Introduction to R

2024-04-08

Today we are going to Install and run some very basic actions using R.

I use a combination of the base R program, and a developer envionrment called R studio. Both are freely available online. The work on Mac, PC, and Linux.

R can be downloaded from the CRAN website https://cran.r-project.org/.

R-Studio can be downloaded from Posit https://posit.co/download/rstudio-desktop/.

Once you have both installed, we can start to enter commands into the console. The console is responsible for 'running' the code you type. Let's try to use R as a calculator.

```
5+2
```

```
## [1] 7
```

```
7+1
```

```
## [1] 8
```

```
1000/200
```

```
## [1] 5
```

You can also assign values to 'objects' and run those instead, for example: Here we assign the value of 10 to X, and 15 to Y.

```
X <- 10
Y <- 15
X + Y
```

```
## [1] 25
```

The <- arrow here means 'assign' and it can be used alongside the '=' sign to assign values to objects. Stay consistent, it is easier to assign using the <- symbol.

```
A <- 2
B = 8
A * B
```

```
## [1] 16
```

R can process many different types of data. Text data is read as 'strings' and are designated as character or factor data. This is how you can define discrete variables.

```r
K <- "BIO206"
L <- "_tutorial"
paste0(K,L)
```

```
## [1] "BIO206_tutorial"
```

What you see above is the use of a function - paste0 is a function. Functions are the backbone of R and allow you to perform a whole host of different tasks related to processing your data of choice. Let's build some pretend data and apply some functions.

Let's run some functions related to the mean and median, two descriptive statistics that allow us to estimate the central tendency of our data. The mean and median reflect two summary measures that attempt to describe the center of the distribution using a single value.

```r
MyVector <- c(3,8,9,5,10)
MyVector
```

```
## [1]  3  8  9  5 10
```

```r
mean(MyVector)
```

```
## [1] 7
```

```r
median(MyVector)
```

```
## [1] 8
```

Here we used the function 'c' which means concatenate, to combine a series of numbers. R sees these numbers as a vector. Vectors have unique properties that allow us to run specific functions on them.

We don't always have all the data we need for an individual sample, and we need to tell R so as not to encounter an error. This leads us to talking about 'arguments' in the context of R coding.

```r
MyVector2 <- c(15,20,31,NA,25,19,28)
mean(MyVector2)
```

```
## [1] NA
```

```r
mean(x = MyVector2, na.rm = TRUE)
```

```
## [1] 23
```

You'll notice that we've entered some extra pieces in between the parentheses. In the prior code chunks I added nothing but the object. That's because R has a built-in default for each function - I just wanted to use the default mean calculation. However, when we have missing data we need to something different because R does not know how we want to process missing values.

If you hit the TAB key when your cursor is within the function parentheses R-Studio will give you options for what you need to fill-in. Here, I need to add 'na.rm = TRUE' which means NA values should be removed. This extra argument must be separated from the first argument by a ',' symbol.

```
BodyMass <- c(162,164,155,198,201,175)
BodyHeight <- c(172,168,160,175,182,171)
Sex <- c(rep(x = "Female", 3), rep(x = "Male", 3))
BodyData <- cbind.data.frame(Sex, BodyMass, BodyHeight)
BodyData
```

```
##      Sex BodyMass BodyHeight
## 1 Female      162        172
## 2 Female      164        168
## 3 Female      155        160
## 4   Male      198        175
## 5   Male      201        182
## 6   Male      175        171
```

Now we're starting to combine multiple functions together to build a 'data frame' which is just a way to describe a table in R.

We began by building two vectors of continuous data using the 'c' function, and naming the objects. We then built a third vector that included character data to describe a discrete variable. What's good about R is that you can build functions within functions. Here, we use 'c' and 'rep' to join two repeated character strings together in a single vector. Finally, we joined all three vectors together via column-binding and printed the data frame to the console.

Let's observe how R is 'seeing' our data.

```
str(BodyData)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ Sex       : chr  "Female" "Female" "Female" "Male" ...
##  $ BodyMass  : num  162 164 155 198 201 175
##  $ BodyHeight: num  172 168 160 175 182 171
```

You'll see that there are two different data types here. chr means character and num means numeric. What if I wanted to view just part of the data table. I can do one of two things. I can use a function to look at the top set of rows (just see if everything looks right), and I can also using 'indexing' to view specific parts of the data table. Indexing is achieved by simply placing some square brackets after your object.

The general format looks like this: MYOBJECT[rows,columns]. Your data frame has two dimensions, rows and columns, hence why we need to separate these out using a comma in the indexing square brackets.

I also use a colon to produce a sequence of numbers. By typing 1:5, you will get a sequence of numbers from 1 through to 5. You can also use the 'c' function to join non-adjacent columns or rows together. If you leave and entry blank, R will assume you want to include all the rows or columns in your data frame.

```
head(BodyData)
```

```
##      Sex BodyMass BodyHeight
## 1 Female      162        172
## 2 Female      164        168
## 3 Female      155        160
## 4   Male      198        175
## 5   Male      201        182
## 6   Male      175        171
```

```r
BodyData[ ,2:3]
```

```
##   BodyMass BodyHeight
## 1      162        172
## 2      164        168
## 3      155        160
## 4      198        175
## 5      201        182
## 6      175        171
```

```r
BodyData[1:2,]
```

```
##      Sex BodyMass BodyHeight
## 1 Female      162        172
## 2 Female      164        168
```

```r
BodyData[1:2,1:2]
```

```
##      Sex BodyMass
## 1 Female      162
## 2 Female      164
```

```r
BodyData[,c(1,3)]
```

```
##      Sex BodyHeight
## 1 Female        172
## 2 Female        168
## 3 Female        160
## 4   Male        175
## 5   Male        182
## 6   Male        171
```

What if we wanted to find the mean and median of our body mass and body height data for males and females independently? We can do that using some other very helpful functions 'apply' and 'aggregate.'

```r
apply(X = BodyData[,2:3], MARGIN = 2, FUN = "mean")
```

```
##   BodyMass BodyHeight
##   175.8333   171.3333
```

```r
aggregate(x = cbind(BodyMass,BodyHeight)~Sex, FUN="mean", data = BodyData)
```

```
##      Sex BodyMass BodyHeight
## 1 Female 160.3333   166.6667
## 2   Male 191.3333   176.0000
```

```r
aggregate(x = cbind(BodyMass,BodyHeight)~Sex, FUN="median", data = BodyData)
```

```
##      Sex BodyMass BodyHeight
## 1 Female      162        168
## 2   Male      198        175
```

The 'apply' function allows us to apply a function across a series of rows or columns. You give the function your data frame (or a subset of that data frame as I have done here), select whether you want to assess rows (MARGIN=1) or columns (MARGIN=2), and then include the function name in quotes (I wanted to assess the mean here).

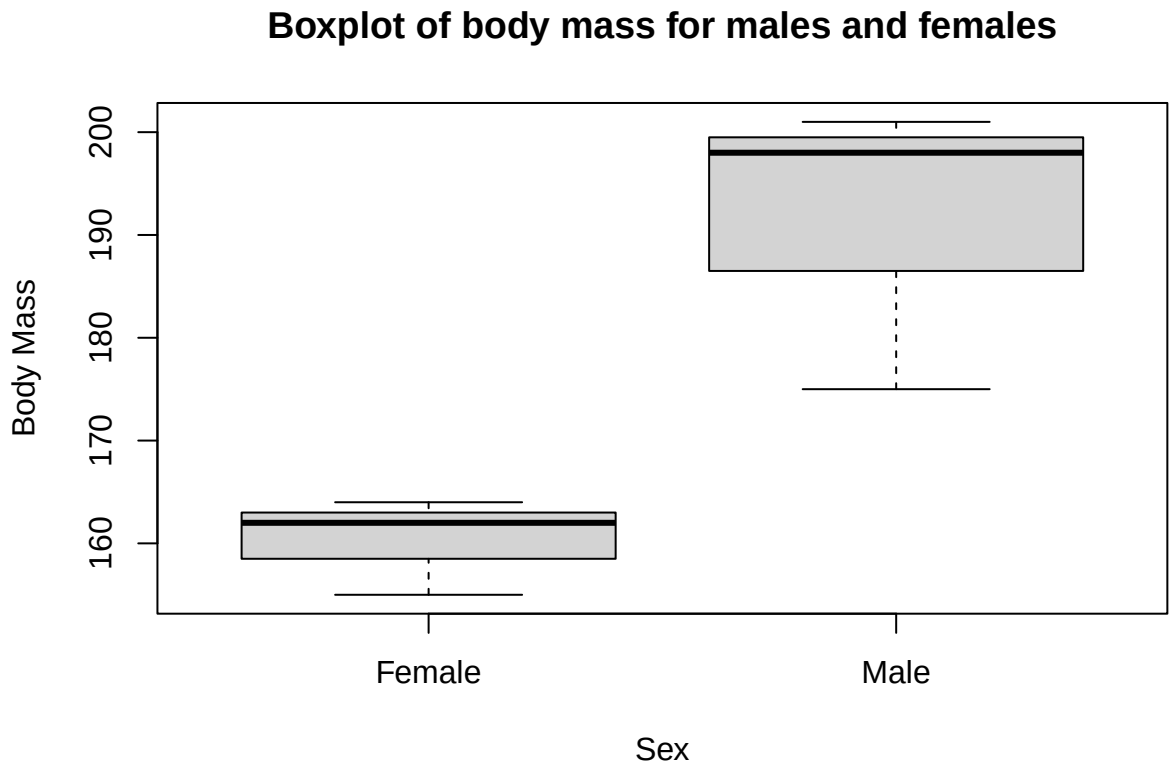
The aggregate function is immensely useful, allowing you to calculate group means or medians (or any function) across a suite of data. Here, we want to calculate body mass and body height means/medians for each sex independently.

Plotting

Finally, let's produce a boxplot so we can visualize our body size data.

We need to add some extra arguments so we can label the axes and title the plot. I chose to plot the body mass data, but you could also try the height data as well to see how that changes.

```
boxplot(BodyData$BodyMass~BodyData$Sex,
        xlab = "Sex", ylab = "Body Mass",
          main = "Boxplot of body mass for males and females")
```



RESOURCES TO LEARN R.

R has a whole host of different packages that do not come with the base installation. Some of them we will be using during this class. One of my favorites is called Swirl. Swirl is a package that allows you to learn R, in R. It types prompts to you in the console and you can answer the questions it gives you. It is an immensely useful tool to aid in learning the ins-and-outs of the R programming language. To install, follow the code chunk. Note that I have added a pound symbol, or hash tag, as it allows you to add comments to a line of code so you can remember what a specific line does. They are very useful to describe what you're doing in a team setting where you are likely to share code.

```
#To make the lines below run, you need to remove the pound symbol from them.
#I have them here so the code does not execute when I Knit the document.

#install.packages("swirl")
#library(swirl)
```

TASK.

I want you to find the mean and median from a sample population of two closely related primate species: bonobos and chimpanzees. The data here is measurement data of the skull length and skull height separated by sex.

I want you to construct a data frame that includes the data below and obtain species means and medians. Then find the mean and median for each of the four groups (species*sex).

Chimp skull height F (cm) - 25, 35, 53, 47, 41, 61.

Chimp skull length M (cm) - 45, 48, 55, 51, 49, 66.

Bonobo skull height F (cm) - 34, 33, 59, 45, 64, 72.

Bonobo skull length M (cm) - 43, 49, 50, 48, 54, 61.

Plot a boxplot with species and skull length. Finally, plot a boxplot with all four groups.