**Coastal Solutions Workshop**
**February 20, 2023 2:30-5:30 PM**
MODULE 1: Introduction to R

R is a free statistical software program which is incredibly versatile, useful, and commonplace in environmental fields. As with any coding language, there is a learning curve associated with learning R. Don't expect to feel like an expert right away – be patient with yourself and your confidence will build over time.

The goal of this module is to introduce the R language and practice with several common functions. By the end of this lesson, you should be able to:

- Set up an organized script, load data, and explore data.
- Use which() and apply() to answer questions about an example dataset.
- Produce a rough plot using example data.
- Understand a simple for-loop.
- Conduct simple statistical tests in R.

**Basic tasks**

To some degree, R is just a gigantic calculator. You can use normal mathematical operations in your code. For example:

```
2+5
2*5
2/5
2^5
```

Use arrows to assign objects.

```
x <- 3
y <- 10
x + y
x * y
```

R has simple naming conventions for objects:

- Object names can't *start* with a symbol or number. Object names can't *end* with a symbol.
- No spaces

```
ice.cream <- "yes please!"
```

**Types of objects**

Scalars are objects composed of 1 item.

```
x <- 3
scalar <- 2 + 2
ice.cream <- "yes please!"
```

Vectors are 1-dimensional objects of the same data type.

```
v1 <- c(2, 10, 5)
v2 <- c("Coastal", "Coastal", "Inland")
length(v1)
```

Matrices are 2-dimensional objects of the same data type, meaning that they have x number of rows and y number of columns. While there are a variety of ways to make vectors, let's take a look at the three most common ones (mat1, mat2, and mat3).

```
x1 <- c(1, 3, 3)
x2 <- c(4, 5, 6)

mat1 <- cbind(x1, x2)
mat2 <- rbind(x1, x2)

mat3 <- matrix(c(1, 2, 3, 4, 5, 6), nrow=3, ncol=2, byrow=TRUE)
mat3
dim(mat3)
```

Data frames are a set of vectors in "spreadsheet" format. Different vectors can have different data types! You can create a data frame using an existing matrix (e.g., df1), or you can create a data frame by giving data to the `data.frame()` function and assigning column names.

```
df1 <- data.frame(mat3)
df1

df2 <- data.frame(month = c("March","April","May"),
       bird_index = c(4, 9, 11))
df2
```

**Indexing**

We can use square brackets to access data. Here are some examples using our v3 vector and our mat3 matrix.

```
v3 <- 1:15
v3[2]
v3[6]
v3[c(1,4,14)]

v3[-2]
mat3[3, 1]
mat3[-2, ]
```

**Subsetting data frames**

Similar to accessing different elements in a vector or matrix, we can subset parts of a data frame to select and pull out components. As before, we can use square brackets to index certain values. The "$" symbol denotes a column by name.

```
df2$month
df2[ ,1]
df2[ ,"month"]

df2[2,1]
df2$month[2]
```

**Classes**

R has names for different classes of data.

Numeric: Numbers, including Inf and NA
Character: Non-numeric strings, denoted by " "
Factor: Categories.
Logical: TRUE or FALSE

```
class(df2$month)
class(df2$bird_index)
is.numeric(df2$month)
is.numeric(df2$bird_index)
```

**Keeping it organized**

It's very important that we clearly describe what's happening in our scripts – we want our code to be *repeatable* (meaning that you could re-run and get the same results), and *shareable* (meaning that the person next to you could run it and understand what's happening). We can use the "#" symbol to add annotations to our code.

Four best practices to set up your script:

a) Describe what the code does.
b) Load any packages
c) Set your working directory
d) Load data

```
#### ------------------------------------------------------------ ####
#### Coastal Solutions Workshop: Introduction to R
##     This script uses example datasets to practice with R.
##     Questions come from the workshop handout.
#### ------------------------------------------------------------ ####

#### Set working directory
setwd("C:/Users/Andrew/Dropbox/Coastal_solutions")

#### Load data
worldfloras <- read.csv("worldfloras.csv")
```

Remember that you can also set your working directory using the RStudio interface. This will automatically run the `setwd()` command using the folder you select. To make your code repeatable, it's often best practice to copy + paste the `setwd()` code from the console to script. Session >> Set Working Directory >> Choose Directory

**Other tips and tricks**

1. Using tab will make life easier
   - Tab can show you available column names
   - Tab can show you available arguments
   - Tab and auto-complete object names

2. In the console, use the "up" arrow to bring up the previous entry.

3. R has easy functions for common descriptive stats:

```
mean()
median()
```

```
min()
max()
```

```
sd()
summary()
```

4. Use `?function()` to get help.

```
?mean
```

#### ---------- **Practice Break #1!** ---------- ####

*Getting set up*

1. Close out of RStudio completely. You do not need to save your changes or save the workspace. *Note: Saving your workspace means that R will create a .RData file to save your objects and a .Rhistory file to save the console text.*

2. Open up a new script and get set up to work with the worldfloras dataset. Describe what your script will do, set your working directory, and load in the worldfloras dataset. See the example code above.

3. Save your work by clicking the "save" icon in the top banner of the script window, and name your script. You'll notice that the script name changes from red to black when there are no unsaved changes. Just like a word document, it's good practice to save your script as you work.

*Exploring the data*

4. How many rows in the worldfloras dataset? *Hint:* `nrow()`

5. It's always a good idea to check your data before running any summaries. After using `View()` to show the data, try these two other functions: `head()` `str()`

6. Use the `summary()` command to find the mean and median endemism rate.

7. What is the mean and standard deviation for the number of plant species per country?

#### ---------- ####

**which()**
Evaluates logical expressions and returns TRUE index values. It also pairs well with indexing!

```
x <- 1:15              # shortcut to create a vector 1-15
which(x==12)
which(x != 10)
which(x==4 | x==5)     # | means "or". & means "and".

## Try it out with indexing
worldfloras[which(worldfloras$Population > 100),]
worldfloras$Area[which(worldfloras$Continent=="S.America")]
```

**apply()**
Applies a function to data or groups of data. There are related functions for applying functions across different types of data: `sapply`, `tapply`, `lapply`.
`tapply()` comes in handy for applying a function by a group ID.

```
apply(X = worldfloras[,c(2:5)], MARGIN = 2, FUN = mean)

tapply(X=worldfloras$Flora, INDEX=worldfloras$Continent, FUN=mean)
```

#### ---------- **Practice Break #2!** ---------- ####

1. How many countries have greater or equal to 5000 plant species? *Hint: look up the length() function.*

2. Which country has the highest rate of endemism? *Hint: look up the which.max() function.*

3. What is the sum of the population within each continent?

4. What is the average latitude within each continent?

#### ---------- ####

**Plotting**
Plotting is hierarchical in R. That means that there are different levels of functions that control different parts of the plotting environment.

`par()` sets plotting parameters. Most par arguments can also go straight into the plotting function. Parameters set by `par()` carry over to multiple plots. To clear them, use `dev.off()`.
Here's an example with some common parameters. More on this later.

```
par(las=1,tcl=0.3,bty="n",cex.axis=0.9, cex.lab=1.5,mgp=c(2,0.2,0))
```

The next layer in our hierarchy is plotting commands which create a brand new plot. The basic plotting function, `plot()`, takes an "x" argument and a "y" argument, which plot data along the x and y coordinates.

```
plot()
hist()
boxplot()
barplot()
```

Finally, we can use lower-level functions to add things to plots.

```
points()
lines()
polygon()
arrows()
```

```
legend()
axis()
title()
```

R also comes with over 650 colors which you can access by name.

```
colors()
plot(x = worldfloras$Latitude, y = worldfloras$Flora, col="red")
```

Hexidecimal color codes also give a lot of flexibility. Hex numbers range from 00 to FF, giving 255 different values.
The 3 units of color are Red, Green, and Blue.

Black: "#000000"
White: "#FFFFFF"
Green: "#00FF00"

For a great way to visualize and select hexcolors, visit https://htmlcolorcodes.com/.

```
plot(x=worldfloras$Latitude, y=worldfloras$Flora, col="#006BFF80")
```

#### ---------- **Practice Break #3!** ---------- ####

1. Create a histogram showing the distribution of endemism rates.

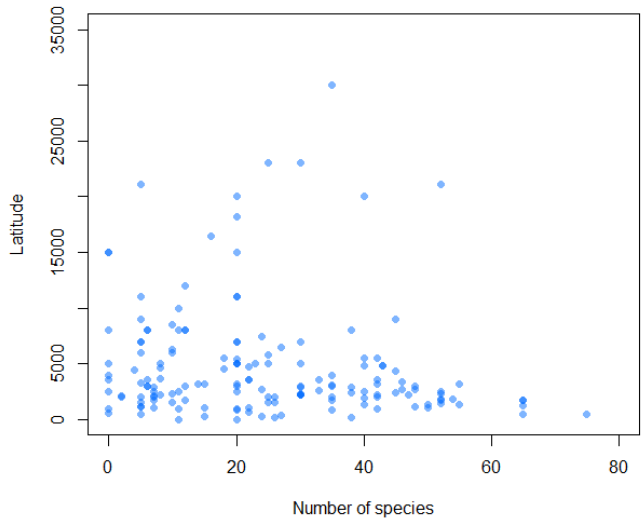
2. Create a basic scatterplot showing the relationship between latitude (x axis) and the number of species (y axis).

3. Add more descriptive x and y axis labels to your plot (e.g., rename the x axis to "Number of species"). *Hint: look up the arguments xlab and ylab for help*.

4. Change the limits of the x and y axes so that the x axis shows 0-80 and the y axis shows 0-35000. *Hint: look up the arguments xlim and ylim for help*.

5. Use colored circles for each point instead of the default open circle. *Hint: look up the arguments pch and col for help. Google "R pch" for a chart of available options*.

Your final plot should look something like this!




#### ---------- ####

**For-loops**

Just like most programming languages, R has a way to iterate calculations using "loops". The "for-loop" is the most common.

In the parentheses after "for", you define a sequence of values which sets the number of iterations. Then, for each value in the sequence, you can provide a calculation. The first line in the example below would read: "for each value *i* in 1–3..."

```
for (i in 1:3) {
  txt <- paste("the square of", i, "is", i * i)
  print(txt)
}
```

We can also store the output in a vector by combining the for-loop with indexing. To do this, we first create an empty vector, "out", with the length we want. Then, we use a for loop to fill that empty vector with the specified values.

```
out <- rep(NA, 50)
for (i in 1:length(out)) {
  out[i] <- paste("the square of", i, "is", i * i)
}
```

**Correlation test**

After reading in the dataset "Woodpecker_data.csv", we can test whether two variables are correlated with each other. Note that the order of variables doesn't matter. In this example, the Pearson correlation coefficient (r) is 0.026.

```
cor.test(hr$Dead_tree_BA, hr$Elevation)
```

**Student's t-test**

```
t.test(x, y) # Just an example using made-up variables.
```

**ANOVA**

We can use a 1-way ANOVA to test whether the means are statistically different when there are 3 or more populations. In our case, we will compare home range size between the 3 national forests. Use the `aov()` function to run an ANOVA. The function `TukeyHSD()` conducts Tukey's Honest Significant Different test, which compares means between individual groups.

```
mod1 <- aov(hr$HR_size ~ hr$Nat_forest)
summary(mod1)
TukeyHSD(mod1)
```

**Simple linear model**

It's often best practice to save the model as an object so that we can access it later. The function `summary(...)` gives use a helpful summary of the output.

```
mod2 <- lm(formula = HR_size ~ Dead_tree_BA, data = hr)
summary(mod2)
```

The lower-level plotting function `abline()` can take the model object as an argument and plot the mean relationship between x and y.

```
plot(hr$Dead_tree_BA, hr$HR_size, pch=16)
abline(mod2, col="blue")
```