

Navigating to different/specific directories

- To move into a different directory (with goal of moving to the ontario-report directory), type:
    - cd Desktop
    - ls
    - cd ontario-report
- If you have a problem with having your "Desktop" folder within OneDrive, you might need to type something similar to one of the following instead:
    - /c/Users/USERNAME/OneDrive - Cornell University/Desktop/ontario-report
    - ~/OneDrive/Desktop/ontario-report
- To check to see we have all of our files in ontario-report, type:
    - ls
- To move out of our ontario-report directory to the prior directory, type:
    - cd ..
    - Note: this takes us back to "Desktop"
- To navigate back to ontario-report, type:
    - cd ontario-report
    - pwd
    - ls
- You can also use the absolute path, for example you could type:
    - /Users/augustuspendleton/Desktop/ontario-report
- Note: typing "cd ." doe not move you to a new/previous directory

Making a directory and moving files

- To make a directory named "code", type:
    - mkdir code
    - ls
- To move files into our new directory, type:
    - mv plotting.R code
        - mv means move
        - plotting.R is the file we want to move
        - code is the directory we want to move our file to
- To check you moved the file into the desired directory, type:
    - ls code
        - This shows us all of the files in the "code" directory

- To make a directory called "data", type:
    - mkdir data
- To move all .csv files to your "data" directory, type:
    - mv buoy_data.csv data
    - mv sample_data.csv data
    - mv taxon_abundance.csv data

- Wildcard: used to match files and directories to patterns
    - For example, when you type:
        - mv *.csv data
            - all of the files ending in *.csv will be moved to the "data" directory
            - the asterisk essentially replaces everything before the .csv

- To make a directory called "figures", type:
    - mkdir figures
- To move our figures into our new "figures" directory, type:
    - mv *.jpg figures

Viewing Files

- To navigate to our data folder, type:
    - cd data
        - Note: CSV is a type of data-containing file
- To look at sample_data.csv, type:
    - less sample_data.csv
- To look at the rest of the data:
    - use arrow keys to navigate up, down and side-to-side
    - Note: column headers show up on the first line (ex. "sample_id")
    - Use "q" to quit
- Challenge: find other commands that visualize files in Unix command line
    - Examples from the group:
        - head

        - tail
        - cat (stands for concatenate)
        - <
        - echo
    - Using examples:
        - head sample_data.csv
            - gives you first few lines
        - tail sample_data.csv
            - gives you the end

- Can we visualize a jpeg? Let's take a look!
    - cd ..
    - ls
    - cd figures
    - ls
    - cat awesome_box_plot.jpeg
        - Ouput: nonsense! This is an image, command line interfaces cannot visualize this.
    - less awesome_box_plot.jpeg
        - Output: confusion again! Cannot really do anything with images in a command line interface.

How to edit files

- To navigate to our "code" directory:
    - cd ..
    - cd code
- A few different text editors:
    - Vim (default text editor)
    - Nano (sligthly more user friendly, we will use)
- To edit plotting.R, type:
    - Nano plotting.R
        - Note: the keys that you see on the bottom are shortcuts
        - Now scroll to bottom at last piece of code and edit/add your text
        - once you have changed text and are happy with it:
            - Hit ctrl + O, Enter, ctrl + exit (this will take you back to command line)
- To check you actually edited document:
    - tail plotting.R
        - now you should be able to see the text you added/edited at the bottom of your plotting.R document

Intro to Git & GitHub

- Please log in to GitHub
- GitHub is a place to post code, save code, and make it publicly available
    - You can also use it to collaborate with other people on projects and track changes!
- Keeping track of changes is your code is important!
    - Version control is like an unlimited 'undo' and allows many people to work in parallel

Setting up Git

- Open terminal (mac)/GitBash (windows)
- Type:
    - git config --global user.name "YOUR GIT USERNAME"
    - git config --global user.email "THE EMAIL YOU USED ON GITHUB"
        - include quoation marks but place your own unique username/email inside quotation marks
    - git config --global init.defaultBranch main
        - Here we are telling GitHub we want the main branch to be called "main"!
    - git config --global core.editor "nano -w"
        - Set nano to be our default
    - git config --list
        - this will tell you everything you've set up
        - press 'q' to take you out of this pane

Creating a repository and adding files to a repository

- To get to correct directory (our ontario-report directory), type:
    - pwd
    - cd ..
    - pwd
- Note: A repository is a general lcoation, like a directory with a bunch of files and directories within it
- Make ontario-report folder into a repository, type:
    - git init
        - Note: this initializes an empty Git repository
    - ls -a
        - This checks if a .git directory has been made (reveals hidden directories)
    - git status
        - checks status of our repository
- At this point have just made a repository, like a container for all of your files.
    - Must use a series of commands to get files into your repo.
        - First: use "git add" to tell GitHub to essentially track a certain file
        - Second: can add a file to your repo through the "git commit" function once you make changes, it is like a saved change in your repository
        - This is complicated but will make more sense as we progress through the lesson!

- To make a new text file, type:
    - nano note.txt
        - In nano, type: "We plotted temperature and cell abundance."
        - Once you have written in your text file, use ctrl+o to write and save, hit enter to save and use ctrl+x to exit.
- To see contents of our file, type:
    - cat note.txt
- Type:
    - git status
        - Here you should see your note. txt file listed that your just created!

- git add note.txt
    - This tells Git to start tracking the note.txt file
- git status
    - Now it will say that there is a new file: note.txt
- git commit -m "Start notes on analysis"
    - Do not need to specify files here. Note we can commit everything we have added all at once.
    - -m specifies message
- git status
    - see there is nothing to commit now!
- git log
    - this will tell you what you've done so far in this repository
- nano note.txt
    - Add an additional line in nano: "Each point represents a sample."
    - Hit ctrl+O, enter and ctrl+x
- git status
    - you can see note.txt has been modified
- git diff
    - this command shows us what has changed in note.txt (compares previous version of note.txt with new version of note.txt)
- git add note.txt
- git commit -m "Add information on points"

To make a new file, type:

- nano more_notes.txt
    - In nano type: "Here I will put more notes."
    - Hit ctrl+O, enter and ctrl+x

To edit nano.txt and commit changes, type:

- nano note.txt
    - In nano, type: "Environmental groups are shown by colour".
    - Hit ctrl+O, enter and ctrl+x
- git add note.txt
- git add more_notes.txt
- git commit -m "Add notes about plots."

To check what we've done, type:

- git log
    - can click 'q' to exit

Intro to GitHub

- Open up GitHub in browser
- In upper right corner, under create new, create a new repository
    - this will allow us to link local and remote repositories
- Name repository "ontario-report"
- You should not select to add a README file, a .gitignore or a license.
- Click create repository
    - creates completely empty repository for us to populate with the files in our local

repository
- Under quick setup, click the SSH tab and copy the key.
- Go into your terminal (mac) or GitBash (windows) and type:
    - git remote add origin *HERE PASTE IN SSH KEY*
        - note the address shouls start with git@github.com
    - git remote -v
    - ls -al ~/.ssh
        - checks if an SSH key has already been set up
        - nobody in the workshop had an existing SSH key, so we will generate one
    - ssh-keygen -t ed25519 -C "YOUR EMAIL ADDRESS"
        - insert your email associated with your GitHub account inside the quotations
        - this makes a unique, identifying key, that gives permission for your computer to give files, info etc. to your GitHub account
        - if you see any prompts hit enter until you see your SSH key
    - ls -al ~/.ssh
        - this can see if you have generated your SSH key correctly
    - cat ~/.SSH/id_ed25519.pub
        - you should hopefully see the long string of text you saw ealier, ending in your email address
- You can then copy this entire line of text ending in your email, go to GitHub in your browser (your SSH key), click on your icon and then "settings"
        - Then click the "SSH and GPG keys" tab
        - Click to add new SSH Key
            - Title it however you please (Ex. Cassidy's Work Computer)
        - Paste in SSH key into the box that says "Key"
        - Click the green "Add SSH key button"
- Navigate back to terminal (mac) or GitBash (windows) and type:
    - ssh -T git@github.com
        - type yes if prompted!
        - your computer should be saying hi to you if the SSH key has worked okay!

Pushing/Pulling changes to GitHub

- Now we are going to take our local repo and push it to our remote repo! Exciting!
- Navigate back to your ontario-report directory in terminal (mac) or GitBash (windows) and type:
    - git push origin main
- Go to GitHub in your browser and you should be able to see the two files (note.txt and more_notes.txt) you pushed and committed! How cool!

- Hypothetically, if someone else is collaborating with you, you always want to make sure you have the most recent version of your file!
- To try pulling from GitHub to your local, type:
    - git pull origin main
        - If it says "Already up to date." it means no changes have been made and that your local and remote repositories match.
- Challenge: Upload the rest of your documents in your local ontario-report to Github. To do this, type:
    - git add code

        - git add figures
        - git add data
        - git commit -m "Uploading all directories"
        - git push origin main
- If you navigate to GitHub in your browser, refresh your page and you should see all of the directories you just pushed!
- You can also add a README, this can contain whatever you like, but typically contains information about the contents of the repository, how the repository is organized etc.

Collaborating with GitHub

- One person will act as the "Collaborator" and one person will act as the "Owner"
    - The Owner need to give the collaborator access. The Owner will follow these directions.
        - Go to ontario-report repo
        - Go to settings
        - Hit "Collaborators" and hit the green "Add people" button
            - enter your partner's username and then hit the green "Select a collaborator above" button
        - The Collaborator needs to accept the access by going to their "Notifications" tab
        - The Collaborator needs to download the Owner's repo onto their own computer. This is called cloning.
- The Collaborator must navigate to their Desktop on the terminal (mac) or GitBash (windows) and type:
    - git clone git@github.com:USERNAME/ontario-report.git ~/Desktop/other_user-ontario-report
        - if you type "ls" you should see a directory named "other_user-ontario-report" in your desktop.
    - cd other_user-ontario-report
        - this navigates into the new ontario-report directory you've gotten
    - nano notes.txt
        - now type whatever you want into this text file
    - git add notes.txt
    - git commit -m "Adding new line as a collaborator"
    - git push origin main
- The Owner can then try and see the new commits on their repo when looking at their GitHub on their browser.
- The Owner will now pull the edits your partner has made in your repo. The Owner needs to type the following in temrinal (mac) or GitBash (windows):
    - git pull origin main

Please take a moment to introduce yourself. Write your name, department, and goals for the workshop.

1. Gus Pendleton - Microbiology Department. My goal is to make sure you leave the workshop

more confident that you can analyze data with R, troubleshoot errors, and ask for help when you get stuck.
2. Hannah Sylvester- Animal Sciences Department. I hope to learn the basics of R for data analysis.
3. Kalem Hanlon - Nutritional Sciences Department. I hope to get a refresher and become more comfortable with data analysis using R.
4. Gurpreet Kaur, Department of Animal Science. I hope learn more about R and improve my data analysis skills.
5. Kathryn Miller, Food Bank of the Southern Tier & Cornell A&S alum. Hoping to learn if R is applicable to my current role working with data.

6. Lindsey Fei, Nutritional Science, I hope to learn more R code skills for helping on sata analysis for my project
7. Woojin Huh, Horticulture, I hope to be more familiar with data analysis using R and learn Git and Github as well.
8. Fatima (Fatemeh)- Plant Science. To recall how to code in R, learn Git and Github.
9. Isako (Izzi) Di Tomassi - Plant Pathhology and Plant-Microbe Biology; to refresh my skills and knowledge in time for a computational biology course I am taking this semester
10. Rachel Whiteside - Natural Resources - learn how to analyze and visualize data in R and learn Github!
11. Mark Watson - Microbiology - I hope to get a formal introduction to data analysis tools outside what I have taught myself casually over the years.
12.Dilmini Alhakoon, Clinical Sciences - learn Git and Github
13. Misha Kazi - Weill Institute for Cell and Molecular Biology. I hope to refresh my R skills and learn more about using R for data analysis
14. Kevin England- Microbiology Dept. I hope to learn more about making reports in R markdown, git, and the unix shell!
15. Carlos Irias, Department of Animal Sciences, I want to improve my skills using R, and learn how to do data analysis.
16. Casey Iwamoto, Department of Biological and Environmental Engineering, To refresh my R skills and learn how to use GitHub.
17. Mirza Arsyad, Plant breeding and genetics, to learn new skill in data analysis.
18. Hi, this is Upasana. I am a BBS grad student from Dr. Tobi Dörr's lab. Hoping to learn skill and practice R to feel more confident about understanding stats, making graphs, etc.
19. Taylere Herrmann. Plant Pathoglogy Department. Excited to learn more about R and become more familiar with using it.
20. Meg Shi, research technician in Microbiology. wanna refresh my mind on R and learn more about coding and rely less on ChatGPT
21. Samantha Davies, graduate student in biological and environmental engineering. I hope to improve my data management skills and learn coding techniques to aid in my data processing
22. Riley Harding, Plant Pathology (PPPMB), I want to learn Rmarkdown and refine my analysis workflow

Lesson #1: R for Plotting

Link to our first lesson: https://marschmilab.github.io/Cornell_Carpentries_Jan2025/01-r-plotting/index.html

Notes on RStudio settings

- Tools --> Global Options : this allows you to to customize your RStudio appearance

- Step #1: Save workspace to .RData should be set to NEVER under your Global Options
- Font size can be changed through the Zoom option in the Appearance tab of Global Options (Tools --> Global Options --> Appearances)

Getting the workspace set up

- Create project with your ontario-report file (should be in your Desktop, but this will depend on where you saved it to!)
    - use the Existing Directory button
- Create a new file --> R script
    - save this new file as **plotting.R**

Code

- type 2 + 2 and hit Run or use Control/Command + Enter to run the code
    - you should see 4 pop up in your console! This is where the outputs from the code you ran in the Editor will be
- **load in the tidyverse package**
    - library(tidyverse)
- going to the Help tab on the bottom right of your screen allows you to look through the Postit CheatSheets, which can help you to better understand the functions you're working with!
- **loading in our data**
    - sample_data <- read_csv("sample_data.csv")
    - running just sample_data will produce a tibble that has all the data from the excel file stored in it
    - <- is our assignment operator
        - you can do with with Alt + - on Windows and Option + - on Mac
    - # signs allow us to make comments on our code-- anything after a # sign will not be read as code therefore it will not be executed as a function
- #assign values to objects
    - name <- "agar"
    - name
    - year <- 1881
    - year
    - name <- "Fanny Hesse"
    - name
        - notice how this time, when you ran the name variable, its content was overwritten from being "agar" to "Fanny Hesse"
- #bad names for objects
    - 1number <- 3
        - this will give us an error, because we cannot name variables starting with a number. You can include numbers, so long as they are not the first character of the variable name.
        - 1number <- 3 won't work, but number1 <- 3 will work
    - objects are case sensitive! try the following:
        - Flower <- "marigold"
        - Flower
        - flower <- "rose"
        - flower

- sample_data <- read_csv("sample_data.csv")
- what happens if you just run read_csv() ? You get an error saying you didn't input the file name.
- **REMEMBER TO PERIODICALLY SAVE YOUR WORK BY SAVING YOUR RSCRIPT**
    - you can do Cmd/Cntrl + S, or hit Save in the File tab
    - unsaved Rscripts will have their titles be in a red font with an asterisk at the end of it
- #let's comment
    - Sys.Date() #outputs the current data
    - detwd() #outputs the current working directory
    - sum(5,6) #adds numbers
    - read_csv(file = 'sample_data.csv) #reads in csv file
- #create our first plot
    - ggplot(data = sample_data)
        - notice that this produces a grey box in the plots tab on the bottom right!
    - #check the relationship between temperature and microbial cell abundance
    - ggplot(data = sample_data) +
        - aes(x = temperature)
    - ggplot(data = sample_data) +
        - aes(x = temperature) +
        - labs(x = "Temperature (C)")
    - ggplot(data = sample_data) +
        - aes(x = temperature) +
        - labs(x = "Temperature (C)") +
        - aes(y = cells_per_ml) +
        - labs(y = "Cells per mL"
    - ggplot(data = sample_data) +
        - aes(x = temperature) +
        - labs(x = "Temperature (C)") +
        - aes(y = cells_per_ml) +
        - labs(y = "Cells per mL" +
        - geom_point()
    - ggplot(data = sample_data) +
        - aes(x = temperature) +
        - labs(x = "Temperature (C)") +
        - aes(y = cells_per_ml) +
        - labs(y = "Cells per mL" +
        - geom_point() +
        - labs(title = "Does temperature affect microbial abundance?")
    - ggplot(data = sample_data) +
        - aes(x = temperature) +
        - labs(x = "Temperature (C)") +
        - aes(y = cells_per_ml) +
        - labs(y = "Cells per mL" +
        - geom_point() +
        - labs(title = "Does temperature affect microbial abundance?") +

- aes(color = env_group)

- ggplot(data = sample_data) +
    - aes(x = temperature) +
    - labs(x = "Temperature (C)") +
    - aes(y = cells_per_ml) +
    - labs(y = "Cells per mL" +
    - geom_point() +
    - labs(title = "Does temperature affect microbial abundance?") +
    - aes(color = env_group) +
    - aes(size = chlorophyll)

- ggplot(data = sample_data) +
    - aes(x = temperature) +
    - labs(x = "Temperature (C)") +
    - aes(y = cells_per_ml) +
    - labs(y = "Cells per mL" +
    - geom_point() +
    - labs(title = "Does temperature affect microbial abundance?") +
    - aes(color = env_group) +
    - aes(size = chlorophyll) +
    - labs(size = "Chlorophyl (ug/L)",
        - color = "Environmental Group")

- ggplot(data = sample_data) +
    - aes(x = temperature) +
    - labs(x = "Temperature (C)") +
    - aes(y = cells_per_ml/1000000) +
    - labs(y = "Cells (millions/mL)" +
    - geom_point() +
    - labs(title = "Does temperature affect microbial abundance?") +
    - aes(color = env_group) +
    - aes(size = chlorophyll) +
    - labs(size = "Chlorophyl (ug/L)",
        - color = "Environmental Group")

- ggplot(data = sample_data) +
    - aes(x = temperature) +
    - labs(x = "Temperature (C)") +
    - aes(y = cells_per_ml/1000000) +
    - labs(y = "Cells (millions/mL)" +
    - geom_point() +
    - labs(title = "Does temperature affect microbial abundance?") +
    - aes(color = env_group) +
    - aes(size = chlorophyll) +
    - aes(shape = env_group) +
    - labs(size = "Chlorophyl (ug/L)",
        - color = "Environmental Group",
        - shape = "Environmental Group")

- #combined, neater code
    - ggplot(data = sample_data) +
        - aes(x = temperature,
            - y = cells_per_ml/1000000,
            - color = env_group,
            - size = chlororphyll) +
        - geom_point() +
        - labs(x = "Temperature (C)",
            - y = "Cells (millions/mL)"
            - title = "Does temperature affect microbila abundance?",
            - size = "Chlorophyll (ug/L)",
            - color = "Environmental Group")

- #importing datasets
    - https://seagull.glos.org/data-console/groups/379 is the link to the free-to-access data from the buoys
    - buoy_data <- read_csv("buoy_data.csv")
    - View(buoy_data) #remember that you can use the Tab button to complete object names inside parentheses!
    - dim(buoy_data) #gives us the rows and columns of our data set
    - head(buoy_data) #will show you the first few lines of your data, gives you a sneak peek!
    - tail(buoy_data) #the opposite of the head command-- this will show us the end of the dataset
- #plot some more
    - ggplot(data = buoy_data) +
        - aes(x = day_of_year,
            - y = temperature,
            - color = depth) +
        - geom_point()
- #check the structure of our data object
    - str(buoy_data)
- #back to plotting!
    - ggplot(data = buoy_data) +
        - aes(x = day_of_year,
            - y = temperature,
            - color = depth) +
        - geom_line()
    - ggplot(data = buoy_data) +
        - aes(x = day_of_year,
            - y = temperature,
            - group = sensor,
            - color = depth) +
        - geom_line()
    - ggplot(data = buoy_data) +
        - aes(x = day_of_year,
            - y = temperature,

- group = sensor,
- color = buoy) +
- geom_line()

- #introduce facets
  - ggplot(data = buoy_data) +
    - aes(x = day_of_year,
      - y = temperature,
      - group = sensor,
      - color = depth) +
    - geom_line() +
    - facet_wrap(~buoy)
  - ggplot(data = buoy_data) +
    - aes(x = day_of_year,
      - y = temperature,
      - group = sensor,
      - color = depth) +
    - geom_line() +
    - facet_wrap(~buoy, scales = "free_y")
- #facet grid
  - ggplot(data = buoy_data) +
    - aes(x = day_of_year,
      - y = temperature,
      - group = sensor,
      - color = depth) +
    - geom_line() +
    - facet_grid(rows = vars (buoy))
- #discrete plots
- #boxplots
  - ggplot(data = sample_data) +
    - aes(x = env_group,
      - y = cells_per_ml) +
    - geom_boxplot()
- #try different plots!
  - ggplot(data = sample_data) +
    - aes(x = env_group,
      - y = cells_per_ml) +
    - geom_violin()
  - ggplot cheatsheet: https://ggplot2.tidyverse.org/
- #back to boxplots!
  - ggplot(data = sample_data) +
    - aes(x = env_group,
      - y = cells_per_ml) +
    - geom_boxplot() +
    - geom_jitter()

- ggplot(data = sample_data) +
    - aes(x = env_group,
        - y = cells_per_ml) +
    - geom_jitter() +
    - geom_boxplot() #since the box plot was written second, it will cover the jitter points! be sure to write the boxplot part of the code first!!
- ggplot(data = sample_data) +
    - aes(x = env_group,
        - y = cells_per_ml) +
    - geom_boxplot() +
    - geom_jitter(aes(size = chlorophyll))

- #playing around with colors!
    - ggplot(data = sample_data) +
        - aes(x = env_group,
            - y = cells_per_ml) +
        - geom_boxplot(color = "pink") #notice how this makes the outline pink-- this is because 'color' and 'fill' are different in ggplot!!
    - ggplot(data = sample_data) +
        - aes(x = env_group,
            - y = cells_per_ml) +
        - geom_boxplot(fill = "pink")
    - sample(colors(), size = 10) #generates a random color palette for you to use
    - ggplot(data = sample_data) +
        - aes(x = env_group,
            - y = cells_per_ml) +
        - geom_boxplot(aes(fill = env_group))
    - ggplot(data = sample_data) +
        - aes(x = env_group,
            - y = cells_per_ml) +
        - geom_boxplot(aes(fill = env_group)) +
        - scale_fill_manual(values = c("pink", "tomato", "papayawhip"))
    - ggplot(data = sample_data) +
        - aes(x = env_group,
            - y = cells_per_ml) +
        - geom_boxplot(aes(fill = env_group)) +
        - scale_fill_brewer(palette = "Set1")

- #custom palette time!
    - #install.packages("wesanderson")
    - library(wesanderson)
    - ggplot(data = sample_data) +
        - aes(x = env_group,
            - y = cells_per_ml) +
        - geom_boxplot(aes(fill = env_group)) +

- scale_fill_manual(values = wes_palette('Cavacanti1')))

- #learning about transparency
  - ggplot(data = sample_data) +
    - aes(x = env_group,
      - y = cells_per_ml) +
    - geom_boxplot(fill = "darkblue", alpha = 0.3) #full opacity is alpha = 1
  - ggplot(data = sample_data) +
    - aes(x = env_group,
      - y = cells_per_ml) +
    - geom_boxplot(fill = "darkblue", alpha = 0.3)
  - #getting rid of outliers
    - ggplot(data = sample_data) +
      - aes(x = env_group,
        - y = cells_per_ml) +
      - geom_boxplot(fill = "darkblue",
        - outliers = FALSE,
        - alpha = 0.3)

- #univariate plots
  - ggplot(sample_data) +
    - aes(x = cells_per_ml) +
    - geom_histogram()
  - #change bin size
    - ggplot(sample_data) +
      - aes(x = cells_per_ml) +
      - geom_histogram(bins = 7)
  - #trying different plots!
  - ggplot(sample_data) +
    - aes(x = cells_per_ml) +
    - geom_density()
  - ggplot(sample_data) +
    - aes(x = cells_per_ml) +
    - geom_density(aes(fill = env_group), alpha = 0.5)
  - ggplot(sample_data) +
    - aes(x = cells_per_ml) +
    - geom_density(aes(fill = env_group), alpha = 0.5) +
    - theme_classic()
  - ggplot(sample_data) +
    - aes(x = cells_per_ml) +
    - geom_density(aes(fill = env_group), alpha = 0.5) +
    - theme_minimal()
  - ggplot(sample_data) +

- aes(x = cells_per_ml) +
- geom_density(aes(fill = env_group), alpha = 0.5) +
- theme_bw()

- ggplot(sample_data) +
    - aes(x = env_group,
        - y = cells_per_ml) +
    - geom_boxplot() +
    - theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))

- #saving plots
    - you can use the save as image button in the export tab from the plot tab in the bottom right window
    - you can also save your plots using code!
        - ggsave("awesome_plot.jpg", width = 6, height = 4) #this saves to whatever working directory you're in!
            - #to adjust the resolution use dpi = and then enter your desired resolution
            - ggsave("awesome_plot.jpg", width = 6, height = 4, dpi = 500)
    - you can temporarily save your plots to your environment by makign them objects:
        - box_plot <-
            - ggplot(sample_data) +
            - aes(x = env_group,
                - y = cells_per_ml) +
            - geom_boxplot() +
            - theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
        - box_plot
- #adding changes to the plot to make them black and white
- box_plot + theme_bw() #this doesn't change the boxplot object, it just adds a layer on top of it. to save the changes, you need to make a new object or overwrite the existing object, as shown below:
    - box_plot <- box_plot + theme_bw()
    - box_plot
    - ggsave("awesome_box_plot.jpg", width = 6, height = 4)

SAVE YOUR SCRIPT!!!!!!!!!!!!!!!! cmd/cntrl + S !!!!!!!!!!!!!!!!!!!!!!