Handling Data in R

# Data Import

## Desktop version of R

Keep your data in the same folder as this .Rmd file. If they are not kept in the same folder, then you have to tell R exactly where your data is by copying and pasting the file pathway directly into R. But overall, it is much easier and simpler to keep everything in the same folder. By default, the folder that your .Rmd file is in is called the “working directory”. This is the location that R assumes all of your input files are located. R will also place all of your outputs in this working directory, unless you specify otherwise.

## RStudio online

Upload the data file like demonstrated in the video, then write the same read.csv() function to import the data file onto R.

# Input & Output

You can check what your working directory is by using the function getwd(), or “get working directory”. Interestingly, this code has no arguments - in other words, it does not need anything nested between the () to work - and its output should be a folder pathway to where your .Rmd file is. If you want to change your working directory - which at this point in time is not recommended - you can use the function setwd(), or “set working directory”. Unlike getwd(), you need to paste the desired folder pathway between the () of setwd() in quotation marks "" to set a new working directory. For example: setwd("C:\\Desktop\\Handling data in R\\") if you are using Windows and setwd("/Users/Desktop/Handling data in R/"). Finding a pathway in Mac is relatively complicated, so again, it is highly recommended that you place everything in one folder so you do not have to worry about setting a new working directory.

After setting a working directory, you can check all of the files that exist in your directory by using dir(). This function is fantastic for checking whether you are in the right folder, whether there are any extra files, and, most importantly, whether there are any missing files.

Checking whether you have all of the necessary files in your directory is important will make your life so much easier when you import data into R. For example, if the file **trees\_data.csv** is in your working directory, all you need to do to import this into R is: read.csv("trees\_data.csv", header = TRUE). If this file is not in your working directory, you would need to replace "trees\_data.csv" with the entire pathway of the file - this will make your code much bulkier, more confusing, and less shareable.

Note also that the file name must be in "" and must be exactly like the file on your desktop. A common problem that students have is downloading the same csv file multiple times, resulting in an automatic “-1” (i.e. trees\_data-1.csv) - in this case, you need to also add “-1” to your code for it to work. The argument header = TRUE is also important if you are importing data with existing headers (AKA column names). Try changing it to header = FALSE, what does the imported data look like now?

The functions for importing data from your hard drive into R are relatively intuitive: read.csv() for csv files, read.table() for txt files. There are other functions as well, but we will not be covering them in this course. These are the two main importing functions that we will use.

When installing packages, it is suggested that you # all install.packages() codes into comments because: \* It will cause problems when you try to knit the .rmd file to Word and \* it is generally insensitive to share this document to someone and make them accidentally install something into their hard drive

You can also export the data from R to txt or csv by using write.table(). This table, by default, will be exported to your working directory. Try typing ?write.table into the console, you can read more information about this function there!

# Location of the parent directory / folder you saved this .RMD file  
getwd()

## [1] "/Users/hoangdieuan/Desktop/R1"

# Listing all files in the directory  
dir()

## [1] "f1.csv" "Handling data in R Part 1.Rmd"   
## [3] "Handling-data-in-R-Part-1.html" "Handling-data-in-R-Part-1.Rmd"   
## [5] "height\_dbh.txt" "MyData.txt"   
## [7] "trees\_data.csv" "VRI\_data.csv"

# Importing a csv file  
tree1 <- read.csv("trees\_data.csv", header = TRUE)  
  
# Importing a .txt file  
HDB <- read.table("height\_dbh.txt", header = TRUE)   
  
# Importing VRI data  
vri\_data <- read.csv("VRI\_data.csv", header = TRUE)  
  
# Importing an excel data file  
#install.packages("readxl")  
#library("readxl")  
#tree2 <- read\_excel("trees\_data.xlsx")  
  
# Export data as a .txt file  
write.table(HDB, "MyData.txt")  
  
# Export data as a .csv file  
write.table(HDB, file = "f1.csv", sep = ",", col.names = NA, qmethod = "double")

# VRI Data

Recall that there are several ways that you can call for a variable in a dataset: (1) using $ or (2) using attach(). attach(), you guessed it, attaches the data to this R session so we do not need to use $. attach() is usually more useful when you are only working with 1 dataset in your .Rmd file, else it will get very confusing if you or someone you share this file to run this file out of order.

We can find a summary of all statistics of a dataset using summary(). In the example code below (vri\_summary = summary(vri\_data)), you may see that we are using a = instead of a <- to assign a value to a variable. In this case, = and <- function similarly (you can change = to <- and re-run the code). However, they may have different functions in other scenarios. We will not be covering these scenarios, so for the context of this course, just understand that = and <- are similar. **Important note:** Recall that R’s “equal to” is actually ==. So = and == are **VERY different**!

You can also find the dimension of the data using dim(). This should tell you how large the dataset is (AKA numbers of rows and columns) without you having to run the actual data. This may save you some time if your dataset is too big.

Finally, head() and tail() print the first and last few rows of a dataset for you - again, helpful if your dataset is very large. You can also specify how many rows you want to print by adding an integer after the data frame name. For example, head(vri\_data, 10) will print the first 10 rows for you.

attach(vri\_data)  
  
# Set an object for string the data summary  
vri\_summary = summary(vri\_data)  
  
# Dimension of the data  
dim(vri\_data)

## [1] 2700 5

# Print first few rows  
head(vri\_data)

## OBJECTID SPECIES\_CD\_1 PROJ\_AGE\_1 PROJ\_HEIGHT\_1 LIVE\_VOL\_PER\_HA\_SPP1\_125  
## 1 1 0 0 0  
## 2 2 0 0 0  
## 3 3 0 0 0  
## 4 4 0 0 0  
## 5 5 0 0 0  
## 6 6 0 0 0

# Print last few rows  
tail(vri\_data)

## OBJECTID SPECIES\_CD\_1 PROJ\_AGE\_1 PROJ\_HEIGHT\_1 LIVE\_VOL\_PER\_HA\_SPP1\_125  
## 2695 2695 CW 332 26.2 116.612  
## 2696 2696 YC 287 13.1 39.479  
## 2697 2697 HW 357 39.2 625.704  
## 2698 2698 CW 267 25.2 187.683  
## 2699 2699 HW 332 28.2 212.893  
## 2700 2700 CW 332 26.2 133.921

# Data selection and variable information

Once we have successfully imported our data into R and gained general information about it, we can also go into more specific data exploration. Firstly, we can tell R to only extract a few specific rows and columns from the dataset by using []. Note that the first argument in this bracket represents the rows and the second argument represents the column. For example, when we type vri\_data[100:110, 4:5], we’re telling R to extra **rows 100 to 110** and **columns 4 to 5**. We use : between two numbers to indicate that it is a range of sequential numbers that we want. So, 100:110 means from **100 to 110**.

If it is not a range of rows/columns that you want to extract, but very specific, non-consecutive rows/columns, you can nest all the rows/columns that you want in a vector, like in vri\_data[c(10, 20, 23), ]. Here, we only want **rows 10, 20, and 23**. Finally, if you want to extract all rows or all columns, simply leave the argument blank. In other words, vri\_data[10, ] tells R to extract **row 10** and **all columns**, but vri\_data[ , 10] tells R to extra **all rows** and **column 10**.

The rest of the functions in this tutorial have already been covered by a previous R lecture file or your assignment. The functions themselves are not hard, it is understanding what they do and when to use them that requires a bit more thought. And as always, the best way for you to be better in R is to practice!

# Extracting some rows and all columns  
  
vri\_data[10:13, ]

## OBJECTID SPECIES\_CD\_1 PROJ\_AGE\_1 PROJ\_HEIGHT\_1 LIVE\_VOL\_PER\_HA\_SPP1\_125  
## 10 10 0 0 0  
## 11 11 0 0 0  
## 12 12 0 0 0  
## 13 13 0 0 0

# Printing all variable names   
names(vri\_data)

## [1] "OBJECTID" "SPECIES\_CD\_1"   
## [3] "PROJ\_AGE\_1" "PROJ\_HEIGHT\_1"   
## [5] "LIVE\_VOL\_PER\_HA\_SPP1\_125"

# An example of creating a variable with sequential values  
x <- 1:5  
  
# Finding the average of the variable x  
mean(x)

## [1] 3

# Printing the average of variable ProJ\_Age\_1  
  
mean(PROJ\_AGE\_1)

## [1] 230.6289

# Printing data types of some variables  
  
class(SPECIES\_CD\_1)

## [1] "factor"

class(PROJ\_AGE\_1)

## [1] "integer"

class(PROJ\_HEIGHT\_1)

## [1] "numeric"

# Summarizing the entire dataset  
  
summary(vri\_data)

## OBJECTID SPECIES\_CD\_1 PROJ\_AGE\_1 PROJ\_HEIGHT\_1   
## Min. : 1.0 : 59 Min. : 0.0 Min. : 0.00   
## 1st Qu.: 675.8 CW :1136 1st Qu.:167.0 1st Qu.:13.10   
## Median :1350.5 DR : 19 Median :267.0 Median :20.30   
## Mean :1350.5 HW : 970 Mean :230.6 Mean :20.45   
## 3rd Qu.:2025.2 PLC: 182 3rd Qu.:307.0 3rd Qu.:29.30   
## Max. :2700.0 SS : 218 Max. :543.0 Max. :53.20   
## YC : 116   
## LIVE\_VOL\_PER\_HA\_SPP1\_125  
## Min. : 0.00   
## 1st Qu.: 33.12   
## Median :102.81   
## Mean :156.26   
## 3rd Qu.:237.89   
## Max. :988.53   
##