TEND_Merging_Data

Me

2023-05-24

R Markdown

I want to preface this markdown by first reflecting on some of the mistakes I made on my last Rmd on Importing Data. The biggest mistake I made was writing many lines and chunks of code without thinking about the working directory, or the packages that would later be required later on in the project. While we may not know the exact applications/packages we will be using, it is a good habit to load the packages at the top of your markdown or script:

Setting Working Directory for Rmd

```
knitr::opts_chunk$set(root.dir = "C:\\Users\\adcre\\OneDrive\\Desktop\\Desktop_RStudio", echo = TRUE)
getwd()
```

```
## [1] "C:/Users/adcre/OneDrive/Desktop/Desktop_RStudio"
```

Adding packages.

While I am really only interested in the 'tidytable' application in the 'tidyverse' package, In am adding these other packages just to show that this code chunk is very important to have at the top of your Rmd, because it will serve as a reference for you, the program, and anyone who is reading. This offers a lot of insight on your intentions when writing the code.

```
## -- 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
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
##
## Attaching package: 'psych'
##
##
## The following objects are masked from 'package:ggplot2':
##
```

```
##     %+%, alpha
##
##
##
## Please cite as:
##
##
##  Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.
##
##  R package version 5.2.3. https://CRAN.R-project.org/package=stargazer
```

Merging Data!

This markdown will contain code chunks for various situations in which you would like to merge data sets. Perhaps you would like to merge two data sets, one representing the brain-related data and the other representing the behavioral data for a group of participants. Or, maybe you would like to bind two similar data sets by rows or columns, depending on the composition of the data.

```
# Quick reminder: you can use 'T' instead of 'TRUE' hey serve the same purpose in TRUE/FALSE arguments
# First, we will load the data in. This should be well understood not only programmatically, but you sh
brain_data <- read.csv("Brain_Data.csv", header=T, sep=",", na.strings=c("NA", "888", "999"))
bx_data<-read.csv("Bx_Data.csv", header=T, sep=",", na.strings=c("NA", "888", "999"))

# Now we will merge, but make sure to assign a variable to this new data frame. let's call it merged_da
merged_data <- merge(brain_data, bx_data, "ID", all=T, no.dups = )
view(merged_data)
str(merged_data)
```

Our first code chunk will use the merge() function. In this example, i will use brain and behavior data from one group of subjects. If you look at my 'Importing Data' Rmd, you will see a lot of the same code! merge() comes from base R! With that being said, it is important to add arguments to this function to specify the way in which you want the data "merged", so to speak.

```
## 'data.frame':    55 obs. of  38 variables:
##  $ ID                            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Group.x                       : chr  "MDD" "MDD" "MDD" "MDD" ...
##  $ RSFC1                         : num  3.4 5.41 6.99 3.43 3.44 ...
##  $ RSFC2                         : num  3.04 3.79 3.51 4.02 3.52 ...
##  $ RSFC3                         : num  4.95 3.39 3.76 3.93 4.07 ...
##  $ RSFC4                         : num  3.33 3.88 3.11 3.51 4.13 ...
##  $ RSFC5                         : num  4.15 3.21 3.52 5.32 3.78 ...
##  $ SubjID                        : chr  "01-T1" "02-T1" "03-T1" "04-T1" ...
##  $ Group.y                       : chr  "MDD" "MDD" "MDD" "MDD" ...
##  $ Gender                        : int  2 1 2 1 2 2 2 2 3 1 ...
##  $ Sex                           : int  2 1 2 1 2 2 2 2 2 1 ...
##  $ Ethnicity                     : int  2 2 1 2 1 2 2 2 1 2 ...
##  $ Race                          : int  5 6 7 5 6 2 5 2 1 5 ...
```

```
##  $ Puberty                          : num  3.5 4 3.5 4.5 4 4 5 4.5 4.5 5 ...
##  $ Age_at_Bx                        : num  15.2 13.6 15.7 15.1 14.3 ...
##  $ Age_at_Scan                      : num  15.2 13.7 15.8 15.1 14.3 ...
##  $ Height                           : num  63.5 67.5 59.1 70.5 64 62.1 62 67 63.8 66.9 ...
##  $ Weight                           : num  167 107.6 91.4 125.4 179 ...
##  $ Bx_to_Scan_Days                  : int  12 8 29 7 7 5 11 7 14 13 ...
##  $ CDRSR_total                      : int  53 33 50 42 65 49 49 81 70 61 ...
##  $ CDRSR_tscore                     : num  70 58 68 64 80.5 67.5 67.5 86 83 77 ...
##  $ Current_Med                      : int  0 0 0 1 1 1 1 0 1 1 ...
##  $ Recurrent_Past_Month             : int  1 3 1 3 3 3 3 3 3 1 ...
##  $ Recurrent_Current_Duration_Weeks: int  NA 2 NA 156 364 156 312 8 676 NA ...
##  $ Age_First_SI                     : int  NA 12 13 13 10 12 14 13 8 NA ...
##  $ SI_Group                         : int  0 1 1 1 1 1 1 1 1 0 ...
##  $ SI_Past_Month                    : int  1 3 2 1 3 2 1 3 3 1 ...
##  $ Hx.Attempt                       : int  0 1 1 0 1 0 0 0 1 0 ...
##  $ Age.First.Attempt                : int  NA 13 15 NA 10 NA NA NA 10 NA ...
##  $ Number.of.Attempts               : int  0 2 1 0 3 0 0 0 5 0 ...
##  $ RADS_DM                          : int  26 26 25 25 27 30 27 32 32 30 ...
##  $ RADS_AN                          : int  11 19 13 18 10 19 13 25 16 16 ...
##  $ RADS_NS                          : int  16 32 21 25 29 25 21 29 29 19 ...
##  $ RADS_SC                          : int  21 22 18 18 28 23 17 26 26 22 ...
##  $ RADS_total                       : int  74 99 77 86 94 97 78 112 103 87 ...
##  $ PHQ9                             : int  19 14 18 16 19 21 16 23 23 21 ...
##  $ PHQ9_q9                          : int  1 3 0 1 3 2 1 2 2 2 ...
##  $ SIQ                              : int  10 50 17 42 34 75 34 74 52 11 ...
```

Now that we have explored a little bit with merge(), let's use rbind() and cbind(), which allow you to merge data frames by row or by column. We will start with rbind()

```r
# rbind() and cbind() are both part of base language, but have a lot of advantages in specific situatio
# rbind() would be useful in a scenario where you know you will not have any matching rows, so the df's

# Generate example dataset 1
set.seed(1)
data_1 <- data.frame(
  ID = 1:5,
  Value = rnorm(5)
)

# Generate example dataset 2
set.seed(2)
data_2 <- data.frame(
  ID = 6:10,
  Value = rnorm(5)
)

# Save the datasets as CSV files
write.csv(data_1, "data_1.csv", row.names = FALSE)
write.csv(data_2, "data_2.csv", row.names = FALSE)

# Now, we we will use rbind()
```

```r
# Read the datasets from CSV files
data_1 <- read.csv("data_1.csv")
data_2 <- read.csv("data_2.csv")

# Combine the datasets using rbind()
rbind_data <- rbind(data_1, data_2)

# View the combined dataset
str(rbind_data)
```

```
## 'data.frame':    10 obs. of  2 variables:
##  $ ID   : int  1 2 3 4 5 6 7 8 9 10
##  $ Value: num  -0.626 0.184 -0.836 1.595 0.33 ...
```

Let's use more complex functions! This code chunk in particular will be using the 'dplyr' package, which contains the 'dplyr' application. Within this application is functions like left_join, and many other functions that allow you to merge in more complex ways than the base language counterparts.

Before using 'mutating joins' in functions with regard to dplyr, which are a set of functions including full_join, half_join, left_join, and right_join. I will now introduce the '%>%' and the '::' operators.

```r
# '%>%' operates like a pipe, allowing you to make more readable, efficient, and concise code for dplyr
# Example data frame
data <- data.frame(
  id = c(1, 2, 3, 4, 5),
  name = c("Alice", "Bob", "Charlie", "David", "Eve"),
  age = c(25, 32, 40, 28, 35),
  city = c("San Diego", "Los Angeles", "Sacremento", "San Francisco", "Chico"),
  stringsAsFactors = FALSE
)

# Task: Filter and summarize data using traditional approach, and some useful operators in the last exa
filtered_data <- filter(data, age >= 30)
selected_data <- select(filtered_data, id, name, city)
summary_data <- data %>%
  mutate(age_group = ifelse(age >= 30, "30 and above", "Below 30")) %>%
  summarize(avg_age = mean(age))

# Check results
str(summary_data)
```

```
## 'data.frame':    1 obs. of  1 variable:
##  $ avg_age: num 32
```

```r
# '::' is used to access functions or objects from specific packages or namespaces. Ex using dplyr

# Create data frame 1
data1 <- data.frame(
  id = c(1, 2, 3, 4),
  value1 = c(10, 20, 30, 40)
)

# Create data frame 2
data2 <- data.frame(
  id = c(2, 3, 4, 5),
  value2 = c(100, 200, 300, 400)
)

# inner_join returns only the rows with matching "id" values from both data frames. Not suggested becau
inner_result <- dplyr::inner_join(data1, data2, by = "id")

# left_join returns all rows from the left data frame and the matching rows from the right data frame.
left_result <- dplyr::left_join(data1, data2, by = "id")

# right_join returns all rows from the right data frame and the matching rows from the left data frame.
right_result <- dplyr::right_join(data1, data2, by = "id")

# full_join returns all rows from both data frames, combining them based on the "id" column.
full_result <- dplyr::full_join(data1, data2, by = "id")

# You should feel comfortable with viewing your results!
str(left_result)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ id    : num  1 2 3 4
##  $ value1: num  10 20 30 40
##  $ value2: num  NA 100 200 300
```

DURING-Session REFLECTION

I want to keep a record of any significant mistakes or general concepts I was caught up with during the making of the document, so this section will serve as a journal, so to speak.The same will be done for the Post-Session REFLECTION. Since this is a new habit I want to gain, it will be very stream of consciousness and possibly disorganized. But the purpose is to retroactively understand my thought process when looking back over my Rmd's.Reflections or general comments will be made with 5 pound signs, as of may 23 2023.

When making comments within my code chunks, i will stop using the 'quotes' to refer to functions, but rather i will simply write the function and put parentheses() after, to show that it is a function. For example, "I wanted to use str() instead of View() because x, y, and z".

Johanna mentioned a really useful quick tip, and that is using command+Enter when hovering your mouse over a specific line of code. This will run just that line, and not the whole code chunk. This method is an alternative to highlighting the line and pressing 'Run Selected Line(s)', which is an option given when pressing the 'Run' button.

When making Rmd files, i wasn't paying enough attention to the font size i was using on specific comments. I am becoming more aware of how the size of your font expresses significance to a certain extent.

This became much more obvious when I used the 'Visual' tool in the top left quadrant of RStudio. This tool is very helpful for me because it lets me quickly visualize my code before I spend too much time writing something that is unnecessary, incorrect, or anything that could be aesthetically wrong.

GPT was incredibly helpful for understanding concepts and putting a magnifying glass on specific parts of the code that required my attention.

POST-Session REFLECTION

TBD..... :)