R Overview

**Importing Data**

* Can import many different file types using the “read.csv”, “read.delim”, etc commands
* Or can use the R Studio tools: File 🡪 Import Dataset 🡪 From text (base)
* Often you’ll want to specify that you have column headers or row names
  + Can do this either by adding arguments to the read.XXX function (see example below) or specifying this in the window that pops up when using the R studio tools

Ex:

MyData = read.csv(“~/Downloads/NameofDataset.csv”, header = TRUE, row.names = 1)

* Part in quotes = the file path. Make sure to also include the file extension
* header = TRUE means the first row of your data contains the column names
* row.names = 1 means that that first column of your data contains the row names

**Basic Functions to Quickly Examine Your Data in R**

* Once you’ve imported your data, there are a few handy functions you can use to make sure the data looks the way you want

View(data) # Pulls up the whole dataset in the script window area

head(data) # Displays the first 6 rows of the dataset in the console. tail(data) shows the last 6 rows

class(data) # Reports the data “class” (see box below for more info about classes)

dim(data) # Reports the no. of rows and columns. length(data) gives the no. data points for a vector

summary(data) # Provides basic stats summary info (like mean, range, etc) for each column

**Installing and Loading Packages**

* This is how you access functions beyond the basic ones included in base R
* Can do it at the command line using the “install.packages” function (see example below)
* Or can use the R studio tools: Go to packages tab in the bottom right window 🡪 Install 🡪 enter the name of the package you’re looking for
* After you’ve installed a package, you need to load it before you can use it. Do this by either using the “library” function (see example below) or clicking the checkbox next to the package name in the package library list

Ex:

install.packages(“vegan”) # This installs the package

library(vegan) # This loads the package

**Data Classes**

* Any type of data you work with in R will have a “class” either assigned to it, or which you define
* Some of the most common classes include:
  + numeric -- for real numbers
  + integer – for integers
  + character – for text data
  + factor – for #s or words that have some sort of structure (e.g. letter grades A, B, C)
  + logical – for True/False statements

**Adding or Changing Row & Column Names**

* Extract the row/column names using the “rownames” or “colnames” function
* Overwrite original row or column names by passing new information after calling the row or column names (see example)

Ex:

colnames(data) = c(“Species”, “Year”, “Site”) # This forces the column names of Data to be these 3 names.

rownames(data)[4] = “Bird125” # This changes ONLY the 4th rowname in data. All other remains as they were

**Subsetting Data**

* It will often be useful to access certain subsets of your data for analysis. There are a few ways to do this.
* If you know the row or column numbers you want, you can use those within square brackets

Ex:

Data[1,2:5] # Pulls out the 2nd – 5th columns for the 1st row only

Data[, 6] # Since nothing was passed in the row argument, this pulls out the info in the 6th column for ALL rows.

* For columns, if you know the column name, you can use the ‘$’ operator

Ex:

Data$Species # Pulls out the info in the Species column, for all rows

* If you know want to pull out subsets of data based on some sort of property, you can use logic statements within the square brackets

Ex:

Data[Data$Species == “Corvus corax”, ] # Pulls out all the rows where the Species name is Corvus corax

Data[Data$Age > 3, ] # Pulls out all the rows where the value in the age column is greater than 3

OldBirds = Data[Data$Age > 3, ] # Same as above but stores the output (the subsetted data) in a new variable

**Basic Stats**

* There are many simple functions for descriptive statistics such as: mean, median, range, etc.

Ex:

sd(Data$Age) # Calculates the standard deviation of the age column

* There are also functions for running basic statistical tests and regressions such as: t-tests, chi-squared, ANOVAs, simple linear regression
* Going beyond base R, there are also R packages that allow you to do more advanced statistics like mixed-effects modeling, Bayesian stats, principal coordinates analysis, etc.

Ex:

fit1 = aov(Age ~ Species, data = Data) # Runs an ANOVA to see if there are differences in age based on species. If you want to examine the output, you would run: summary(fit)

fit2 = lm(Weight ~ Age, data = Data) # Creates a linear model to see how weight relates to age. To examine the output, you would again use the summary function: summary(fit2)

**Basic Plotting**

* Base R supports many types of plots: histograms, boxplots, scatterplots, line plots, etc.
* The same basic arguments are common across plotting types

Ex.

plot(Weight ~ Age, data = Data, main = “Weight by Age”, xlab = “Age in Weeks”, ylab = “Weight in grams”, xlim = c(0,20), ylim = c(10,17), col = “red”) # creates a scatterplot

* Main adds a plot title. xlab & ylab add x and y-axis labels
* xlim and ylim define the x and y- axis range
* pch defines the plotting symbol (ranges from 1-16), col defines the plotting color
* Pull up the help file for the specific plotting command you’re using to find additional arguments

**Repeating Code/Functions Many Times**

* The *apply function family* (mapply, sapply, lapply, etc.) are useful for repeating a certain function many times

Ex.

sapply(Data[,6:11], FUN = mean, na.rm= TRUE) # First argument tells R the range of columsn/rows you want the function repeated. Second argument tells R what function you want repeated, third argument is included because you want R to ignore any missing values when doing this calculation

Ex.

tapply(Data$Age, Data$Species, FUN = median) # tapply can be used when, instead of specifying the rows/columns, you want to specify that you want the data broken apart based on some other factor. Here, we’re telling R to calculate the median age for all observations broken down by species (i.e. separate median calculations for each species)

* If you want a more complicated function or set of commands repeated, you’ll probably instead want to use a *for loop*
* In a *for loop*, you tell R what code you want repeated, and define the range over which you want it repeated
* The basic structure is:

for (i in 1:10) # Starts with “for” and then a range of values for “i” to loop through

{ print(i) } # some function/ code that you want repeated many times

Ex:

avggrades = rep(0, 16) # Create an empty vector to store values in later

for (i in 1:16) # Define the loop range

{ avggrades[i] = mean(grades[i,]) } # Calculate the mean grade and store it in avggrade

**Creating your own functions**

* Base R, and the many available add-on packages have lots of functions, but there may come a time where you want to create your own function specific to the type of analysis you’re doing.
* The structure of defining a function is essentially this:

FnName = function(x)

{y = x^2 / 3.14

print(y)} # This defines a function called FnName that takes a single input, takes this input, squares it and divides by 3.14, and returns this new value.

Ex.

BMI = function(x,y) # Function name is BMI and has 2 inputs

{print(x / y^2 \* 703)} # Function takes first input, divides by second input squared and multiplies by 703

**The Aggregate Function**

* So powerful, it gets its own section
* The aggregate function splits data into subsets and calculates statistics for each subset
* It’s similar to tapply (see above), and the summary function, but more generalizable

Ex.

Output = aggregate(Data$Weight, by = list(Data$Species), FUN = mean) # Tells R to calculate the mean weight for all individuals based on their species, and store these mean values in a new variable called “Output”