BI 306 Lab

Fall 2021

R Tutorial: Part 1

Welcome to the first lab for Biology of Global Change! We are going to be doings some analysis and visualization of data collected during labs this semester so we will have a brief 2 lab tutorial in the programming language R. R is quickly becoming one of the most popular tools for data analysis and visualization in a lot of biological and ecological fields. R is a free programming language that has a lot of powerful tools to help researchers perform tasks from simple summary statistics to complicated ecological modeling and mapping. We will use this as an introduction to using R, but this course is by no means a coding or statistic course. The expectation here is to learn the goals from each tutorial and participate in the labs so that you will be able to use these skill for analyses in this lab.

Goals for this tutorial

- Install R and R studio
- View the structure of data sets
- Perform basic summary statistics on data (`summarySE()` and `tidyverse`)
- Make R markdown table to display data

Before you begin, you will need to download the following:

- R software **here** for your computer
- RStudio **here**

Hints for starting with R coding

- R is case sensitive so if something is displayed as `iris` and you type `Iris`, R will look for an object named `Iris` instead of the desired `iris`. Be careful with case and general typos because these are some of the most common causes of error messages to start with.
- **Google is your best friend for figuring out how to do things!** If R produces an error message, you can easily search it and will likely find someone with the same (or similar) error and a solution to fix it. Additionally, it is a great way to figure out how to do specific tasks in R. If you ask any seasoned programmer, they will happily tell you how their productivity is generally hindered without internet because everyone hits problems they need to Google!
- **Keep your head up and keep trying!** This is not something that you will learn overnight so keep plugging away and keep trying. Work with your classmates who may already have stats or programming experience who may be able to help you. If you are still stuck, reach out to your TA/TF or instructor for help. We are all happy to help support you.

Quick notes on this document:

- Blue text can be clicked on to access the associated link (for example, you can click **this** to link to the latest IPCC report)

- Text with grey shaded background represents R objects or commands in line with main text like this: `2 + 2`
- Some code chunks are missing code where you can apply what you have learned. Those will have hashed (**#**) out comments in them with instructions to help you write the appropriate code.

Getting started with R

To begin, we will be working with one of the built-in datasets in R called **iris** that contains the sepal and petal widths and lengths of different iris species (you can read more about this dataset here).

```
## Load the iris dataset that is included in R
data("iris")
```

After loading the data, we can see what is actually included in the data using the **head()** function.

```
head(iris)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

The **head()** function will display the top six rows of data (you can modify this to show, for example, the top 3 rows like this **head(iris, 3)**). This is a nice way to get a sense of what your data look like. We can now see that the **iris** data set contains four columns containing data for sepal length, sepal width, petal length, petal width, and species. These columns contain different types of data that we can understand a bit better by using the **str()** function to view the *structure* of the data.

```
str(iris)
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 
1 1 1 1 ...
```

From this function, we can now see that the **iris** data set is a **data.frame** object, it contains 150 observations (or rows of data), and 5 variables (columns). Additionally, the function displays the type of data in each column (identified after the **$**) and first few observations. We see that the first four columns all contain numeric (**num**) values while the last column (**Species**) contains factors with three different levels (these levels are the different types of species within that column).

In this document, we are only displaying the first few lines of the data for simplicity, however, there are many situations in which we may want to view the entire data set in the R console. The easiest way to do that in the console is just to type the name of your data frame in the console and press enter/return. So if we wanted to view the full **iris** data frame, we would simply type that into the console and press enter/return. Alternatively, you can type it into the R Markdown script and press command/control and return/enter at the same time and the data will be printed in the console.

Try doing this below:

```
    ## Type the name of the data frame below and have it run to print
## the full data frame in the console window:
```

Another option for viewing the data frame is through the function **`View()`** and this option is a bit more interactive and really helpful for organizing columns by values to explore the data a bit more. By typing the data frame name we wish to view within the parentheses of this function, R will open a tab beside this current R markdown script with the data frame name. The tab will display the data frame and from this tab we are able to scroll through all the rows of data or even click on a column name to order the rows by that column from lowest to highest or highest to lowest. While this is a helpful function while working on data in R, it will not print in the R markdown. If we want to print an entire data frame in the R markdown output, we should use the above method instead of **`View()`**.

Give this a try:

```
    ## Use the View() function on the iris data
```

Downloading and loading packages

R comes with a lot of functions already included that you can use for your analyses, however, there are so many powerful and helpful packages out there that include additional functions that you will likely want to use. For example, we will want to calculate some simple summary statistics on the **`iris`** data and the package **`Rmisc`** contains a helpful function **`summarySE()`** that can accomplish this. In order to use this function, you will need to download this package. Most R packages can be easily downloaded in RStudio from the packages tab in the lower right panel or using the command **`install.packages()`**. To practice, go ahead and download the **`Rmisc`** package using the command below.

```
    ## Download the Rmisc package from CRAN
install.packages("Rmisc")
```

Once you have installed the package (*you only need to do this once or if you update your R version*), you need to load that package using the function **`library()`** or **`source()`** before you can use any of the included functions.

```
    ## Load the Rmisc package before using it
library(Rmisc)
```

Great! You now have the **`Rmisc`** package installed and loaded in your current RStudio session. While you only need to install a package once, you will need to load the package every time you begin a new session and want to use those functions.

Helpful tip: If you are using a package for the first time or are looking for helpful information about the functions within a package, you can view the documentation for that package by running the following code with the name of the package you are interested in.

```
    ## View the documentation associated with the Rmisc package
help(package = "Rmisc")
```

Simple summary statistics

Once you know what the structure **`str()`** of your data looks like, it is also good to look at some basic summary statistics. There are many different ways to calculate summary statistics in R, but here we will explore two different methods using different R packages.

In this course, you will be expected to be able to calculate, interpret, and report the following variables:

- **Sample size** (N): number of observations/individuals included in a study
- **Mean** ($\overline{X}$): the sum (Σ_x) of the values divided by the number of values (N)

$$\overline{X} = \frac{\Sigma_x}{N}$$

- **Standard deviation** (SD; σ): measures the dispersion of a dataset relative to its mean ($\overline{X}$)

$$\sigma = \sqrt{\frac{\Sigma(X - \overline{X})^2}{N - 1}}$$

- **Standard error** (SE): measured how much discrepancy there is likely to be in a sample's mean compared to the population mean ($\overline{X}$). Standard error (SE) builds off the standard deviation (σ).

$$SE = \frac{\sigma}{\sqrt{N}}$$

- **R^2** : *coefficient of determination*; the proportion of the variation in the dependent variable that is predictable from the independent variable(s). The closer R^2 is to 1, the more strongly related two variables are.

Method 1: summarySE()

The first method will use the function **`summarySE()`** within the R package **`Rmisc`** installed and loaded above. This function can calculate the sample size, mean, standard deviation, standard error, and confidence intervals for your data (you can read details on the function **here**).

This function requires the user to input some information so the function knows how to calculate the summary statistics so we will first break down all the arguments that need to (or can be) specified. Here is what the function looks like: **`summarySE(data = NULL, measurevar, groupvars = NULL, na.rm = FALSE, conf.interval = 0.95)`**

argument	definition
data	a data frame
measurevar	the name of a column that contains the variable to be summariezed
groupvars	a vector containing names of columns that contain grouping variables
na.rm	whether to ignore NA's in your data (TRue = NA's ignored)
conf.interval*	percent range of the confidence interval (default is 95%) * *We will not be using CI in this lab, but it is good to know!*

If we want to measure the summary statistics of the measured **Sepal.Length** by **Species** in the **iris** dataset, we would fill in the following into the function:

```
summarySE(data = iris, measurevar = "Sepal.Length", groupvars = "Species",
 na.rm = TRUE)
   ##      Species  N Sepal.Length        sd         se        ci
## 1     setosa 50        5.006 0.3524897 0.04984957 0.1001765
## 2 versicolor 50        5.936 0.5161711 0.07299762 0.1466942
## 3  virginica 50        6.588 0.6358796 0.08992695 0.1807150
```

This output now shows grouping variable (**groupvars**) **Species** as the first column, followed by the number of observations within each species (sample size; **N**), the calculated mean length (notice that the mean column is given the same name as the measured variable – **measurevar = "Sepal.Length"**), standard deviation (**sd**) of each species, standard error (**se**), and the 95% confidence interval (**ci**).

Now that we can calculate the summary statistics of the **Sepal.Length** per **Species**, we should assign this function output so that we can come back to it later for things such as plotting it as a figure. To assign this, we simply need to give the output from the function a name. R object names should not contain spaces and are case sensitive. It is best to use underscores (_) to separate words in object naming if you must (instead of **-** or **.**). Here, we will name our summary results **sepal_length_sum**. When assigning objects to names, we first provide the name (**sepal_length_sum**), followed by the left assignment (**<-**), and then what we want to be saved with the name (here it is the **summarSE()** results).

```
sepal_length_sum <- summarySE(data = iris, measurevar = "Sepal.Length", gr
oupvars = "Species", na.rm = TRUE)
```

Notice that R did not print the results from the function in the same way it did earlier. This is because you have assigned the object to a name. If you want to view the results after assigning the name, you now just need to run the name assigned to the output like so:

```
sepal_length_sum
   ##      Species  N Sepal.Length        sd         se        ci
## 1     setosa 50        5.006 0.3524897 0.04984957 0.1001765
## 2 versicolor 50        5.936 0.5161711 0.07299762 0.1466942
## 3  virginica 50        6.588 0.6358796 0.08992695 0.1807150
```

Last thing we will do with the **sepal_length_sum** data frame will be to display it as a nice table in the R markdown output. This is a nice way to display your data once you have a summary of the data. To do this, we will use the **knitr** package with the function **kable()**. This package (and function) can do a lot of really helpful things for R markdown output files, especially if you use them to create reports on data (you can review the documentation of the package here or read specifically about the kable function).

We will need to install and load the **knitr** package to make nice tables in the R markdown output.

```
## You do this one! How do you install the knitr package? Add the code bel
ow.
# install package
   ## How do you load the knitr package? Add the code below.
# load package
```

Once you have **knitr** loaded, making a basic table is very simple. The name of the data frame we want to display as a table just needs to be typed within the parentheses like so:

```
kable(sepal_length_sum)
```

Species	N	Sepal.Length	sd	se	ci
setosa	50	5.006	0.3524897	0.0498496	0.1001765
versicolor	50	5.936	0.5161711	0.0729976	0.1466942
virginica	50	6.588	0.6358796	0.0899270	0.1807150

The default **kable()** table is pretty good, but you can also add helpful things like a table caption. Below we add a caption to the sample table created above, but you are encouraged to explore other ways to modify the table by reading the function documentation. To add the caption, we add the code **caption = ""** within the parentheses of the function and then type the desired table caption within the quotes (**" "**).

```
kable(sepal_length_sum, caption = "Table 1 | Summary statistics of sepal l
ength by species created using summarySE()")
```

Table 1 | Summary statistics of sepal length by species created using summarySE()

Species	N	Sepal.Length	sd	se	ci
setosa	50	5.006	0.3524897	0.0498496	0.1001765
versicolor	50	5.936	0.5161711	0.0729976	0.1466942
virginica	50	6.588	0.6358796	0.0899270	0.1807150

Method 2: Tidyverse

The second method we will use will rely on **tidyverse** syntax. This is a really useful collection of R packages that are designed for data science and make your code very clean for other people to understand. This method will require a bit more effort up front, but is more flexible than using the **summarySE()** function.

Start by installing and loading the **tidyverse** package.

```
## You do this one! How do you install the tidyverse package? Add the code
 below.
# install package
   ## How do you load the tidyverse package? Add the code below.
# load package
```

Once you have **tidyverse** installed and loaded, we can build the code to calculate some summary statistics. Just like when we use the function **summarySE()**, we need to specify how we want the data to be grouped for calculating the mean and other statistics. Above we grouped by **Species** so we will do that again here with the **group_by()** function.

In addition to defining the grouping variable, we then need to add a function that we want to apply to the data. For now, we will just calculate the mean **Sepal.Length** per **Species** like above. To calculate mean in the tidyverse, we use the **summarise()** function (you can read more about it here). Within the **summarise()** function, you can pass a summary function, such as the **mean** function, and name that column something (here we are naming it **mean_Sepal.Length**).

We will add these functions to our data (**iris**) using the pipe (**%>%**) operator. This pipe has been defined in the tidyverse to pipe a value forward into a function like below. We can pipe several functions together for our desired outcome, which is why using tidyverse syntax produces fairly clean code.

```
   iris %>%
  group_by(Species) %>%
  summarise(mean_Sepal.Length = mean(Sepal.Length))
   ## # A tibble: 3 × 2
##   Species    mean_Sepal.Length
##   <fct>                  <dbl>
## 1 setosa                  5.01
## 2 versicolor              5.94
## 3 virginica               6.59
```

We now see the mean **Sepal.Length** per **Species** has been calculated. We can compare this to the mean calculated above using **summarySE()** and they result in the same values with slightly different significant digits. One thing to note about using **tidyverse** is that it will generally convert **data.frame** objects to **tibble** format, which is just a 'lazy' type of data frame that can be faster to process, but you can read more about that here if you are interested. Just know that **tidyverse** will convert to a **tibble** and that may impact some functionality of your new data structure in some situations. Here, this will not be a problem so we will move forward.

Now that we have calculated the mean **Sepal.Length** per **Species**, we want to also add the sample size (**N**) and standard deviation (**sd**) to our function. We will add the standard deviation calculation to the code by adding the **sd()** function within the **summarise()** function, similar to the **mean()** function. Next, we will add the sample size with the function **n()** again within **summarise()**. See that within the **summarise()** function that the other functions are separated by by **,**. If you do not include those commas, the code will result in an error.

```
   iris %>%
  group_by(Species) %>%
  summarise(mean_Sepal.Length = mean(Sepal.Length),
            sd_Sepal.Length = sd(Sepal.Length),
            N = n())
   ## # A tibble: 3 × 4
##   Species    mean_Sepal.Length sd_Sepal.Length     N
##   <fct>                  <dbl>           <dbl> <int>
## 1 setosa                  5.01           0.352    50
## 2 versicolor              5.94           0.516    50
## 3 virginica               6.59           0.636    50
```

Again, this will now produce the same values as the **summarySE()** function used above with slightly different significant figures.

The **summarise()** function we are using here is really powerful for these summary data (mean, sample size, mean, etc.), however, for some summary statistics we will need to perform small calculations. For example, if we want to calculate the standard error (SE) of the mean, we will need to use the sample size (**N**) and standard deviation (**sd_Sepal.Length**) (see the equation for SE above). We can add another line within the **summarise()** function to also calculate the SE using the named variables for the mean ($\overline{X}$; **mean_Sepal.Length**), sample size (N; **N**), and standard deviation (σ; **sd_Sepal.Length**) like shown below:

```
  iris %>%
  group_by(Species) %>%
  summarise(mean_Sepal.Length = mean(Sepal.Length),
            sd_Sepal.Length = sd(Sepal.Length),
            N = n(),
            se_Sepal.Length = sd_Sepal.Length / sqrt(N))
   ## # A tibble: 3 × 5
##   Species    mean_Sepal.Length sd_Sepal.Length     N se_Sepal.Length
##   <fct>                  <dbl>           <dbl> <int>           <dbl>
## 1 setosa                  5.01           0.352    50          0.0498
## 2 versicolor              5.94           0.516    50          0.0730
## 3 virginica               6.59           0.636    50          0.0899
```

This looks great! The last thing to do now is assign this **tibble** to a name just like with the **summarySE()** output. For this one, we will name it **sepal_length_sum_tidy** to specify that is it the summary output from the `**tidyverse** method. Go ahead and name it yourself here:

```
   ## Name the above new object 'sepal_length_sum_tidy' like we named the
## summarySE() output above using the left assignment (<-)
```

Go ahead and also display the **sepal_length_sum_tidy** data as a table (using **kable()** like above) with a descriptive caption:

```
   ## Use the kable() function to make a table of your ```sepal_length_sum_ti
dy``` data that will
## display the data nicely in the R markdown output
```

First R Assignment:

Your first lab assignment is to get you a bit more familiar with R. **This will be due by the the start of next lab**. You will submit both your R markdown code and the Word output of your work. This assignment can be completed with other members from your lab section but if you work with others, please include their names on your submission.

1. **Create a R markdown document that will knit as a Word document.** You will submit both the R markdown code and the Word document for this assignment so this will be assessed as a part of your submission.

2. **Calculate summary statistics.** Using the *iris* data, calculate the *mean*, *sample size*, *standard error*, and *standard deviation* of the **Petal.Width** per species of flower. Do this using both methods we discussed in lab:
   A. Using *summarySE()*
   B. Using *tidyverse*
3. **Display summary data as R markdown table** Choose one of the summary data frames you create above (either using *summarySE()* or *tidyverse*) and have it display as a table in your R markdown output.

Helpful resources

- R tutorial with links to other tutorials/resources
- R Markdown cheat sheet
- Data visualization with ggplot2 cheatsheet
- Base R cheat sheet
- A bunch of R cheat sheets

Still stuck? Reach out to your TA/TF or attend an R help session held throughout the semester!