Welcome to Etherpad!

This pad text is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents!

Get involved with Etherpad at http://etherpad.org
*
*OSU Data Carpentry website:
* https://osu-carpentry.github.io/2019-10-24/okstate/

*
*Data Management with Spreadsheets
*
***Download spreadsheet 1: https://ndownloader.figshare.com/files/2252083**

- Spreadsheets: useful for storing data, creating graphs, running formulas for statistical calculations and data analysis
- Good for DATA MANAGEMENT! But have to develop good practices...
  - 1) Create a new file with cleaned/analyzed data. Keep your original data file separate!
  - 2) Keep track of the steps you took when cleaning/analyzing your data (maybe create a README text file, just to document/write out your steps taken)
  -
  - **Quality control exercise spreadsheet: https://github.com/datacarpentry/spreadsheet-ecology-lesson/blob/gh-pages/data/survey_sorting_exercise.xlsx?raw=true**

  -

*https://data.research.cornell.edu/content/readme = good resource for how to create a good README file
*
*Cardinal rules for structuring data:

- Leave the original data alone/separate (don't change it/work directly in that file)
  - Organize your data appropriately (put all variables in appropriate labeled column; each row contains observations)
  - Don't combine multiple bits of info in one cell! (One cell should house strictly one piece of information!)
  - Remember to export cleaned data to a text-based format like CSV (comma-separated values); open-source, user friendly, more software can be used (non-proprietary like Excel files)
    - NOTE: Excel is a proprietary format. Unless a person has Microsoft Excel installed, they may not be able to access/view that data. This is why formats like CSV are recommended. They do not have proprietary restrictions. People are better able to access/view the data regardless of what spreadsheet software they may have.
-

Some comments about saving files as CSV: since Excel is proprietary, not all the features of a spreadsheet developed in Excel will be available if the file is opened in other software. Likewise, if you save an Excel spreadsheet as a CSV or TSV file, the formatting and other features specific to Excel will be lost.

***Exercise: Assessing Raw Sample Data**
*date not written correctly

*weight unit?
*species and name written in same box?
*format is not same in all year?
***Problems from the example (things to avoid/watch out for in your own practices):**
* Multiple tables
* Multiple tabs
* Not filling in zeros
* Using problematic null values
* Using formatting to convey info vs. just raw text values (e.g. highlight cell, text colors, fonts, bold)
* Placing comments/units in cells
* Entering more than one piece of info in a cell
* Using problematic field names
* Using special characters in data
* Including metadata in table
* Date formatting

- Preferred method: Separate ALL components of date into separate columns (column for year, column for month, column for day)
    - Recommended Excel formulas to run on a full date (e.g., 10/24/2019 in column G, row 5:
        - **=month(G5)** Extracts the month from the specified cell (assuming the value is a recognizable date)
        - **=day(G5)** Exctracts the day
        - **=year(G5)** Extracts the year
-

For special analysis items like "means" and "totals," save them to another file. Do not try to include them as separate, loosely attached
Also, try avoiding special formatting ("Date," "Currency," etc.)

Under the "Data" tab, you can set up rules/protocols for data entry values and set up notes for errors. (E.g., A column for months accepts only numbers 1-12. If you type 13, an error box will pop up, along with an explanation of the column's limitation to data value inputs of 1-12.
"Data Validation" will help identify errors detected within your dataset based on these established parameters.
You can use "Conditional Formatting" to run certain checks on data values in a column (e.g., duplicate values). Any targeted values of concern will be highlighted.

- NOTE: When running Conditional Formatting, have to make sure that all of the values within a column are formatted consistently.

*OpenRefine:
* If OpenRefine does not run, try entering in your browser: http://127.0.0.1:3333/ or http://localhost:3333/
* Data file link: https://ndownloader.figshare.com/files/7823341
*
*Any questions? Madison Chartier: madison.chartier@okstate.edu

*

*Intro to R - Kay Bjornen, Data Initiatives Librarian

*1. Set up folders for data_raw, data_output and fig_output

*2. Download the Code handout and save it under the name:

-
- data-carpentry-script
-

*https://datacarpentry.org/R-ecology-lesson/code-handout.R

You should have 4 windows/tabs:

- script (project)
  - console
  - environment
  - files
  -

If you are missing one, you may be missing the script window. Try double-clicking on the "source" window tab (right above "Console"). Otherwise, try: File--> New Project --> New Script.

Create 3 files:

- data_raw (to store original data files)
- data_output (to store data you have created, manipulated, changed)
- fig_output (to store our figures/visuals we create)

In the Script Window:
This is where you tell R what coding you want it to run. This documents all the code you run.

- ### = comment lines; not active code. This is where you can write notes for yourself regarding what the code is meant to do, any changes you've made, etc.
  - ?? = request for more information about a command/what a command code does

You will enter your code in the Script Window. The results of that code will run in the Console Window below it.

You can create objects/variables readily by writing the object/variable name and assigning it a value. Alt + - creates an assignment arrow to assign a value to an object/variable.
Ex) weight_kg <- 55 ; "weight_kg" is an object name. The object is assigned a value of 55.
You can reassign new values to objects anytime. Just follow the same procedure. Enter the object name, create an arrow, and identify the new value to be assigned.
BE ADVISED! When you change an object value, the change does not automatically apply to all instances where an object may have been used.
Object names are case-sensitive.
You cannot give function names to objects! This will confuse the system between running a function/command and pulling in an object value. Nouns for names; verbs for functions.

Avoid dots in object names.
To print an object value: just enter the object name in the script window.

* = multiplication

Console Window
This shows the results of the code you run. It does not save in the same way as the script window.

-

In Environment Window

- This will store all the objects with assigned values that you've created.
- If you assign new values to existing objects, this window will record the most recently assigned value.
- Includes objects, functions, and datasets

In the Files Window

- Make sure "Data_carpentry is your "working directory. Your working directory is typically where all the files you want R to
- (If it is NOT, write command "getwd()" in the Script Window)
- If you need help understanding what specific functions code commands do, you can select "Help" to see definitions of these specific commands. Otherwise, type "??" in front of the command code you're seeking to use.
- Where graphs and plots will be displayed.
- Packages = specialized tools that can be uploaded and attached to your R for specialized purposes

*
*
Functions = mini scripts meant to run specific commands strung together (for efficiency). The arguments are the values you enter into the function, on which you want the function to run. Some functions may be limited in how many arguments or what kind of arguments they can take.

Some functions:
sqrt() = square root function
round() = rounding function

- round(value, digits = number of digits to round to AFTER the decimal)
- Ex) round(3.14159, digits = 2) = 3.14

Vector = most common and stable datatype in R; string of values, numbers, or characters
Use c() to assign series of values to vectors (c = "combine")
Ex) weight_g <- c(50, 60, 65)
animals <- c("cat", "dog", "buffalo")
Cannot mix and match datatypes within a vector. Has to be ALL characters, or ALL numbers, etc.
length(vector_name) = identifies how many values/entries are contained within a vector
class(vector_name) = identifies datatyps of values in the vector (characters vs. numbers)
str(vector_name) = structure of vector. The datatype, how many entries, the very specific entries in the vector
To add more values to an existing vector: vector_name <- c(vector_name, newValue1, newValue2)
NOTE: How you order the values will influence how they appear: vector_name <- c(newValue1, vector_name, newValue2)

typeOf(vector_name) = identifices kind of vector (see below)

To see a particular value within a vector: vector_name[#] ; the number represents the position the value should be within the greater order of the vector
To reorder values within a vecotr: vector_name[c(1,2,5,4,3,6)] Shift the numbers around to reflect what currenty-assigned position you now want the value to take

Atomic vector = simplest R datatype linear vector with a single type. Includes character, numeric(double), logical, integer.

Other data structures: lists, matrices, data frames, factors

& (and); | (or)

weight_g[weight_g<50 | weight_g>=60] --> this will print all values that are either less than 50 or greater than or equal to 60 within a vector
== is the symbol for equal in this context

%in% --> use to locate a specific string within your vector

- ex) animals%in% c("rat", "cat", "dog", "duck", "goat") --> R will look for the 5 named values to see if they occur in the vector to which the %in% command is applied.
-

Remove missing/NA values: na.rm=TRUE, if running through a function

- Ex) mean(vector_name,na.rm=TRUE)

To purposely removed NA values from a vector:
1) vector_name[!is.na(vector_name)] OR
2) na.omit(vector_name)

Starting with Data
Study is of the species repartition and weight of animals caught in study plots. Data is
stored as a comma separated value (CSV) file.
Each row is a record for a single animal

*Manipulating and analyzing data with dplyr - https://datacarpentry.org/R-ecology-lesson/03-dplyr.html

*If you are joining us from yesterday:

- File > Recent Projects
- Click on the project we created yesterday
- Open code-handout.R
- If you can't find it, run:

*download.file("https://datacarpentry.org/R-ecology-lesson/code-handout.R", destfile =
* "./code-handout_day2.R")

- Run

*install.packages("tidyverse")
*library(tidyverse)
*surveys <- read_csv("./data_raw/portal_data_joined.csv")

Make sure you have folders in your directory called:

- data_raw
- data_output
- fig_output

*If you are starting from scratch:

- File > New Project > New Directory > New Project
    - Click **Browse** to select the directory you want to save this project in (e.g. Desktop, Documents) and click OK
    - Type in a name for the directory (e.g. data-carpentry)
    - Click Create Project
- In the bottom right corner, click New Folder (under the Files tab)
    - Name the new folder data_raw
    - Create two new folders called
        - data_output
        - fig_output
- Then run this code:

*download.file("https://datacarpentry.org/R-ecology-lesson/code-handout.R", destfile =
* "./code-handout.R")
This line of code will download the data file from the internet and save it to your computer.
It includes the URL to the code, and the folder and file name on your computer for where to save it.

- Click on code-handout.R in the bottom right corner
- Run

*

Purpose of a creating an "R Project": This is an easy way to set up a directory, where you can keep all your files, all your work in one consistent place.

Environment window has a "History" tab. This is a good place to look to keep track of all the code you've run. This is especially useful if you accidentally switch between the script and console windows to write and run your code.

list.files(): How to list all files within your current working directory, OR a file you name in the parentheses [e.g. list.files("data_raw")]

After installing a package into your library, you have to call the package into your project to use it.

- library(tidyverse) [You should have run this command already in setting up your work for the day.]

Review:
Create objects and assign values: x <- 5

Check datatype of object value: **typeof**(x)
Create a list: y <- **c(1, 3, 6)**

- REMEMBER: You cannot mix datatypes within a list!
- Indicate null values: z <- c(3, 7, **NA**, 1, 5, **NA**, 11)
- Eliminate null values: **na.rm** -----> mean(z, **na.rm = TRUE**)
- Get help on a function: **?**function_name (can also consult the **"Help" tab** in the Files Window)
- Construct dataframes: yz <- **data.frame**(y, z)
- To select a column within a dataframe: yz**$**y ; mean(yz**$**z, na.rm = TRUE)
-

NOTE: **read.csv** VS **read_csv**: read.csv is a bit of an older version of the command, but it runs consistently on basic R. read_csv has a bit more up-to-date functionality, but it runs strictly within the "tidyverse" package. Have to have that installed. If you have "tidyverse" installed, we recommend deferring to the read_csv command.

"Tibble" (tbl): More efficient dataframe generated by read_csv command. Prints only the first 10 rows of a dataframe.

To grab fundamental data about a dataframe:
nrows(surveys) = rows
dim(surveys) = rows and columns
str(surveys) = identifies all column headers, data types under those columns

dplyr and tidyr: packages that come with "tidyverse." Good for summarizing/grouping data, joining data, creating new variables, isolating components of a dataframe

Select a very particular set of columns to look at: **select**(surveys, plot_id, species_id, weight)

- Name the dataframe you're working in first, then follow with all the particular columns you want to look at.
- OR **select**(surveys, -record_id, -species_id) to identify columns you DON'T want to look at.

To isolate certain data values: **filter**(surveys, year == "1995")

- Identify the dataframe, then the particular variable and value desired.

Pipes (**%>%**): Way to join multiple commands/functions together when manipulating data. Time-saving tool. (Do this function, **THEN** run this one, **THEN** this one after that.)

- surveys_smal <- surveys **%>%** Grab the dataframe, THEN
  - filter(weight < 5) **%>%** Select all observations with weight values less than 5, THEN
  - select(species_id, sex, weight) Show only the 3 columns named.

*Keyboard shortcut for pipes: ctrl + shft + m

To remove/delete an object or dataframe: **rm**(surveys)

To clean your console:
Edit --> Clear Console
Ctrl + L

Little broom icon in upper righthand corner of the Console Window. (Recommended NOT to use this broom icon in the other windows, especially the environment window. This will delete EVERYTHING.)

Mutate: Create new columns based on the values in existing columns. (Example: perform operations on values registering as grams in one column, and create a new column, where they're listed as milligrams.)

- **mutate**(weight_kg = weight / 1000, weight_lb = weight_kg*2.2)
    - Within the function, name the new column(s) you want to create and assign them the values/operations they should run.
-

Correct indentations in coding: Ctrl + I

Reminder: is.na targets nulll values. To select these null values and have the program exclude those from your calculations, run the command with an exclamation point in front of the command:

- **!is.na**(surveys$weight) -----> Should focus on all the values that are NOT NULL in the weight column.

Gather and **spread** commands: Ways to manipulate/transpose the data

Export data: write_csv() [command best to use with "tidyverse"; write.csv command runs in more basic R]

- **write_csv**(surveys3, "data_output/surveys_wt_conv.csv")
    - Identify the dataframe you've created first, then specify the filename you want within the filepath to the appropriate folder.
    - NOTE: You can only use backward slashes, NOT forward slashes.

You have to export your data in order to save it on your hard drive. Otherwise, it's exclusively in RStudio, where it might not be saved.

*
*Data Visualization with ggplot2 - https://datacarpentry.org/R-ecology-lesson/04-visualization-ggplot2.html

If you didn't get through the last exercise, load the data for this exercise with the following:

*download.file("https://ndownloader.figshare.com/files/18106187", destfile =
* "data_output/surveys_complete.csv")
*surveys_complete <- read_csv("data_output/surveys_complete.csv")

***ggplot2.tidyverse.org** (Good source of useful help for ggplot2)

ggplot2 comes with "tidyverse."

- You'll specify the data you're working with, mapping for x and y axes (aesthetic), and a geom (to indicate what kind of visual you want)
- NOTE: There are many helpful instruction tools regarding ggplot online!
- **ggplot**(surveys_complete, # Call in the dataframe
    - **aes**(**x** = weight, # Identify the x and y axes
        - **y** = hindfoot_length)) +
    - **geom_point**() # Type of visual you want (point graph)

- Some arguments to manipulate the visual:
    - geom_point(**alpha** = 0.1) # "alpha" changes the density of the points
    - aes(x = weight,
        - y = hindfoot_length,
        - **color** = species_id) # "color" can be used to add additional info to the visual, outside the assigned axes

- geom_ can run multiple different visuals/graphics.
    - geom_**point()**
    - geom_**jitter()** (Bunch of points around the same place, spaces them out a bit more for easier interpretability)
    - geom_**boxplot()**
    - geom_**violin**() (Gives better sense of distribution than a traditional boxplot)
- Time serices:
    - geom_**line**()
    -

NOTE: You can layer geoms on top of each other (use them together).

- **ggplot**(surveys_complete,
    - **aes**(**x** = specied_id,
        - **y** = weight)) +
    - **geom_boxplot**() +
    - **geom_jitter**(alpha = 0.3, color = "tomato")

NOTE: If interested in interactive visualizations, there's an R app called "Shiny." Outside the scope of this lesson, but if you are interested, could be worth looking up.

count(): Way to calculate unique values occurring within specified columns

- **count**(genus, year)
- This will create a new column/variable in your dataframe to store those counts. Will typically store as "n"
    -
- You can add "groups" to incorporate another variable outside the x and y axes:
    - aes(x = year,
        - y = n,
        - **group** = genus) NOTE: This is why "color" works so well; it acts by grouping.

- my plot <- ggplot(yearly_counts,
    - aes(x = year,
        - y = n)) +
    - geom_line() +
    - **facet_wrap**(vars(genus)) + # Another way to create groups in the data by a variable, AND separate that data into its own plot (rather than reading them all together in one plot.
    - **scale_y_log10**() + # Way to manipulate readability of the visualization; changes the scaling of the y axis
    - **labs**(**title** = "Observed genera",
        - **x** = "Year of observation",
        - **y** = "Number of individuals") # Way to title your visual and label your axes

You can manipulate fonts and font sizes.
You can create/position legends.
DPI, width, etc.

ggsave("fig_output/genera_time.pdf", my_plot) : Saves your work on ggplot2. (Make sure you're creating objects with your work! So it's actually stored in an entity. Otherwise, it's just code that will not save.)

- Similar procedures, but reversed: identify the filename/filepath you want to save it to, then identify the object you want to save.
-

To install other packages: Go to the "Packages" tab in the Files Window.
Swirl: Interactive package for learning R in RStudio! Could be useful for continued work/education on using R.

Also, check out free e-book: **R for Data Science** by Garrett Grolemund and Hadley Wickham (goes over a lot of "tidyverse")

cran-r-project.org: Includes additional packages developed for R; very strong criteria for packages developed here to make sure they are easy and friendly to use.

*Databases Using SQL
Data Download: https://figshare.com/articles/Portal_Project_Teaching_Database/1314459

DB Browser: https://sqlitebrowser.org/dl/

Common misconception: **Databases and Excel spreadsheets are NOT the same thing**. Databases give data in tabular format, but they work very differently from Excel spreadsheets.

- **Databases** are often used as storage mechanism for long-term archiving of data, or to receive continual incoming data.
- **Excel spreadsheets**: where you can directly edit cells and use formulas based on other cells (the data you clean/manipulate)
- **Databases send commands/queries to a database manager**; the manager does calculations and queries on the user's behalf; the manager returns results in a tabular format. User does not have direct access or immediate editing privileges to the data stored in a database. The data will remain unaltered (your original, raw data).

Databases useful for:

- Improved quality control (no human error/malice)
- Efficient/fast
- Keeps original data separate from analysis procedures that change it
-

Database manager: software that does the work for the user. Recommend using open-source software

(it's free and supported by an active community of developers). We are using **DB Browser**. This implements **SQLite**.

SQL = Structured Query Language; this is a widely-used language that will work across a variety of software platforms

Primary key: very unique value/id, intended to occur only once and be uniquely assigned to a single entry for easy, distinct identification
Foreign key: takes you to a primary key in another table
Shared values between 2 different tables/datasets are what creates/establishes relations between datasets in a database.

Database design:
atomic values (one bit of data per one cell)
one field per type of info (each column contains values of a consistent datatype)
split different classes of data into different tables
relational databases connect the different classes of data together

Create a new database: **File --> New Database** (Filetype should be SQLite File)
Upload CSV file to DB Browser: **File --> Import --> Table from CSV file** (Check box to indicate **"Column names in first line"**)
To work with tabular data uploaded, select "**Modify Table**" in main working window of DB Browser. You will be prompted to define the datatypes for all of the columns in the dataset.

NOTE: SQL is NOT case-sensitive! SELECT, select, sELecT are all the same thing to SQL

Wildcard character: *

Fundamental commands:

- SELECT (Stipulate a a value/object you want grabbed; could be a good place to use the wildcard *)
    - FROM (Identify dataset where this value/object should be)
    - WHERE (Specify specific conditions that have to be met with the particular value in question; ex. year >= 1995)
    - ORDER BY ... ASC/DESC (Pick a field by which to sort the data values AND stipulate how to sort them [ascending or descending order])
        - ORDER BY taxa ASC;
    - GROUP BY

Conclude your query with a semicolon (;)

- SELECT *
- FROM surveys
- WHERE year >= 2000;

Boolean operators: AND, OR (way to set multiple conditions to be met)

- WHERE (year >= 2000) AND (species_id IN ('DM', 'DO', 'DS'));

-- Comments, where you can leave instructions regarding what the code does, any changes that have

been made. Readable text that is NOT executable code.

Aggregate functions (can call them in when initiating the SELECT command): COUNT(), SUM(), AVG(), MIN(), MAX()

- SELECT species_id, COUNT(*)
- FROM surveys
- GROUP BY species_id;

NOTE: SELECT species_id, COUNT(species_ID) AS occurrences

- Use of AS is to assign another term by which to reference a previous instance of code (in this case, COUNT(species_ID).
- Can then use shorthand/alternate name in lieu of rewriting the command.

CREATE VIEW: Way to create a very specific view of the data and create a dataframe/subset of that select data.

- CREATE VIEW summer_2000 AS
- SELECT *
- FROM surveys
- WHERE (year = 2000) AND (month > 4 AND month < 10);

You can then treat this new sub-dataset as you do the original data file you called into your database.

- SELECT *
- FROM summer_2000
- WHERE species_id == 'PE';

Joins: Way to join/combine data from two different tables, using primary and foreign keys

- JOIN command will come after SELECT and FROM; have to identify the point of commonality (the relation) between the 2 datasets using ON command. This commonality should be the primary keys for each dataset. (The primary key in the set you've been working with; the foreign key that corresponds with the dataset you are joining in)
- SELECT *
- FROM surveys
- JOIN species
- ON surveys.species_id = species.species_id;

You can select very specific columns you want pulled from each of the data files you're joining:

```
SELECT surveys.year, suveys.month, species.genus, species.species
FROM surveys
JOIN species
ON surveys.species_id = species.species_id;
```

With more complicated coding, you can actually use R and work in RStudio in lieu of working in DB Browser with SQL. Have to install RSQLite package with dbplyr. RSQLite connects to the SQL database, R will write and run code in R, but it will translate (via dbplyr) into the necessary SQL on the

backend to pull the info you want from the database. You will not have data stored in RStudio, but it will identify the SQLite database as the source for the data you need.

- src_dbi(database_name)
    - This command will identify the database you're consulting and will give you a preliminary view of all the data tables stored within that database.
- tbl(database_name, "table_name")
    - Identify the specific table you want to work with from the overall database.

*Any questions? Phillip Doehle: doehle@okstate.edu

*
*THANK YOU for attending DATA CARPENTRY!
*
*If you would like to become a helper or instructor with OSU Carpentries, please contact Phillip Doehle (doehle@okstate.edu) or Kay Bjornen (kay.bjornen@okstate.edu)!
*
*For more information and to review lesson materials: https://carpentries.org/ (we used the Ecology Workshop materials for our Data Carpentry session).