Welcome to Software Carpentry Etherpad!

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

Use of this service is restricted to members of the Software Carpentry and Data Carpentry community; this is not for general purpose use (for that, try etherpad.wikimedia.org).

Users are expected to follow our code of conduct: http://software-carpentry.org/conduct.html

Data Carpentry workshop June 25 - 26, 2018

https://punama.github.io/2018-25-06-UF-DataCarp/

Instructions if you still need to install components needed for the workshop: https://uf-carpentry.github.io/2018-03-20-UFDataSymposium/install.html

To download the spreedsheet we are using:
https://ndownloader.figshare.com/files/2252083

Tips for Excel:
Do not use spreadsheets for statistics, data analysis, code, figures
Spreadsheet are good for data entry and clean up and quality control
DO NOT use spreadsheet as lab notebook as difficult to be read by computers

Key Points

- Never modify your raw data. Always make a copy before making any changes.
- Keep track of all of the steps you take to clean your data.
- Organize your data according to tidy data principles.

Structuring data in spreadsheets:

1. variables in columns
2. observations in rows (single table, per spreadsheet, see "messy data" file for a good example of bad organization)
3. one cell/one datapoint (don't combine)
4. leave raw data be... (see above)
5. after "cleaning" data, export as csv format (or something similar), more versatile.
6. null cells--don't use zero, use NA (but this may be an issue) or leave cell blank
7. mixing values and units--don't--use the unit in the header name
8. Column naming--no spaces in column names, use "_" or "CamelCase", and no numbers at beginning

of column name

Metadata...
description of your data
good for you and other people to use your data
defining null values, more descriptive naming of a shorter column name
README file, what each data file is and how they are related
data & metadata should be physically separate from one another

Problems with the exercise spreadsheet

1. Consolidate 2013 and 2014 data into single table with both plot and species
2. Add note column for calibration notes instead of color coding or including with measurement
3. Remove units from measurement values (note in column heading)
4. Clarify NA code - Not Applicable , or an actual species class?
5. Consistent date codes - MM-DD-YYYY
6. Separate species and sex into separate columns

Quality Control Tools

- Data Validation
    - Data --> Data Validation
    - Goal of data validation is to prevent 'bad' data from being entered in the first place
    - Data validation also makes entry easier and the data more robust
- Sorting
    - Data for sorting exercise: https://github.com/datacarpentry/spreadsheet-ecology-lesson/blob/gh-pages/data/survey_sorting_exercise.xlsx?raw=true
    - Data --> Sort
    - Data --> Filter
- Conditional Formatting
    - Home --> Conditional Formatting
    - Conditional formatting provides a quick visual cue for potentially problematic data

https://github.com/datacarpentry/spreadsheet-ecology-lesson/blob/gh-pages/data/survey_sorting_exercise.xlsx?raw=true

A few takeaways:

Spreadsheets are very good for recording data but not necessarily for analyzing data
Always save your raw data separately
Record the steps you took to manipulate/clean your data
Export your data to CSV (Comma Separated Values) so you can work with it in other programs

10:45am - Open Refine for data cleaning (http://www.datacarpentry.org/OpenRefine-ecology-lesson/)

For data:

Download this data file to your computer: "https://ndownloader.figshare.com/files/7823341

Benefits to open refine:
- keeps a record of changes (Reproducible)
- open resource (free things are good things in life!)

Installing:
Follow the Setup instructions to install OpenRefine.
If after installation and running OpenRefine, it does not automatically open for you, point your browser at http://127.0.0.1:3333/ or http://localhost:3333 to launch the program.

NOTE: use either Firefox or Google Chrome for OpenRefine

importing data - When a table is imported into OpenRefine, all columns are treated as having text values.

Faceting - to facet for e.g. click "scientificName" --> click arrow --> facet --> text facet (this will open a facet on the left column)

clustering - click the "scientificName" text facet (in the left) --> click the cluster tab

- merge can combine the clusters together

split - to split we click "scientificName" --> click arrow ---> Split into several columns...--> In the pop-up, in the Separator box, replace the comma with a space, Uncheck the box that says Remove this column

A little more on numeric facets:

- For a column you transformed to numbers, edit one or two cells, replacing the numbers with text (such as abc) or blank (no number or text).
- Use the pulldown menu to apply a numeric facet to the column you edited. The facet will appear in the left panel.
- Notice that there are several checkboxes in this facet: Numeric, Non-numeric, Blank, and Error. Below these are counts of the number of cells in each category. You should see checks for Non-numeric and Blank if you changed some values.
- Experiment with checking or unchecking these boxes to select subsets of your data.

COMBINING COLUMNS:
click the column you want to combine e.g. genus --> edit column --> add column based on column genus. In expression box input:
value + " " + cells['species'].value. This combines the current column (genus - which is value) with species and the " space" makes a space between the two

to combine two columns note that this will work only on numeric columns (http://guides.library.illinois.edu/openrefine/combining)

Scatterplots - for example recordID, and use the pulldown menu to > Facet > Scatterplot facet. A new window called Scatterplot Matrix will appear. There are squares for each pair of numeric columns

organized in an upper right triangle. Each square has little dots for the cell values from each row. NOTE: scatterplot only works for numeric data

- We can examine one pair of columns by clicking on its square in the Scatterplot Matrix A new facet with only that pair will appear in the left margin.

Exporting out data - Click the Export button in the top right and select Export project.

A few takeaways:

- OpenRefine is a powerful, free and open source tool that can be used for data cleaning.
- All changes are being tracked in OpenRefine, and this information can be used for scripts for future analyses or reproducing an analysis.
- When importing data, OpenRefine will treat all columns as text values and it would be wise to transform columns with numbers in numeric (especially those you will work with).
- Faceting and clustering approaches can identify errors or outliers in data.
- OpenRefine provides a way to sort and filter data without affecting the raw data.
- OpenRefine also provides ways to get overviews of numerical data.
- Cleaned data or entire projects can be exported from OpenRefine.
- Projects can be shared with collaborators, enabling them to see, reproduce and check all data cleaning steps you performed.

Extra resources:
https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users (has some info about Geo-coding)
http://enipedia.tudelft.nl/wiki/OpenRefine_Tutorial

1:00 pm - Data Management with SQL

before we start:
Install DB Browser for SQLite: https://sqlitebrowser.org/
Download this data to your computer: http://dx.doi.org/10.6084/m9.figshare.1314459. Click on **Download all** to download the zip file. Unzip it to a location that you can easily find on your computer.

Basic process for working with data: Raw Data --> Clean Data --> Import and Analyze --> Results --> Visualize
Questions are (based on plots.csv, species/csv, and survey.csv in the data folder):

- How has the hindfoot length and weight of *Dipodomys* species changed over time?
- What is the average weight of each species, per year?
- What information can I learn about *Dipodomys* species in the 2000s, over time?

What kinds of operations do we need to perform in order to answer questions about the data?

- select subsets of the data (rows and columns)
- group subsets of data

- do math and other calculations
- combine data across spreadsheets

Relational databases provide efficient representation of data and preserve your original raw data

- Database is essentially a series of tables and linkages
- Relational databases are able to work on very large datasets

File --> Open Databse --> portal_mammals.sqlite

Browse Data tab allows you to see the data in a table-like format

Create a new database

- Import data: File --> Import --> Table from CSV --> surveys.csv --> column names in first line
- Do the same for plots.csv and species.csv

Modify Table allows you to change the data types for each field

- Text, Integer, Real, Numeric, BLOB (Binary Large OBject)

Execute SQL tab facilitates things like queries. **Good practices to note:**

- Use all CAPS for actions you are taking
- Use all lowercase for the table/field you are using
- End the query with a semicolon
- Start a line with two dashes (--) to add a comment; good to document your code
- Recommended order: SELECT --> FROM --> WHERE --> ORDER BY

List of functions: http://www.sqlitetutorial.net/sqlite-functions/

```
-- Get times from surveys table
SELECT year, month, day
FROM surveys;

-- Wildcard search with a limit of 10 rows returned
-- The * selects the entire record
SELECT *
FROM surveys
LIMIT 10;

-- Distinct function finds the unique values
SELECT DISTINCT species_id
FROM surveys;
```

```
-- Distinct works on multiple columns too
SELECT DISTINCT year, species_id
FROM surveys;


-- You can include calculations in your query
SELECT year, month, day, weight/1000.0
FROM surveys;
```

And yes, there are mice that weight 8 grams. This species (Perognathus flavus) is the cute silky pocket mouse. Its average weight is 6-9 grams.

Be careful of empty values! These could result in zeros being inserted into your data.
You also might need to coerce the calculation to Real vs Integer (e.g., divide by 1000.0 instead of 1000)

```
-- Write a query that returns the year, month, day, species_id and weight in mg.
SELECT year, month, day, species_id, weight*1000
FROM surveys;


-- Filtering = WHERE
-- Get all species_id that are DM
SELECT *
FROM surveys
WHERE species_id='DM';


-- Multiple filter criteria
SELECT *
FROM surveys
WHERE (year >= 1985) AND (year <= 2000);

-- Another example of multiple filter criteria
SELECT *
FROM surveys
WHERE ((species_id  = 'DM') OR (species_id = 'DO')) AND (year = 2000);


-- Produce a table with all records from Plot 1 where all the wieghts are greater than 75 grams. Return the date, species ID, and weight in kg.
SELECT day, month, year, species_id, weight/1000.0
FROM surveys
WHERE (plot_id = 1) AND (weight > 75);


-- IN function is another way to do filtering
```

```
SELECT *
FROM surveys
WHERE (year >= 2000) AND (species_id IN ('DM', 'DO', 'DS'));


-- Sorting can be accomplished via ORDER BY
-- Ascending = ASC; Descending = DESC
SELECT *
FROM species
ORDER BY taxa ASC;

-- You can order by multiple categores too
SELECT *
FROM species
ORDER BY genus ASC, species ASC;

-- Another example using multiple cirteria
SELECT genus, species
FROM species
WHERE taxa = 'Bird'
ORDER BY species_id ASC;


-- Let’s try to combine what we’ve learned so far in a single query.
-- Using the surveys table write a query to display the three date fields, species_id, and weight in
kilograms (rounded to two decimal places), for  individuals captured in 1999, ordered alphabetically by
the species_id.
SELECT year, month, day, species_id, ROUND(weight / 1000.00, 2)
FROM surveys
WHERE (year = 1999)
ORDER BY species_id;



-- COUNT counts the number of instances
-- SUM adds up all the values
-- ROUND will round things to the secified number of decimal places
-- You can also combine this with GROUP BY
SELECT species_id, COUNT(*), SUM(weight)
FROM surveys
GROUP BY species_id;


-- You can rename an output column by creating an alias using the AS keyword
SELECT species_id, COUNT(*) AS occurrences
FROM surveys
GROUP BY species_id;
```

```
-- The HAVING keyword is simlar to the WHERE keyword except that it is an aggregator
SELECT species_id, COUNT(*) AS occurrences
FROM surveys
GROUP BY species_id
HAVING occurrences > 10;


-- You can save the queries you generate so you can reuse them later. Thus, making your work
reproducible.
-- To do this, you can create a view, which is saved to the database structure.
CREATE VIEW summer_2000 AS
SELECT *
FROM surveys
WHERE (year=2000) AND ((month>4) AND (month<10));

-- Once a view is created, you can query it the same way as you would do another table.
SELECT *
FROM summer_2000
WHERE species_id = 'PE';



-- Dealing with blank/null values
-- Note that SQL will treat blanks as 0 values, which can give you erroneous results (like when doing an
average)
SELECT AVG(weight)
FROM summer_2000
WHERE (species_id = 'PE') AND weight != "";

-- Theoretically, the following should work as well (but it was not working in class):
  AND weight IS NOT NULL;

-- Some selections automatically remove the nulls, for example:
SELECT *
FROM summer_2000
WHERE species_id = 'PE' AND weight;


--JOINING TABLES
---- ON keyword specifies the common fields

SELECT *
FROM surveys
JOIN species
ON surveys.species_id = species.species_id;

--alternative way
SELECT *
FROM surveys
```

```
JOIN species
USING(species_id)

-- Another way to join that limits the number of columns returned
SELECT surveys.year, surveys.month, surveys.day, species.genus, species.species
FROM surveys, species
WHERE survey.species_id = species.species_id;

-- another way
SELECT surveys.year, surveys.month, surveys.day, species.genus, species.species
FROM surveys, species
USING(species_id);


-- Challenge: Write a query that returns the genus, the species name, and the weight of every individual captured at the site
SELECT species.genus, species.species, surveys.weight
FROM surveys
JOIN species
USING(species_id);

-- Alternative way with fewer keystrokes
-- However, be careful in cases where the tables have multiple common column names
SELECT genus, species, weight
FROM surveys, species
USING (species_id);

-- Removing the nulls in the above query
SELECT genus, species, weight
FROM surveys, species
USING (species_id)
WHERE weight;
```

Take a look at the summary challenge at the bottom of the lesson page: http://www.datacarpentry.org/sql-ecology-lesson/03-sql-joins/index.html

###Day 2:
Morning: Introduction to R

NOTES: Please make sure you have a downloaded Rstudio

R: What is it good for?
- open source, so many available packages that can be used
- manipulating data and doing stats in the same place
- reproducible, so anyone can take the code and run the exact same anaylses
- very flexible graphics for making plots
- LOTS of help to be found in books, online, etc.

- many test datasets are available, which you can find through data() function

RStudio
- interface for using R
- helps organize what you're doing in R
- four main panels: original panels are the script (top left), console (bottom left), environment/history panel (top right), and misc. (bottom right)
- script is what you can save, reopen, and run again
- console is where code actually runs
- to comment, use the pound sing #

variable/object - item you're assigning a variable to
the way to assign a value is <-
shortcuts: Mac: option + -

- Windows: alt + -

what's the difference between using the arrow versus an equal sign?

- - at this point, there isn't much difference
- - it can get a little more confusing as coding goes along, because = and == can have different means
- - here's a blog post about the difference if you're interested:
- https://renkun.me/2014/01/28/difference-between-assignment-operators-in-r/

How to run code:

- Click the 'Run' button

Mac: Command + Enter
Windows: Control + Enter

R can do basic arithmetic, such as 3 + 5

to have R output the value of the object you've assigned a name to:
1) print(weight_kg)
2) put parenthases around the code where you assign it, (weight_kg <- 50)

one reason to assign a name to a value is to continue manipulating that value
ex. 2*weight_kg = 100

R already has some base functions that are included in the program
- you can find them by using the help tool
- or you can type ?function, such as ?sqrt. This will explain what the function does, what the argument inputs and defaults are, etc
examples: sqrt() for the square root, round(3.2)
- many functions have a default value for a lot of the inputs
- if you put your mouse in the parentheses for the function and press tab, it will tell you what arguments the function needs
- could also use the args() function, which will give you the arguments

Google is your best friend. Srsly tho. And the more you use R, the easier googling the correct thing will

be

Vectors:
vector is a series of values

- can use the c() function, which stands for concatenate
- produces an item that has multiple values
- can hold numbers, character strings, etc.
- length() function will tell you how many values are in the vector
- when you create a vector, it should be all the same data type

Data Types

- logical: TRUE or FALSE
- character: letters
  - for character strings, you need to put quotations around it
- numeric, integer, double: all numeric data types
- class() function will show you what your data type is
- str(), or structure, will show you the data type for each column in that object
- Factors - allows you to categorize and label items; good for making categorical data

to see the data that in your environment, you can click on it in the environment or use the View() function

also, different ways to store the data

- lists - multiple types of forms
- matrices
- dataframes
- vectors

Dataframes

- give it a name!
  - data <- data.frame(animals, weight_g)
- subset the dataframe
  - you can use the $ to specify columns in a dataframe
  - can subset by rows, columns, or a combination of the two
  - square brackets index items
  - always [row, column]
  - leaving one of those blank will return of the matching rows or columns
  - ex - data[1,] will give you the first row and all the columns
  - ex - data[1,2] will give you the value in the first row and second column
  - ex - data[1:3,] or data[c(1,2,3),] will give you the first three rows and all columns
- can subset based on conditions
  - data["weight_g" > 60,] OR data[data$weight_g > 60,] # find the rows where that is true in the weight_g column
  - creates a logical vector and finding everything that equals TRUE
  - can use the == to make a conditional that equals something specific
  - & is used for 'and' and | is used for 'or'
  - can also write the condition outside of the square brackets

        - cond <- data$animals == 'mouse' | data$animal == 'rat'
        - that would mean find the values in the animal column that is equal to mouse OR rat
        - data[cond,]
- dealing with missing values
    - typically defined as 'NA'
    - can create vectors with missing values
        - ex - animals <- c("mouse", "rat", "dog", NA)
        - ex - heights <- c(10, 20, 10, NA, 6)
            - if you run mean(heights), it will return NA because it can't calculate when an NA is present
            - so instead, you can remove NAs by writing mean(height, na.rm = TRUE), which will then caluclate on the numbers only

Working Directories

- - the place where you store all the things you'll need in your analysis
- - you can check your current working directory location with getwd()
- - you can set your working directory with setwd()
- - you can click on the small button with the three dots in the top right of the files window to help you figure out where you are

Projects

- - a folder where you can put everything needed to run the code. You can send someone else a project, and they can run everything in that project imeediately
- - to make a new project:
    - File > New Project > New Directory > Empty Project
- - the subdirectory that you assign will be the working directory for the project
- - when you open that project, you will automatically be in that working directory

Getting data into R:
- to download data from the internet, use

- download.file(url = "https://ndownloader.figshare.com/files/2292169", destfile = "Data/portals_data_joined.csv"), which will get data from a URL

- to read in data that is in the Data folder, use the read.csv()
- also give it a name so you can work with it

- surveys <- read.csv("Data/portals_data_joined.csv")

- you can add the stringsAsFactors = TRUE or FALSE in the read.csv() to indicate where you want columns with text to be considered factors or not

Look at your data:
head() gives you the first 6 lines (or more/less if you specify different) of the data
str() shows you the structure of the data you've read in
summary() will give you the summary of the data

str(surverys$sex)
- give you factor with 3 levels: blank, F, M
- that means that 1 = "", 2 = F, 3 = M

- levels() will show you the levels in the factor
- nlevels() shows you the number of levels in the factor

you can always reorder the factors levels
sex <- c("M", "F", "F", "M", "M")
sex <- factor(sex, levels = c("M", "F"))

if you need to convert a numeric column out of factors, have to convert to character first to get the actual numbers rather than the assigned factor numbers

- as.numeric(as.character(height))

to create a new column
surveys$sex2 <- as.character(surveys$sex)
characters don't have any numeric value to them

Changing Levels:
levels(sex) gives "" "F" "M"
levels(sex)[1] will show you the first level
levels(sex)[1] <- "unknown" will change the first level from "" to whatever you assign it to me
levels(sex)[2] <- "female"
levels(sex)[3] <- "male"
levels(sex) will now return "unknown" "female" "male"

Formatting dates:
install.packages("lubridate")
library(lubridate)

Day 2 afternoon: more R
http://www.datacarpentry.org/R-ecology-lesson/03-dplyr.html <- lesson notes (with scripts!)

We're going to be using functions from the "tidyverse" packages
install.packages("tidyverse")

You can learn more about an R package from vignettes: just google the package name and you can find help info and examples

We'll be using a package called "dplyr"
(Google "dplyr cheat sheet" to get a nice pdf showing common things it can do)

https://www.rstudio.com/resources/cheatsheets/

In Rstudio, main menu bar, "Help"---> "Cheatsheets"

####code####

library("tidyverse") #loading the package

```
surveys <- read_csv("data/portal_data_joined.csv") #reading data
str(surveys) #checking the dataframe structure

#subset columns of the data

select(surveys, species)

#subset rows of the data
filter(surveys, species == "albigula", year == 2000) #need to put strings into quotes ""

#pipes ---
# using combination keys of "cmd (ctrl) + shift+M to" input  pipes "%>% "

filter(surveys, weight < 10) %>%
```

- `select(year, weight) %>%`
- `filter(year < 1990)`

```
#pipe example

sex <- as.character(as.factor(c("male", "female", "female", "male", "male"))

#if you use pipe then it looks below, and it's easy to comment each code line
sex <- c("male", "female", "female", "male", "male") %>%
```

- `as.factor() %>% #convert to factor`
- `as.character() #convert to character`

Note: the pipe %>% comes from the tidyverse package (it won't work if you don't have that library loaded)

If when you load the tidyverse package you see a print out called "Conflicts" this means that there a functions with the same name in two or more packages.
For example, the dplyr package has a function called filter(), and so does the stats package! R will default to using the most recently load package.
You can specify the package by using the notation package::function (i.e. dplyr::filter()).

Functions in dplyr:

```
select()       -- Used to subset columns
filter()        -- Used to subset rows
mutate()      -- Creates a new column
group_by()   -- Group dataset based on a column
summarize() -- compute summary statistic based on group_by()
arrange()     -- sort/re-organize the dataframe


# Change or create new variables ----
# one way to do this:
```

```
surveys$weight_kg <- surveys$weight / 1000

# another way to do the same thing:
mutate(surveys, weight_kg = weight / 1000)

# alternately, create a whole new data frame that is a copy of the old one, but with the extra column
surveys_kg <- mutate(surveys, weight_kg = weight / 1000)



# When you close out of Rstudio, it asks if you want to save the workspace image. What does this mean?
#   - the "workspace" consists of the stuff in the Environment tab -- functions, variables, values etc. Saving it means that the next time you open RStudio all these things will still be  there. This might not be what you want, for example if you come back to something a year later you may have all these values and you don't know where they came from
#   - You can change RStudio's default behavior when it comes to "workspace" with Tools > Project Options > R General > [change all three options to No]
#   - Using Project Options only changes the behavior for the Rproject you are working on. Change "global" default (i.e. default for anything you're working on regardless of which project) with Tools > Global Options > R General and uncheck the box next to "Restore .RData into workspace at startup"
# Restart R session: command+shift+F10 or going to Session > Restart R



# Challenge ----
# Create a new data frame from the surveys data that meets the following criteria:
   # 1. Contains only the species_id column and a new column called hindfoot_half containing values that are half the hindfoot_length values
   # 2. In this hindfoot_half column, there are no NAs and all values are less than 30

# Step 0: use the surveys data set
surveys_filtered <- surveys %>%

# Step 1: make the hindfoot_half column
mutate(hindfoot_half = hindfoot_length / 2) %>%

# Step 2: select "species_id" and "hindfoot_half"
select(species_id, hindfoot_half) %>%

# Step 3: filter hindfoot_half for values < 30 and not NA
filter(hindfoot_half < 30, !is.na(hindfoot_half))

# Note: when we removed NA values earlier (when we were using the "mean" function), we could use the argument "na.rm = TRUE". However the "filter" function we're using now does not allow the argument "na.rm" and so we used the funtion "is.na()" which identifies values that are NA, combined with the "!" operator, which just means "not"



# Create summarized output ----
```

```
# we want to create groupings of the data based on sex, and find the average weight by sex
surveys %>%
  group_by(sex) %>%
  summarise(mean_wgt = mean(weight, na.rm = TRUE))

# The code above should return a 3x2 tibble with columns "sex" and "mean_wgt"


# Note: because we're using the function "mean" again, we can use "na.rm = TRUE" to tell R to ignore NA values
# Note: the tidyverse functions will work with American or British spellings (i.e. "summarize" vs. "summarise")

# Now we want to do a more complicated grouping: sex and species
surveys %>%
  group_by(sex, species_id) %>%
  summarise(mean_wgt = mean(weight, na.rm = TRUE)) %>%
  arrange(species_id) %>%
  print(n = Inf)   # print the full output

# Note: the "arrange" function here re-orders the rows of the output tibble by the species_id column (alphabetically)
# Note: you'll see in this output that the mean_wgt for the top few rows are "NaN". This is short for "not a number" and is slightly different from "NA." In our example, it returned NaN because all values of weight for that group were NA (i.e. all entries with species_id AH and sex F all had weights that were NA, and so there was nothing to take the mean of)


# tally within groups (count the number of rows when data is grouped by sex and species_id)

surveys %>%
  count(sex, species_id)

# Challenge:

   # 1. how many individual rodents were in each plot_type
   Answer:
      surveys %>%
        count(plot_type)

   # 2. use group_by() and summarize() to find the mean, min, and max hindfoot_length for each species
   Answer:
      surveys %>%
        group_by(species_id) %>%
        summarize(mean_length = mean(hindfoot_length, na.rm = TRUE), min_length = min(hindfoot_length, na.rm = TRUE), max_length = max(hindfoot_length, na.rm = TRUE))


# Writing out cleaned data
```

```
write_csv(surveys_filtered, "Data_output/surveys_filtered.csv")

# Create cleaned version of data table with rows containing missing data removed
surveys_complete <- surveys %>%
  filter(!is.na(weight),
         !is.na(hindfoot_length),
         !is.na(sex))

write_csv(surveys_complete, "Data_output/surveys_complete.csv")



# Restart R session (Session > Restart R)
library("tidyverse")

surveys_complete <- read_csv("Data/surveys_complete.csv")

# ggplot figure ----
# there is a cheatsheet for ggplot figures: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

# this is an empty plot with axes
ggplot(data = surveys_complete,
         mapping = aes(x = weight,
                       y = hindfoot_length))

# scatterplot
ggplot(data = surveys_complete,
         mapping = aes(x = weight,
                       y = hindfoot_length)) +
   geom_point()

# line graph
ggplot(data = surveys_complete,
         mapping = aes(x = weight,
                       y = hindfoot_length)) + #"+" means adding a new plot layer
   geom_line() +
   geom_point()

# adding color
ggplot(data = surveys_complete,
         mapping = aes(x = weight,
                       y = hindfoot_length)) +
   geom_point(colour="pink", alpha=0.5) #alpha=1, means solid color, and 0 means transparent; otherwise between 0-1

# color is catgorized by gender
ggplot(data = surveys_complete,
         mapping = aes(x = weight,
```

```
                        y = hindfoot_length,
                        color=sex)) +
  geom_point(alpha=0.1)



ggplot(data = surveys_complete,
       mapping = aes(x = weight,
                        y = hindfoot_length,
                        color=species_id)) +
  geom_point(alpha=0.1) +
  theme_bw() #adding black-white theme

#modify the labels
ggplot(data = surveys_complete,
       mapping = aes(x = weight,
                        y = hindfoot_length,
                        color=species_id)) +
  labs(x="Weight (grams)", y="Hindfoot Length (mm)") +
  geom_point(alpha=0.1) +
  theme_bw() #adding black-white theme

#manually defined the color
ggplot(data = surveys_complete,
       mapping = aes(x = weight,
                        y = hindfoot_length,
                        color=species_id)) +
  labs(x="Weight (grams)", y="Hindfoot Length (mm)") +
  geom_point(alpha=0.1) +
  scale_color_manuak(values=c("F"="blue",
                                                    • "M"="Orange"))
  theme_bw() #adding black-white theme


#boxplot

ggplot(data = surveys_complete,
       mapping = aes(x = species_id,
                        y = hindfoot_length,
                        color=species_id)) +
  labs(y="Hindfoot Length (mm)") +
  geom_boxplot(alpha=0.1) + #if you want the outlier disappear, use outlier.alpha=0
                                                  • # changing color: outlier.color="green"
  scale_color_manuak(values=c("F"="blue",
                                                    • "M"="orange"))
  theme_bw() #adding black-white theme, (e.g., theme_dark())
```

#line plots

yearly_counts <- surveys_complete %>%

- count(year, species_id)

ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n)) +
- geom_points()

ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n)) +
- geom_line()

ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,
- group=species_id)) +
- geom_line()

#count for different species on one plot
ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,
- color=species_id)) +
- geom_line()

seperate panels for each species
ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,
- color=species_id)) +
- geom_line() +
- facet_wrap(~ species_id) +
- theme(axis.text.x=element_text(angle=90))

####Challenge
#For each species, draw two linesm one for each sex

ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,

- color=species_id)) +
- geom_line() +
- facet_wrap(~ species_id) +
- theme(axis.text.x=element_text(angle=90))

#data preparation
yearly_counts <- surveys_complete %>%

- count(year, species_id, sex)

ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,
- color=sex)) +
- geom_line() +
- facet_wrap(~ species_id) + #"~" by which variable
- theme(axis.text.x=element_text(angle=90))

#if you want to remove some of species id, e.g., "BA"

yearly_counts <- surveys_complete %>%

- filter(spcies_id != "BA") %>%
- count(year, species_id, sex)
-

ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,
- color=sex)) +
- geom_line() +
- facet_wrap(~ species_id) + #"~" by which variable
- theme(axis.text.x=element_text(angle=90))

counts_by_sex_plot <- ggplot(data=yearly_counts,

- mapping=aes(x=year,
- y=n,
- color=sex)) +
- geom_line() +
- facet_wrap(~ species_id) + #"~" by which variable
- theme(axis.text.x=element_text(angle=90))

#run the object name will do the plotting

#saving the ggplot

```
ggsave("Data_output/counts_by_sex_plot.png", counts_by_sex_plot, dpi=300, device="png")


#some R meet-ups

www.r-gators.com

can ask questions on R-gator serve
FB R-users
Twitter: @UFCarpentries
```