Introduction to R and Descriptive Statistics
SHSU Innovation Summit – September 2023
Dr. Amber Ulseth, Department of Biological Sciences

Objectives

1) Become familiar with R
2) Learn how to enter data and run descriptive statistics on a data set.

Summary

R is an open source (**free to everyone!**) statistical program based on the 'S-PLUS' programming language. There is a large community of scientists (and growing!) that use R. Because of the popularity of R, the web resources for R are vast and very thorough (see http://www.r-project.org/). Additionally, statisticians have created numerous **"packages"** for R that can allow users to accomplish almost any statistical analysis. This software is available for both Microsoft Windows, Apple OSX, and Linux operating systems.

R users primarily analyze data using the command line interface (CLI). The CLI code allows for more direct control of calculations and greater flexibility, and we suggest that any student with a desire to use R in the future should become comfortable with the CLI coding. This tutorial will give you a basic understanding of statistical modeling in R that you can use throughout your scientific career.

This overview introduces you to R by having you enter a data set and run descriptive statistics on these data. The data you will be entering are length measurements (mm) of daphnia (small planktonic animals that inhabit aquatic ecosystems) collected in June and September from a lake at the University of Notre Dame Environmental Research Center.

R and Rstudio

Rstudio is an interface to run the statistical program R. You will need to download both R and Rstudio. Follow the directions to do so here: https://posit.co/download/rstudio-desktop/

- Always cite 'R' not Rstudio (use the code: **citation()** in your R console)

If needed, there is also a web or cloud version of R and Rstudio. See posit.cloud for more information.

Resources

There are MANY resources to help you get started to trouble shooting code in R. A couple that I find helpful to get you started:

Introduction course from Carpentry: https://datacarpentry.org/R-ecology-lesson/01-intro-to-r.html

Trouble shooting code: Google has always been my go-to as there are many R help forums, but I now most often use ChatGPT: https://chat.openai.com/

Procedure

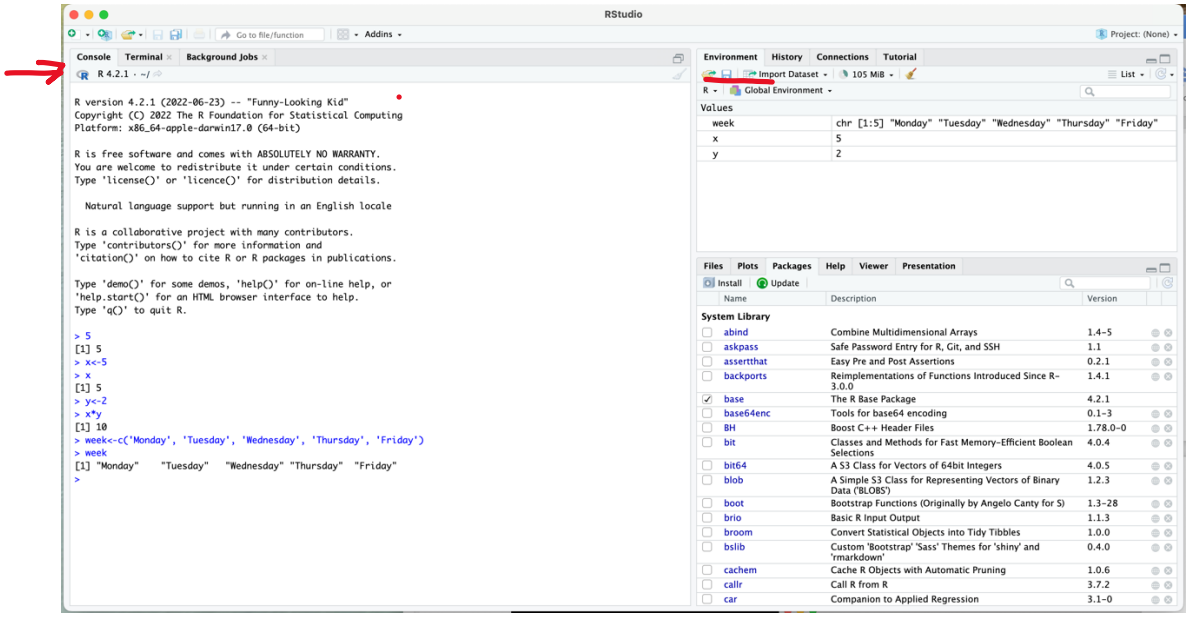
Getting familiar with R

Open RStudio (either on your computer, or via **posit.cloud**)

Initially, the window that opens immediately is the "R Console". This is where code is entered (in blue) and results are shown (in black). You may run commands by typing in code and then hitting ENTER.

For example, type the following in the console, hitting enter after every line:

```
5
x<-5
y<-2
x*y
week<-c('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

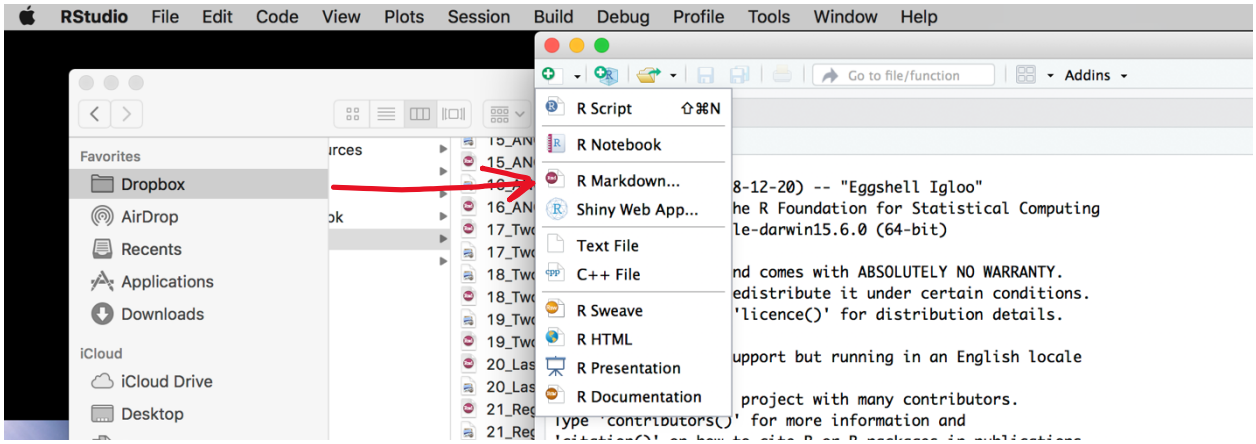


The beauty of coding is to not reinvent our analyses or code every time we want to analyze our data. Therefore, like a word document for any paper you might be writing, we will set up a '**R Script**' code that may be saved and used at another time.

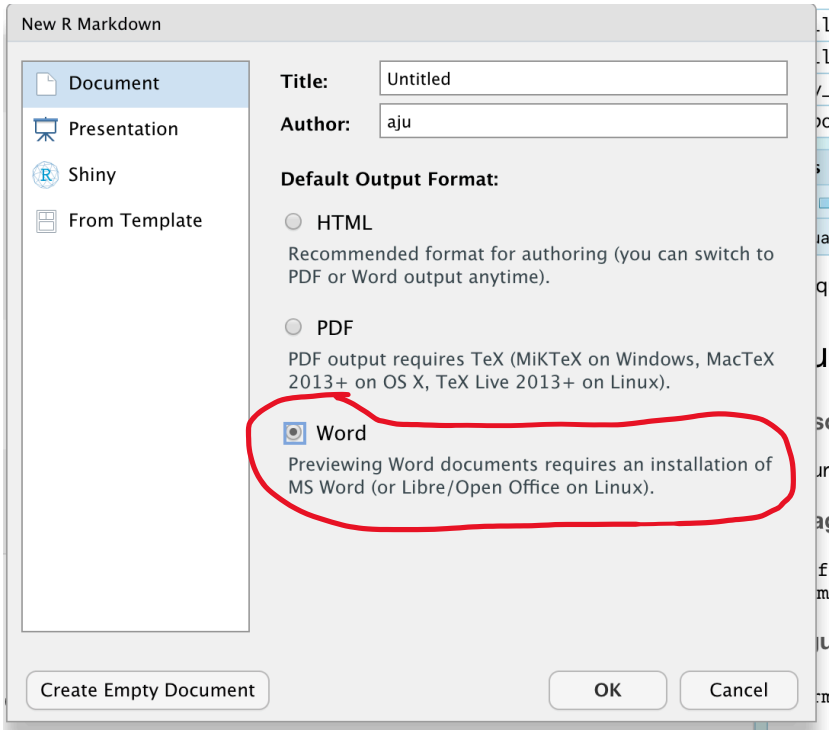
Click "File" and "R Markdown"

(Note: I will show you R Markdown here, but another option is to use Quarto - https://quarto.org/)

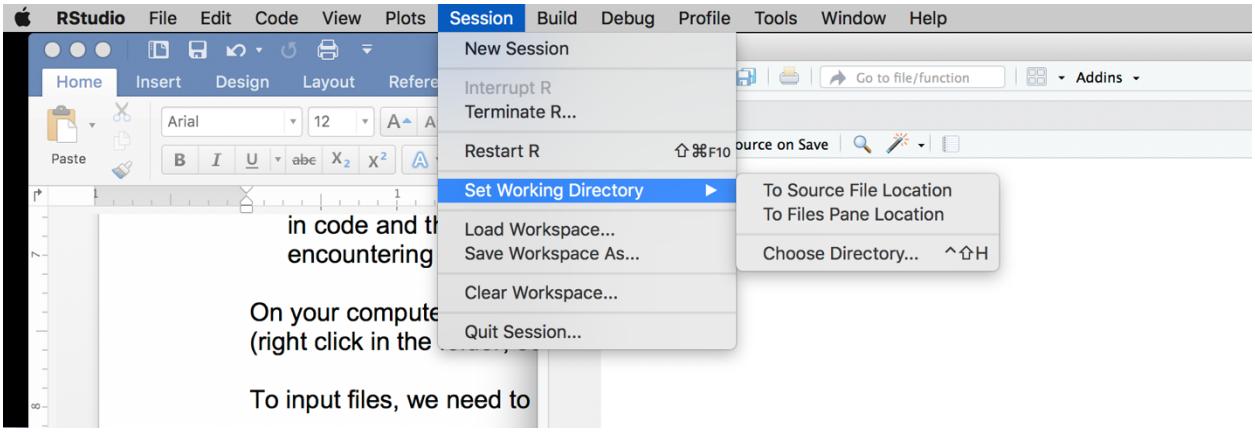
The window that opens now is called the "R markdown Document" where you can type and edit your code and then save it for later use. You can also make notes inbetween 'chunks' of R code. Making notes about what your code does will make life much easier when you return to your code later. ***These notes regarding your code are called 'annotating your code'. Do it. You will thank yourself later***.



Rmarkdown allows to 'knit' our code and output into one document. It is easiest to knit final document into a 'word' document as you can edit it later if needed. You may want to title your file – but this does NOT save it. We will have to save and name the Rmarkdown file in the next step. I suggest implementing organization to your files. It is easiest if your code is saved within the same folder on your computer.



<u>To input files into R, we need to change the directory so R can find your files:</u>



You have options from Session -> :

- Select 'Choose Directory' which will then map the directory to your location of choice
- Select 'To Source File Location' – this will map your directly to where your Rmarkdown file is saved. I use this approach often for organization – so my script and data files are then found in the same location on my computer.

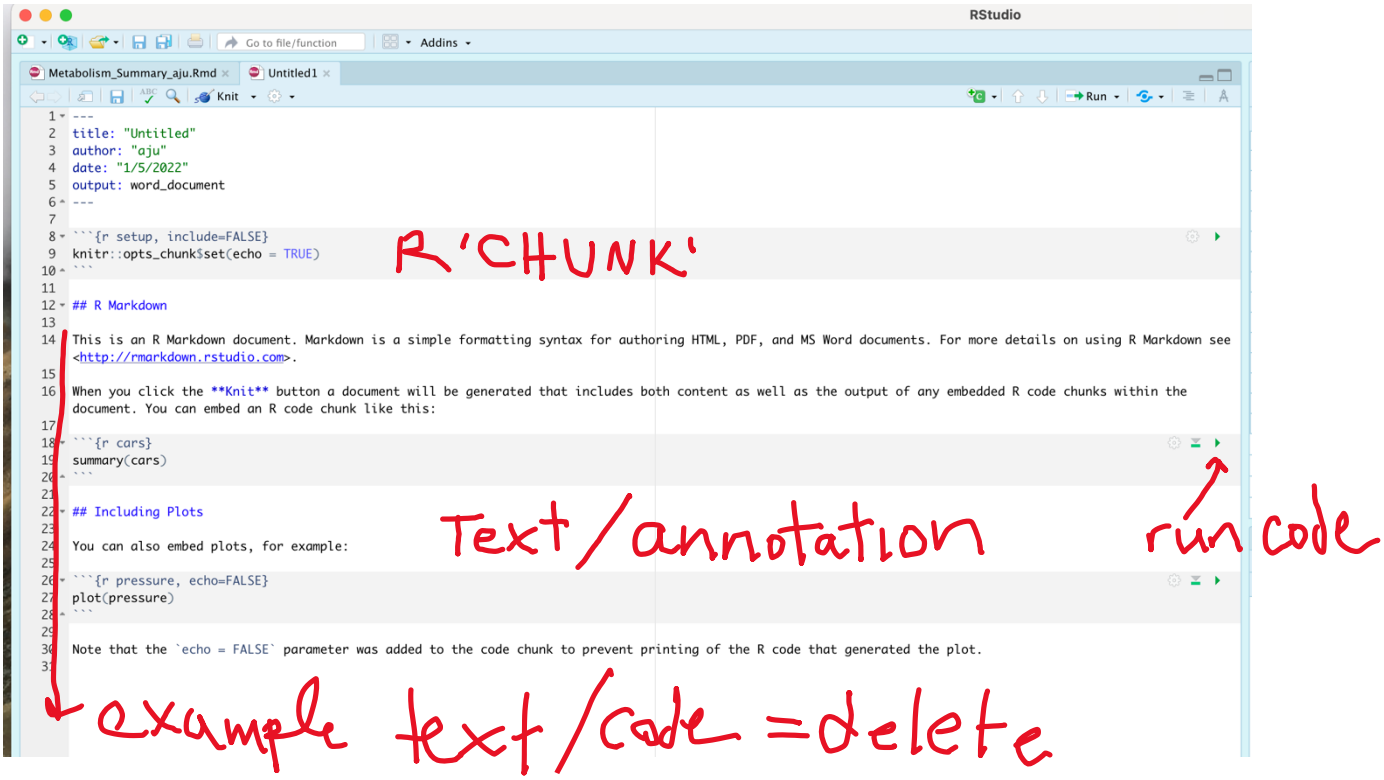
OR code the location of your directory within your 'R script' file. To do this – you will use the function 'setwd()'. For example:

```
setwd("~/Dropbox/Harvey Stream Response Team/ForMetabolism")
```

Select the folder where you will place all your files for use in R

These commands specified your "R Files" folder as what R refers to as a working directory. For simplicity, R reads data and saves scripts together in one folder. You must **set your working directory at the beginning of each session** to be sure R knows where to look for your files.

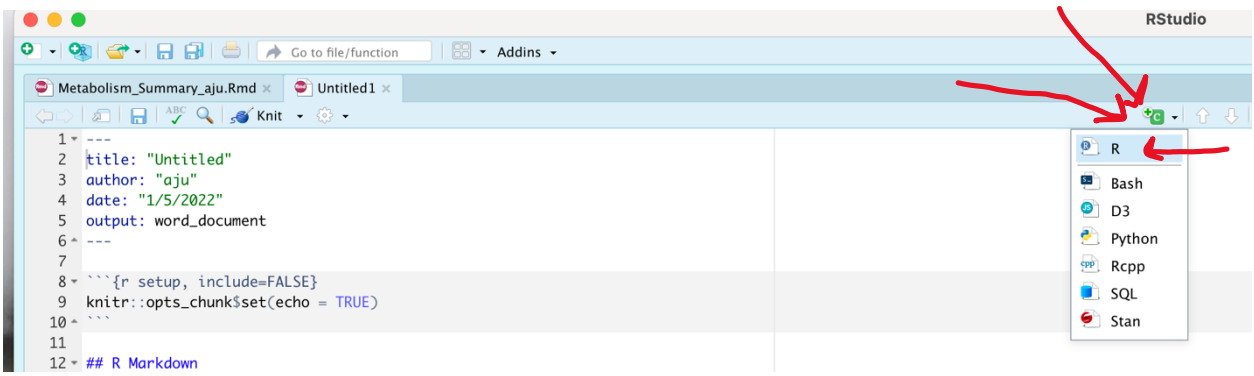
Some Coding Basics:



Rmarkdown allows for you to both annotate/type text along with 'chunks' of code. The 'chunks' in the screenshot above are in grey where the text is in white.

When you first open an Rmarkdown file – R will populate it with example text and code. We can delete these starting at ## R Markdown and below.

To insert a Rscript 'chunk', go to the +C button (on top) to insert Rscript



Within your R Script chunk, we can now code. Try the following:

Scalar object, Vectors and Data frames in R Console

R organizes and handles data in objects called vectors and data frames.

A scalar object is a 'dataframe' that only holds one object at a time. This is the most basic type of data in R.

Within your R script type the following (We can also annotate within our chunk by including a hashtag - # - in front of 'a scalar object'. This indicates to R that the text is not code and not to be 'ran', see how the text is green following '#'):

```
# a scalar object
x0 <- 5 # to code = in R, use '<-' (it is like an arrow pointing from the object of interest to the name of that object)
x0 #this is then how you 'ask for' or 'use' the object x0
```

This will create an object called 'x0' that will have a value of '5'.

A vector is a sequence of data elements and is the simplest data structure.

Example of vector objects:

```
# vector objects
x<-c(1:100) #creates a vector of 1 to 100 by increments of '1'
countries<-c("Uganda","Netherlands","USA","Japan") #creates a vector of country names
```

Other commands to create simple vectors:

```
rep(c(1:5),times=2)
seq(from=1,to=20,by=0.5)
rep(2,times=10)
rep(c(1:5),times=2)
rep(c(1:5),each=2)
```

Try these out to see the outcome.

What if we want to select parts of objects or parts of a vector (or dataframe – which we get to below)?

```
# ask for objects and parts of objects
```

```
x #inputting the name of the object (not what we designate as 'x' above)
y
y[y<0] #selecting all of the values of our 'y' object that is less than zero
countries
countries[2] #we want the second value from our 'countries' vector
countries[1:3] #1-3 values of the 'countries' vector
countries[c(1,4)]
```

We may also perform computations with our vectors. For example, using some of the objects we created above:

```
# vector-based computations
x
x0
y

x*x0 #multiplication of x and x0, * is multiplication in R
x*y


x*z # -> the shorter vector is recycled but a fit for checking length agreement is done.
```

R reads objects in order that you run the code. Therefore, you can overwrite objects in R by simply using the same name. So be aware of the order you are entering code and the names of such code.

At this stage, we will deal with **dataframes**. A dataframe is a table of data arranged in rows and columns, where each row contains the measurements from a single observation and each column contains all the values from a single variable (see Table below).

In R, the first row of a dataframe (or a vector) can be reserved for the variable names. A few things about variable names:

Month	Length
June	1.16
June	1.13
June	1.13
June	1.15
June	1.09
June	0.85
June	1.31
June	1.29
June	1.38
June	1.4
June	1.24
June	1.45
June	1.04
June	1.1
June	1.48
June	1.12

1. **Variable names should be entered without spaces**.
2. **R is case sensitive**, i.e., capitalized vs lower case does matter!
3. **Do not start variable or data file names with numbers. Always start with a letter.**

There are two main ways you can create vectors dataframes in R:

(1) entering the data into a spreadsheet program (Excel) and having R read this data, or
(2) creating the dataframe directly in R.

To create a dataframe to be imported into R, scientists often use excel for raw data entry.
Statisticians often find it easiest to enter and organize data in a spreadsheet program like Excel and then analyze these data in a

separate program. Although it may seem like extra work, this is often the best way to work with large datasets. Other files such as .txt and .csv files also work.

Table 1 in the excel file 'daphnia.xlsx'. **Save data files to the same folder as your Rmarkdown file.**

Note about entering data for use in R: *Misspellings and errant keystrokes can cause serious headaches in R. It is very important to check all your data to make sure there are no mistakes. As mentioned above,* ***R is case sensitive*** *so be sure you check your case! For column names, do not use spaces or special characters. R also reads spaces as missing data.*

Browse to your working directory (R Files)

R can read many file types for this introduction to R, we will use Excel files.

To read in a .xlsx file –:

We will have to install a new 'package' to read an excel file directly. We will use the package 'readxl'.

Note about 'packages': *Some functions (e.g., to calculate the mean of a data set or the standard deviation – similar to what we find in excel) are built into basic R. And some functions have been developed more recently as R is an open-source code.*

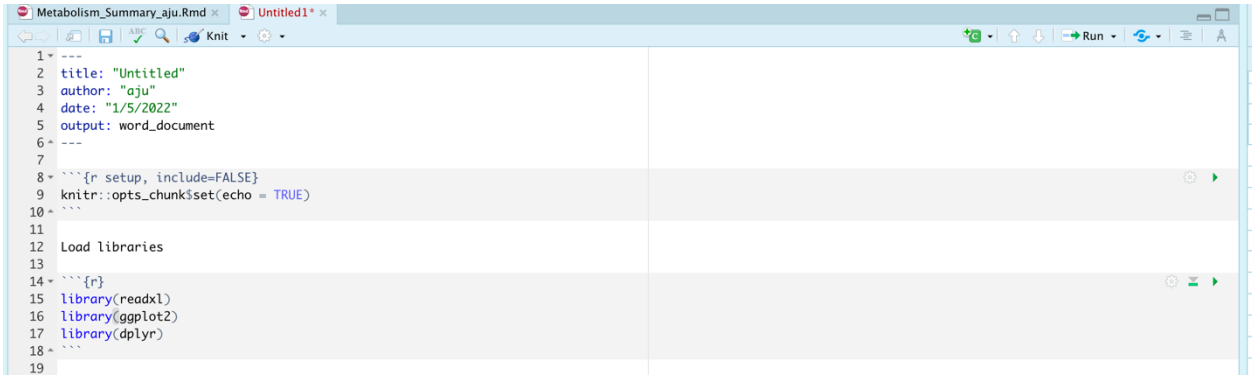
We have 2 options for installing a new package:

Go to 'Tools' -> 'install packages'. Search for 'readxl' and then install.

Once a package is installed onto your computer, you will NOT need to install again. However, when you first open R, you will need to 'load' each package. To do so, you use the following code:

```
library('package_name') #where 'package_name' is the name of the package you wish to load into your current R session.
```

As will be using more and more packages throughout your venture in R, I find it easiest to keep a running list of all those packages to 'load' each time I run my code. I often include a Rscript 'chunk' at the top of my code where I can then copy and paste for each new Rmarkdown file I make:

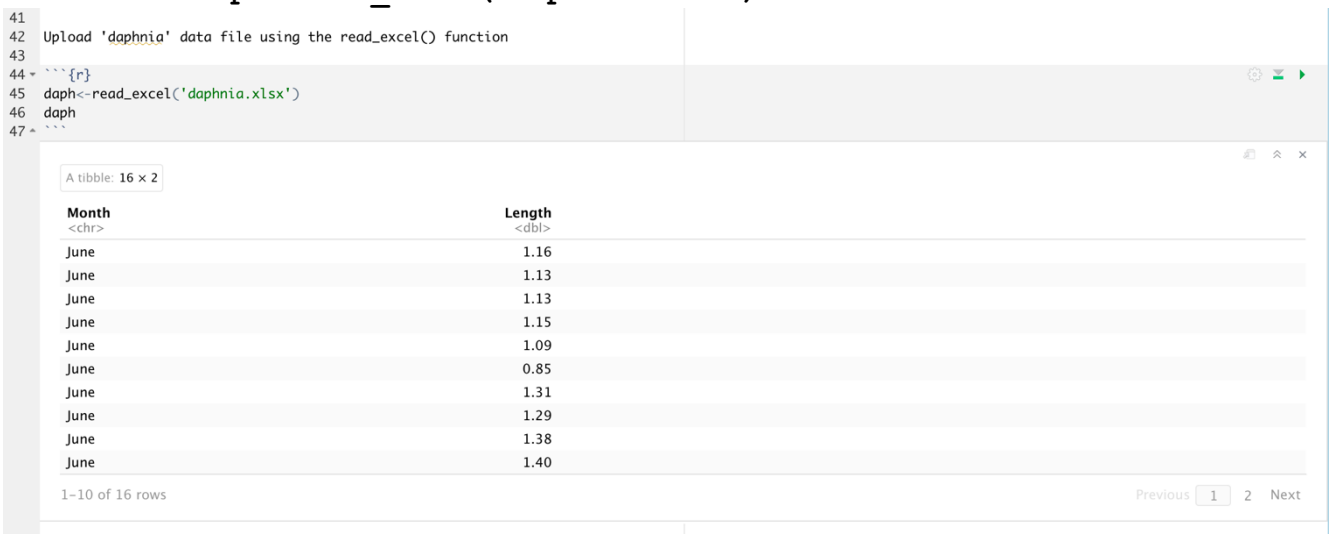


For readxl:

library(readxl)

From the readxl package, you will use the 'read_excel' function as follows:

daph<-read_excel('daphnia.xlsx')



For 'Rstudio.cloud' – we will 'upload' the data directly to the project before using the code: daph<-read_excel('daphnia.xlsx')

Type **names(daph)**

This command shows you a list of the variable names. This is a very useful command if you ever forget what you labeled a column.

Type **str(daph)**

This command shows the structure of the data frame. I use this function most often…

Type **summary(daph)**

This command summarizes your data with descriptive statistics. "Summary" calculates

the mean and median as well as minimum and maximum values. The 1st and 3rd quantile values give you estimates of values where 25% of the data lie above or below these points.

Other functions to explore the data frame include:

```
ncol(daph)
nrow(daph)
names(daph)
str(daph)
head(daph) # for a dataframe with many rows
tail(daph)
daph$Month #note the $ symbol, here I want to see the 'Month' column from the 'daph' data frame.


#Summary statistics
mean(daph$Length) #calculates the mean length
median(daph$Length) #calculates the median

summary(daph) #summary stats
```

Once we have our data into R and have done some basic summary statistics, we can amend this data. Say we want to add data from an additional month:

We could pull up our original excel file, amend the file and then upload again. But here, is another way to amend the file from within R.

First enter the data you want to add by making a list (vector)

```
Length2<-c(1.17, 1.45, 1.43, 1.31, 1.12, 1.11, 1.29, 1.31, 1.42,
1.48, 1.26, 1.45, 1.06, 1.12, 1.51, 1.13).
```

Then, we want to make a list the same length as Length2 designating the month of September:

```
Month<-rep('September', length(Length2))
```

Now make September into a data frame

```
daph2<-data.frame(Month=Month, Length=Length2)
```

And combine 'daph' with 'daph2' using the function 'rbind', which binds the rows based on matching column names.

```
daph<-rbind(daph, daph2)
```

Type **daph** into the Script Window and click the "Submit" button

You should now see your new dataframe, with September data added, in the Output Window. Of course, it is often easier to use the original spreadsheet program to fix your dataframe.

Now look at the summary statistics and the structure of the dataframe

```
summary(daph)
str(daph)
```

The summary statistics given for the lengths is a summary of all the length values. You can gather summary information for separate months in the next step.

```
tapply(daph$Length, daph$Month, mean)
```

This command now gives you the mean lengths of the daphnia in June and September. There are also options for calculating other descriptive statistics of the data from each month.

Calculate the median, standard deviation, variance (**var**), maximum value (**max**), and minimum value (**min**) for daphnia lengths in June and September.

```
tapply(daph$Length, daph$Month, median)
apply(daph$Length, daph$Month, sd)
tapply(daph$Length, daph$Month, var)
tapply(daph$Length, daph$Month, max)
tapply(daph$Length, daph$Month, min)
```

And another way to summarize data, using the 'summarySE' function from the 'Rmisc' package, which gives you the n, mean, standard deviation, standard error, and the 95% confidence interval

install.packages('Rmisc') #once installed, you will not need to run this line of code again

library(Rmisc) #include this in your 'list' of packages. While we don't need to install every time, we need to 'load' the library by using the library(function)

```
d.sum <- summarySE(daph, measurevar="Length",
groupvars=c("Month"))
```

As we get into more and more complex data, I also prefer summary statistics using the group_by() function from the package 'dplyr'

install.packages('dplyr')

```
library(dplyr)

group_by(daph, Month) %>% #%>% is a pipe code where the data to the left are fed into the functions on the right (following the 'pipe')
summarise(
    count=n(),
    mean=mean(Length, na.rm=TRUE) #na.rm=TRUE if we have any missing data)
    sd=sd(Length, na.rm=TRUE)) #note code could also be written in 1 line, I use spaces and place each function on a different line for organization
```

Creating graphs in R

Within R, there are a variety of approaches that are available to make plots. The most common approaches are with the regular R plotting options, and one of my favorites – 'ggplot'.

We want to make a plot illustrating the lengths of Daphnia by month

1. 'Regular' plotting options to make a boxplot of daphnia length ~ month

```
boxplot(daph$Length ~ daph$Month)
```

2. ggplot to make 'point' plot with standard error bars. This builds upon the summary 'd.sum' that you calculated above.

We first have to install 'ggplot2' and then load it.

- Install ggplot2, as you did with 'readxl' and then execute the code:

```
install.packages('ggplot2')
library(ggplot2)
```

#to make plot in ggplot:

```
ggplot(d.sum, aes(x=Month, y=Length)) +
  geom_point(size=3) +
  geom_errorbar(aes(ymin=Length-se, ymax=Length+se), width=.1)
```

Summary of commands

`dataset<-read.csv(file='filename.csv')` - converts a .csv file to an R object

`daph<-read_excel('daphnia.xlsx'`)- converts an excel file to an R object from the 'readxl' package

`library(packagename)` - opens a previously downloaded R package

`help(function)` - opens the help file for a specific function

`?(function)` – as above

`str(dataset)` – allows you to see the structure of the dataset, including variable names and variable types (i.e., numerical, character, etc…)

`attach(dataset)` - attaches the dataframe to simplify calling variables

`summary(dataset)` - gives summary statistics for your dataframe variables

`tapply(variable,category,value)` - calculates values (mean, variance, etc.) for one variable, arranged by a categorical variable (e.g., month)

`subset<-subset(dataset,category=="name")` - creates a subset of the original dataframe, where only the categories labeled by the chosen name are selected

`hist(variable)` - creates a default histogram for the variable of the active dataset (see the help file for more histogram options)