Importing_Data_Crena

Andrew Crena

2023-05-19

R Markdown

```
# When you know that you will be using the same directory throughout your entire R Markdown, using the
knitr::opts_chunk$set(root.dir = "C:\\Users\\adcre\\OneDrive\\Documents\\R_Programming\\TEND", echo = TR
```

Importing Data in the form of Comma-Separated Value (.csv) Files! But first, some housekeeping:

Probably the most common way of ensuring that your working directory is correct is by using the 'setwd()' function, which I won't go into much detail about. Here is an example code:

```
setwd("C:\\Users\\adcre\\OneDrive\\Documents\\R_Programming\\TEND")

# Confirm by using 'getwd()'

getwd()
```

```
## [1] "C:/Users/adcre/OneDrive/Documents/R_Programming/TEND"
```

While the main objective of this Rmd is to serve as a reference sheet for importing data, I found myself focusing on other tasks that are closely related. That is why you will see tangential code or concepts, however if RStudio has taught me anything, it is that one problem can be solved a million ways! This includes packages like readr, tidyverse, and dplyr, and various functions and syntax within those packages. This project is a non-linear, discovery-learning-based approach, and so there is no priority for any one chunk in this Rmd, rather each one gets added on as I continue learning.

First things first, I will provide various ways of setting your working directory and checking directory status, especially as it pertains to your current project. In the past, I have written an entire R Markdown file without telling the computer what directory I want to pull from at the top lines/chunks of code. This became increasingly frustrating when I started changing file locations and working directories in such a way that prevented me from setting a wd for my markdown file. Thus, I am creating this chunk to "iron-out" anythung when it comes to remaining aware of the working directory you are interacting with.

Importing Data in the form of CSV Files (locally)

It is important to understand how our system is using a working directory for an entire Rmd file, so that you don't have to worry about calling to it in later chunks. We will also learn how to specify explicit directories in specific chunks, but this method serves as an efficient way to centralize your directory for a lengthy or complex markdown file. This is actually accomplished in the first code chunk of this Rmd file, above "### R Markdown". WRITE THIS ARGUMENT INTO THAT KNITR FUNCTION:

Another important tool I use is in the File Browser section (bottom right quadrant). Find the file or folder that you would like to set as your wd, and then press "files". An option to 'set as working directory' will be available. PRESS IT!

In addition, the File Browser introduces an option called 'synchronize working directory' when you have selected a specific file/folder. 'synchronize working directory' refers to a feature ensures that the current wd in your environment is synchronized with the location of your R script or project file. While there is no code (to my knowledge) that can achieve the same task, however I find this to be very useful when staying on top of my working directory.

GETWD() & SETWD()

This may seem obvious, but setting a working directory at the top of a specific code chunk will allow the computer to know what directory you would like to pull from in a specific code chunk. When making a markdown file. Ideally, this line of code would be at the top of your code chunk, before any functions need to interact beyond the console.

```
setwd("C:\\Users\\adcre\\OneDrive\\Documents\\R_Programming")
getwd()
```

```
## [1] "C:/Users/adcre/OneDrive/Documents/R_Programming"
```

MISTAKE/LEARNING LESSON:

When initially writing this code, I was not aware that 'tidyverse' contains many of the packages I was loading, even though they came with tidyverse. This will be important to remember, as it caused conflicts in my output. You just need 'tidyverse'. How Neat!

```
# in order to prevent conflicts from using multiple packages in the same
# notebook, I will use the 'conflicted' package
library(conflicted)
```

```
## Warning: package 'conflicted' was built under R version 4.2.3
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'dplyr' was built under R version 4.2.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
```

At this point, I feel prepared to import the csv files into my working directory. I remain very cautious in these situations, but as long as you write comprehensive and intentional code, the data should run smoothly through your work.

```
brain_data <- read_csv("Brain_Data.csv", show_col_types = FALSE) %>%
    spec()

behavior_data <- read_csv("Bx_Data.csv", show_col_types = FALSE) %>%
    spec()

str((behavior_data))
```

```
##
##  cols(
##    ID = col_double(),
##    SubjID = col_character(),
##    Group = col_character(),
##    Gender = col_double(),
##    Sex = col_double(),
```

```
##    Ethnicity = col_double(),
##    Race = col_double(),
##    Puberty = col_double(),
##    Age_at_Bx = col_double(),
##    Age_at_Scan = col_double(),
##    Height = col_double(),
##    Weight = col_double(),
##    Bx_to_Scan_Days = col_double(),
##    CDRSR_total = col_double(),
##    CDRSR_tscore = col_double(),
##    Current_Med = col_double(),
##    Recurrent_Past_Month = col_double(),
##    Recurrent_Current_Duration_Weeks = col_double(),
##    Age_First_SI = col_double(),
##    SI_Group = col_double(),
##    SI_Past_Month = col_double(),
##    Hx.Attempt = col_double(),
##    Age.First.Attempt = col_double(),
##    Number.of.Attempts = col_double(),
##    RADS_DM = col_double(),
##    RADS_AN = col_double(),
##    RADS_NS = col_double(),
##    RADS_SC = col_double(),
##    RADS_total = col_double(),
##    PHQ9 = col_double(),
##    PHQ9_q9 = col_double(),
##    SIQ = col_double()
##  )
```

What if we are downloading csv files from the web to some exploratory analysis, like from Kaggle.com?

```
# Assuming the csv file has been extracted and is in your working directory, he
# only arguments I will introduce in this chunk is the 'show_col_types'
# argument, which is a useful argument that comes with the 'readr' package.
# When show_col_types=FALSE, it will not show you the type of data that is in
# each column, and vice versa for show_col_types=TRUE. In other words, if you
# have a data set in which the column types (numeric, categorical, etc)
library(tidyverse)
dep_scores <- read_csv("scores.csv", show_col_types = FALSE)
View(dep_scores)

# Next, view your assigned variable to make sure that your data frame was made
# properly
View(dep_scores)

# Instead of using the view function, you can also use the str or 'structure'
# function, which will give you a shorter but more descriptive output of the
# entire data set
str(dep_scores)
```

This next code chunk is an example of downloading a data set from the internet so that it is expressed as a data frame in your working directory. It is a depression data set randomly chosen on Kaggle, and I want to write the code that will grab it from the specific URL that refers to the specific data set, or some code that will achieve the same goal but in different ways. The data set can be found with this link: https://www.kaggle.com/datasets/arashnic/the-depression-dataset In this example, we will show how to get the data set into a df after downloading it and extracting the zip file (I find this to be easiest at my current skill level)

```
## spc_tbl_ [55 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ number   : chr [1:55] "condition_1" "condition_2" "condition_3" "condition_4" ...
##  $ days     : num [1:55] 11 18 13 13 13 7 11 5 13 9 ...
##  $ gender   : num [1:55] 2 2 1 2 2 1 1 2 2 2 ...
##  $ age      : chr [1:55] "35-39" "40-44" "45-49" "25-29" ...
##  $ afftype  : num [1:55] 2 1 2 2 2 2 1 2 1 2 ...
##  $ melanch  : num [1:55] 2 2 2 2 2 2 NA NA NA 2 ...
##  $ inpatient: num [1:55] 2 2 2 2 2 2 2 2 2 2 ...
##  $ edu      : chr [1:55] "6-10" "6-10" "6-10" "11-15" ...
##  $ marriage : num [1:55] 1 2 2 1 2 1 2 1 1 1 ...
##  $ work     : num [1:55] 2 2 2 1 2 2 1 2 2 2 ...
##  $ madrs1   : num [1:55] 19 24 24 20 26 18 24 20 26 28 ...
##  $ madrs2   : num [1:55] 19 11 25 16 26 15 25 16 26 21 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   number = col_character(),
##   ..   days = col_double(),
##   ..   gender = col_double(),
##   ..   age = col_character(),
##   ..   afftype = col_double(),
##   ..   melanch = col_double(),
##   ..   inpatient = col_double(),
##   ..   edu = col_character(),
##   ..   marriage = col_double(),
##   ..   work = col_double(),
##   ..   madrs1 = col_double(),
##   ..   madrs2 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Remember that the bottom right quadrant is excellent for doing these same tasks, just not programatically. However, let's practice unzipping a file obtained from the internet!

```
# Assuming the ZIPPED csv file has been extracted and is in your working
# directory, unzip the downloaded file
unzip("kaggle_zipped_data.zip")

# Read the unzipped CSV file into a dataframe
library(readr)
dep_scores <- read_csv("scores.csv", show_col_types = FALSE)

# Make sure to check if your data frame was created and looks good!
str(dep_scores)
```

```
## spc_tbl_ [55 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
```

```
##  $ number   : chr [1:55] "condition_1" "condition_2" "condition_3" "condition_4" ...
##  $ days     : num [1:55] 11 18 13 13 13 7 11 5 13 9 ...
##  $ gender   : num [1:55] 2 2 1 2 2 1 1 2 2 2 ...
##  $ age      : chr [1:55] "35-39" "40-44" "45-49" "25-29" ...
##  $ afftype  : num [1:55] 2 1 2 2 2 2 1 2 1 2 ...
##  $ melanch  : num [1:55] 2 2 2 2 2 2 NA NA NA 2 ...
##  $ inpatient: num [1:55] 2 2 2 2 2 2 2 2 2 2 ...
##  $ edu      : chr [1:55] "6-10" "6-10" "6-10" "11-15" ...
##  $ marriage : num [1:55] 1 2 2 1 2 1 2 1 1 1 ...
##  $ work     : num [1:55] 2 2 2 1 2 2 1 2 2 2 ...
##  $ madrs1   : num [1:55] 19 24 24 20 26 18 24 20 26 28 ...
##  $ madrs2   : num [1:55] 19 11 25 16 26 15 25 16 26 21 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   number = col_character(),
##   ..   days = col_double(),
##   ..   gender = col_double(),
##   ..   age = col_character(),
##   ..   afftype = col_double(),
##   ..   melanch = col_double(),
##   ..   inpatient = col_double(),
##   ..   edu = col_character(),
##   ..   marriage = col_double(),
##   ..   work = col_double(),
##   ..   madrs1 = col_double(),
##   ..   madrs2 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Post-session REFLECTION:

I first want to make note of a mistake I made when setting working directories within an R Markdown file, and running the code through the console expecting the computer to know where the data is at all times. After speaking with GPT, it appears that I failed to establish a reliable method of knowing what directory I am pulling from (or at least, attempting to pull from).

One unexpected argument that I found myself using frequently was the 'include' and 'echo' arguments. If you choose TRUE for include, the output will contain the code chunk, whereas FALSE would omit that code. Same goes for 'echo', but this argument is referring to the OUTPUT being shown or not. Each function has their benefits in certain circumstances.

```
# KRUPALIS ADVICE subsetting data - casewise and listwise (by SID and by
# variable) - [,] syntax [r,c] substringing values, be able to remove the
# 'sub-' from our 'sub-XXXX' SIDs be able to import excel, csv, and text files
# into df merging datasets by SID/visit (full join, half join, left join, right
# join [dplyr]) packages (psych, plyr)
```