Welcome to The Carpentries 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 Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

---

2022-03-07 Data Carpentry for Environmental Scientists workshop

- Instructors:
- Alesia Hallmark
- Gregory Maurer
- Edmundo Medina
- Jonathan Wheeler

Helpers:

- Darren James
- Geovany Ramirez
- Diana Toups-Dugas

NM EPSCOR demographic survey: https://www.surveymonkey.com/r/DQPVFWD

Resources
A link to the Data Carpentry Ecology modules: https://datacarpentry.org/lessons/#ecology-workshop

The VIM text editor: https://www.vim.org/

R package resources:
Searching CRAN (https://cran.r-project.org) and GitHub (https://github.com) and Rdocumentation (https://www.rdocumentation.org) will help you find new packages

Data organization in spreadsheets
Data can be downloaded here: https://datacarpentry.org/ecology-workshop/setup-r-workshop.html
Lesson is online: https://datacarpentry.org/spreadsheet-ecology-lesson/
Description of common formatting problems in spreadsheets: https://datacarpentry.org/spreadsheet-ecology-lesson/02-common-mistakes/index.html

Data management with SQL for ecologists
Lesson materials: https://datacarpentry.org/sql-ecology-lesson/
SQL reference: https://datacarpentry.org/sql-ecology-lesson/reference.html
Exercises:
Let's look at some of the cleaned spreadsheets you downloaded during Setup to complete this challenge. You'll need the following three files:

- surveys.csv
- species.csv
- plots.csv Challenge

Open each of these csv files and explore them. What information is contained in each file? Specifically, if I had the following research questions:

- How has the hindfoot length and weight of *Dipodomys* species changed over time?
    - We would need to separate the species, then make a time series graph (?) for hindfoot length and overall weight
    - we'd need to make some sort trend line/estimate for weight by hindfoot since there are a lot of missing weight values to be able to talk about the relationship between hindfoot length and weight more explicitly
- What is the average weight of each species, per year?
    - I would use the aggregate function in R or something similar, to basically create subsets of data (aggregate by species to create a list according to species, then by year, and then take a mean for the weights within each year). I'm not used to thinking about functions like this logically, so my explanation is unclear. Not sure how to deal with N/As though.
- What information can I learn about *Dipodomys* species in the 2000s, over time?
    - Sex ratio in population - has it changed over time? Weight and hind foot length, and possibly habitat use if the plots are in different habitat types

What would I need to answer these questions? Which files have the data I need? What operations would I need to perform if I were doing these analyses by hand?

We need to seperate all the species and also the weight values are missing which is also important feature.

Exercise:
Import the plots.csv and species.csv tables.

Two ways to do SQL comments
-- this is a single-line comment in SQL
/* This is a multiline
comment in SQL */

A nice shortcut for queries with multiple OR statements is to use the IN verb

SQL "cheat sheet" for this lesson: https://datacarpentry.org/sql-ecology-lesson/files/sql-cheat-sheet.html

Exercise:
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;

SQL so far:

```
-- SQL is case insensitive
/* this is a
multiline comment */
SELECT *
FROM surveys
LIMIT 10;

SELECT DISTINCT species_id
FROM surveys;

SELECT DISTINCT year, species_id
FROM surveys;

SELECT year, month, day, weight, ROUND(weight/1000, 2)
FROM surveys;
```

Excerise
Produce a table listing the data for all individuals in Plot 1 that weighed more than 75 grams, telling us the date, species id code, and weight (in kg).

```
SELECT year, month, day, species_id, weight/1000 FROM surveys
WHERE (plot_id == 1) AND (weight > 75);

SELECT year, month, day, species_id, weight/1000
FROM surveys
WHERE (plot_id = 1) AND (weight > 75)

-- Filtering
SELECT *
FROM surveys
WHERE species_id = "DM";

SELECT * FROM surveys
WHERE year >= 2000;

SELECT * FROM surveys
WHERE (year >= 2000) AND (species_id = "DM");

SELECT * FROM surveys
WHERE (year >= 2000) AND ((species_id = "DM") OR (species_id = "DO") OR (species_id = "DS"));
```

Exercise:
Write a query that returns year, species_id, and weight in kg from the surveys table, sorted with the largest weights at the top.

```
SELECT year, species_id,  weight/1000
```

```
FROM surveys
ORDER BY weight desc;

SELECT year, species_id, weight/1000
FROM surveys
ORDER BY weight DESC;
```

Example SQL:

```
-- Sorting
SELECT *
FROM species
ORDER BY taxa ASC;

SELECT *
FROM species
ORDER BY taxa DESC;

SELECT *
FROM species
ORDER BY genus ASC, species ASC;
```

Exercise:

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, 2)
FROM surveys
WHERE (year = 1999)
ORDER BY species_id
ASC;

select year, month, day, species_id, ROUND(weight/ 1000, 2)
from surveys
where (year = 1999)
order by species_id, asc;


-- Sorting
SELECT *
FROM species
ORDER BY taxa ASC;

SELECT *
FROM species
ORDER BY taxa DESC;

SELECT *
FROM species
```

```
ORDER BY genus ASC, species ASC;

/* Exercise
Write a query that returns year, species_id,
and weight in kg from the surveys table,
sorted with the largest weights at the top.*/
SELECT year, species_id, weight / 1000
FROM surveys
ORDER BY weight DESC;

-- order of execution
SELECT genus, species
FROM species
WHERE taxa = "Bird"
ORDER BY species_id ASC;
```

Anonymous feedback:
Minute card feedback https://forms.gle/hTbWyEFqKYvRDc558

Aggregating and grouping data
Using the count function

```
-- returns a count of the records (rows)  in the surveys table
SELECT COUNT(*)
FROM surveys;

-- return the count of records and sum total of weights
SELECT COUNT(*), SUM(weight)
FROM surveys;

-- return the count of records, sum of weights rounded to 3 decimal places
SELECT COUNT(*), ROUND(SUM(weight)/1000, 3)
FROM surveys
```

I was getting errors until I fixed my spacing between functions
Note that we're rounding 'SUM(weight)/1000' to 3 places in this

```
-- Using MAX, MIN, and AVG functions
```

Challenge

Write a query that returns: the total weight, average weight, minimum and maximum weights for all animals caught over the duration of the survey. Can you modify it so that it outputs these values only for weights between 5 and 10?

```
SELECT SUM(weight)/1000 as total, AVG(weight)as avg, MIN(weight)as min, MAX(weight)as max
FROM surveys
WHERE weight >5 AND weight <10;

select sum(weight), avg(weight), min(weight), max(weight)
from surveys
```

where (weight > 5) and (weight < 10);

Exercise:
Write queries that return:
· How many individuals were counted in each year in total

```
select (year), count (*)
from surveys
group by (year);
SELECT year, COUNT (*)as count
FROM surveys
GROUP BY year;
```

· How many were counted each year, for each different species

```
SELECT Count(*), year, species_id
From surveys
Group By species_id
ORDER By year;
```

· The average weights of each species in each year
· Can you get the answer to both 2 and 3 in a single query?

```
SELECT year, species_id, COUNT (*) as count, AVG(weight) as avg
FROM surveys
GROUP BY species_id, year;

select year, count(*)
from surveys
group by year;

select year, species_id, count(*), avg(weight)
from surveys
group by year, species_id;
```

Aliases (giving new names to columns)

```
-- rename a calculated field
SELECT MAX(year) AS last_surveyed_year
FROM surveys;

-- alias a table name
SELECT *
FROM surveys AS surv;
```

Using the HAVING clause to filter results of aggregated functions

```
SELECT species_id, COUNT(species_id)
FROM surveys
GROUP BY species_id
HAVING  COUNT(species_id) > 10;
```

```
-- create an alias for a calculated field
-- and reference the alias in the HAVING clause
SELECT species_id, COUNT(species_id) as occurrences
FROM surveys
GROUP BY species_id
HAVING occurrences > 10;
```

Exercise:
Write a query that returns, from the species table, the number of species in each taxa, only for the taxa with more than 10 species.

```
SELECT ~~species_id~~ TAXA, count(species_id) AS occurrences
FROM species
GROUP BY taxa
HAVING occurrences >10;
```

```
SELECT taxa, COUNT(*) as n
FROM species
GROUP BY taxa
HAVING n > 10;
```

Combining data With joins

```
-- join data from the surveys and species tables
SELECT *
FROM surveys
JOIN species
ON surveys.species_id = species.species_id;

-- USING clause as an alternative to ON
-- this requires the column used to have the same name in both tables
SELECT *
FROM surveys
JOIN species
USING (species_id);

-- only join some fields or columns
SELECT surveys.year, surveys.month, surveys.day, species.genus, species.species
FROM surveys
JOIN species
ON surveys.species_id = species.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 species
JOIN surveys
USING (species_id);
```

```
SELECT species.genus, species.species, surveys.weight
FROM surveys
JOIN species
USING (species_id);

select species.genus, species.species, surveys.weight
from surveys
join species
on surveys.species_id = species.species_id;

-- different types of joins
-- this returns 34786 rows
SELECT COUNT(*)
FROM surveys
JOIN species
USING (species_id);

-- totale number in surveys
-- returns 35549
SELECT COUNT(*) FROM surveys;
```

CHALLENGE:
Re-write the original query to keep all the entries present in the surveys table. How many records are returned by this query?

```
SELECT COUNT (*)
FROM surveys
LEFT JOIN species
USING (species_id)
```

records returned: 35549

```
SELECT COUNT (*)
FROM surveys
LEFT JOIN species
USING (species_id);
```

35549

```
select species.genus, species.species, surveys.weight
from surveys
left join species
using(species_id);


-- Joins with aggregations
SELECT plots.plot_type, AVG(surveys.weight)
FROM surveys
JOIN plots
ON surveys.plot_id = plots.plot_id
```

GROUP BY plots.plot_type;

Challenge
Write a query that returns the number of animals caught of each genus in each plot. Order the results by plot number (ascending) and by descending number of individuals in each plot.

Solution:
SELECT surveys.plot_id, species.genus, COUNT(*) as n
FROM surveys
JOIN species
ON surveys.species_id = species.species_id
GROUP BY species.genus, surveys.plot_id
ORDER BY surveys.plot_id ASC, n DESC;

Data Analysis and Visualization in R for Ecologists
R and RStudio lesson page: https://datacarpentry.org/R-ecology-lesson/index.html

Intro to R

Challenge 1:
What are the values after each statement in the following?
mass <- 47.5 # mass? 47.5
age <- 122 # age? 122
mass <- mass * 2.0 # mass? 95
age <- age - 20 # age? 102
mass_index <- mass/age # mass_index? 0.931...
(I got the same as above)
Ditto!
same

Challenge 2:
What type of data are each of the following vectors?
num_char <- c(1, 2, 3, "a") chara
num_logical <- c(1, 2, 3, TRUE) num
char_logical <- c("a", "b", "c", TRUE) chara
tricky <- c(1, 2, 3, "4") chara
same as above
same!

Challenge 3:
heights <- c(63, 69, 60, 65, NA, 68, 61, 70, 61, 59, 64, 69, 63, 63, NA, 72, 65, 64, 70, 63, 65)

1. Using this vector of heights in inches, create a new vector, *heights_no_na*, with the NAs removed.
2. Use the function median() to calculate the median of the *heights* vector.

3. Use R to figure out how many people in the set are taller than 67 inches.

heights_na.rm <- heights[!is.na(heights)]
median(heights_na.rm) returns: median: 64
heights_na.rm[heights_na.rm > 67] returns (number greater than 67 = 6)

```
heights_no_na <- [!is.na(heights)]
heights_med <- median(heights)
heights_subset <- [heights > 67]
```

Day 2 anonymous feedback: https://forms.gle/htGx9n659ffuneP39

NM EPSCOR demographic survey (only if you haven't already filled it out):
https://www.surveymonkey.com/r/DQPVFWD

Getting started by downloading data.
1. load tidyverse

- library(tidyverse)

2. Make sure you have data_raw in your project working directory:

- dir.create('data_raw')

3. Download the data into that directory with:

- download.file(url = "https://ndownloader.figshare.com/files/2292169",
- destfile = "data_raw/portal_data_joined.csv")

4. Load the data frame:

- surveys <- read_csv("data_raw/portal_data_joined.csv")

Starting with Data
Welcome back challenge:
Can you figure out why "four" > "five" returns TRUE?
Is it an alphabetical order thing? Five comes before "four" alphabetically so it is earlier in the list, and so it is smaller than "five"
It's saying the characters, not the numbers, are alphabetical
Are letters assigned a number value? Like 1-27? If so, four would be greater

That's my best guess as well ^
R compares their alphabatical order in < and > strings

Based on the output of str(surveys), can you answer the following questions?

- What is the class of the object surveys?
- How many rows and how many columns are in this object?

class: tibble

34,786 rows x 13 cols

Challenge

1. Create a data.frame (surveys_200) containing only the data in row 200 of the surveys dataset.

surveys_200 <- data.frame(surveys[200,])
surveys_200 <- as.data.frame(surveys[200,])

1. Notice how nrow() gave you the number of rows in a data.frame?
    - Use that number to pull out just that last row in the data frame.
    - Compare that with what you see as the last row using tail() to make sure it’s meeting expectations.
    - Pull out that last row using nrow() instead of the row number.
    - surveys[nrow(surveys),]
    - Create a new data frame (surveys_last) from that last row.
    - surveys_last <- data.frame(surveys[nrow(surveys),])
2. Use nrow() to extract the row that is in the middle of the data frame. Store the content of this row in an object named surveys_middle.
3. Combine nrow() with the - notation above to reproduce the behavior of head(surveys), keeping just the first through 6th rows of the surveys dataset.

surveys[200,] // surveys_200 <- surveys[200,]

nrow(surveys)
surveys[34786,]
tail(surveys)
surveys[nrow(surveys),]
surveys_last <- surveys[nrow(surveys),]
surveys_last

# using the reverse indexing (Jon's term) to get the head of the data frame:
# the '-' is equivalent to "all but"
# we are selecting all but the 7th row through the 34786 row,
# which is the value of our 'n_row; variable
surveys_head <- surveys[-(7:n_rows),]
surveys_head

1. Change the columns taxa and genus in the surveys data frame into a factor.

surveys$taxa <- factor(surveys$taxa)ditto this line, but with factor in front of (surveys$genus)
surveys$genus <-(surveys$genus)
is there any reason to not do this:
surveys$taxa<-as.factor(surveys$taxa)
surveys$genus<-as.factor(surveys$genus)
?

1. Using the functions you learned before, can you find out…

- How many rabbits were observed?
- How many different genera are in the genus column?
- str(surveys$genus) retuns 26 levels
- 26 genera

Rename “F” and “M” to “female” and “male” respectively. (In the surveys$sex vector)
after doing the previous function as.charachter(sex) the M and F were already changed to Male and Female

levels(surveys$sex)<- c("Female", "Male")

In RStudio the keyboard shortcut for the pipe operator %>% is Ctrl + Shift + M (Windows) or Cmd + Shift + M (Mac).

Using pipes, subset the surveys data to include animals collected before 1995 and retain only the columns year, sex, and weight.

Using pipes, subset the surveys data to include animals collected before 1995 and retain only the columns year, sex, and weight.

```
surveys %>%
  filter(year < 1995) %>%
  select(year, sex, weight)

surveys_pipe <- surveys %>%
  filter(year < 1995) %>%
  select(year, sex, weight)
```

Challenge:|
Create a new data frame from the surveys data that meets the following criteria: contains only the species_id column and a new column called hindfoot_cm containing the hindfoot_length values (currently in mm) converted to centimeters. In this hindfoot_cm column, there are no NAs and all values are less than 3.

Hint: think about how the commands should be ordered to produce this data frame!

```
surveys_new <-
  surveys %>%
  mutate( hindfoot_cm = hindfoot_length/10) %>%
  select(species_id, hindfoot_cm) %>%
  filter(!is.na(hindfoot_cm), hindfoot_cm < 3) %>%
  head()

surveys_hindfoot_cm <- surveys %>%
  mutate(hindfoot_cm = hindfoot_length/10) %>%
  filter(hindfoot_cm < 3 & !is.na(hindfoot_cm)) %>%
  dplyr::select(c(species_id, hindfoot_cm))
```

Challenge

1. How many animals were caught in each plot_type surveyed?

```
surveys %>%
    count(plot_id)
```

2. Use group_by() and summarize() to find the mean, min, and max hindfoot length for each species (using species_id). Also add the number of observations (hint: see ?n).

```
surveys %>%
  group_by(species_id, hindfoot_length) %>%
  summarize(mean_hind = mean(hindfoot_length),
            min_hind = min(hindfoot_length),
            max_hind = max(hindfoot_length)
            count = n())
```

```
surveys %>%
  filter(!is.na(hindfoot_length)) %>%
  group_by(species_id) %>%
  summarise(hf_mean = mean(hindfoot_length),
            hf_min = min(hindfoot_length),
            hf_max = max(hindfoot_length),
            hf_count = n())
```

3. What was the heaviest animal measured in each year? Return the columns year, genus, species_id, and weight.

```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(year) %>%
  summarise(heaviest_idv = max(weight),
            genus = first(genus),
            species_id = first(species_id))
```

```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(year) %>%
  filter(weight == max(weight)) %>%
  select(year, genus, species, weight) %>%
  arrange(year)
```

USE THE COMMANDS BELOW TO CREATE THE suveys_completed DATASET:
(You will need the surveys_complete.csv file to continue with the plotting lesson)

```
surveys_complete <- surveys %>%
```

- filter(!is.na(weight),
- !is.na(hindfoot_length),
- !is.na(sex))

```
## Extract the most common species_id
```

species_counts <- surveys_complete %>%

- count(species_id) %>%
- filter(n >= 50)

Only keep the most common species
surveys_complete <- surveys_complete %>%

- filter(species_id %in% species_counts$species_id)

```
write_csv(surveys_complete, file = "data/surveys_complete.csv")
```