TEND_Group_Testing

Me

2023-05-28

R Markdown

Setting Working Directory

```
## here() starts at C:/Users/adcre/OneDrive/Documents/Desktop_RStudio
```

```
## here() starts at C:/Users/adcre/OneDrive/Documents/Desktop_RStudio
```

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

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

Adding libraries/packages

```
## -- 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 error
##
## 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
```

Load and Prepare Data

```r
Brain_Data <- read.csv("Brain_Data.csv", header=T, sep=",", na.strings=c("NA", "888", "999"))
Behavioral_Data<-read.csv("Bx_Data.csv", header=T, sep=",", na.strings=c("NA", "888", "999"))

# Merge the data sets
DATA <- merge(Brain_Data, Behavioral_Data, "ID", all=T)
View(DATA)

# The most common form of data is factors, numbers, strings and characters. They are all
# understood by R in different ways and certain functions need data to be in specific types
# to be understood correctly. Below are examples of rewriting the variable "ID" as a
# factor. And also creating a new variables Age as the numeric version of "Age.at.V1".
# To call a variable within a data set you say [name of dataset]$[name of varible],
# if the name of the variable does not exist, it will CREATE the variable in that data
# set. So in this case, "Age" did not exist, it was just Age.at.V1 and Age.at.V2 in our
# sheet, so instead we created a number version of "Age.at.V1" now called "Age"
DATA$ID<- as.factor(DATA$ID)
DATA$Age<- as.numeric(DATA$Age_at_Bx)

# It will be a very good habit to not only check what form your data or columns are
# represented as, it will be very useful to learn the functions that change your code
# into a more effective form.
#
# attach() and detach()
# If you have several variables you are manipulating and don't want to call the data
# frame they belong to each time ([data frame]$[varibale you want]) you can "attach"
# the data set so that R assumes all variables you call until you "detach" it will be
# in the data frame you are looking at. Below is an example of scoring a measurement
# of several components. Instead of having to write "DATA$RADS_DM" etc. for each subscale
# we are adding, we can just write the variable name. THIS WAS COPY+PASTED FROM
# JOHANNA's Rmd
attach(DATA)
DATA$RADS_total_rescore<- RADS_DM + RADS_AN + RADS_NS + RADS_SC
detach(DATA)
View(DATA)
# This would be useful in a situation where you are constantly having to call a column
# that is nested within a variable, or multiple. This could get confusing when you have
# performed significant exploration and have many variables in your environment

# attach() and detach()
# If you have several variables you are manipulating and don't want to call the data
# frame they belong to each time, you can use attach() and detach() to tell the software
# to call only to the data frame you specify. This prevents you from having to type out
# the long code that calls to a specific column nested in a data set
attach(DATA)
DATA$RADS_total_rescore<- RADS_DM + RADS_AN + RADS_NS + RADS_SC
detach(DATA)

# recode() COPY+PASTED FROM JOHANNA'S Rmd
# #Here is how you can easily recode something using the "recode" function, the first
# value will be what the data currently is, and the second value is what you now want
# it to be. For example, we want to round up all of the scores in one column or variable,
# 'Puberty_rounded'.
DATA$Puberty_rounded <- recode(DATA$Puberty, '1.5' = 2.0, '2.5' = 3.0, '3.5' = 4.0,
                               '4.5' = 5.0)
# Dplyr has some very useful functions as well, but this is useful to know

# creating new variable and column
# This is how you can add a new variable with NA values
DATA$attempt_reason<- NA

# assigning NA to conditions in a column
# By using the variable assignment operator '<-', we can assign NA avlues to a specified
# column in a df, and add conditions so that only subjects with scores below 21, for
# example, get reassigned to NA
DATA$Bx_to_Scan_Days[DATA$Bx_to_Scan_Days > 21] <- NA

# rename()
# If you wanted to rename variables or names of columns, you can change them with
# rename(), even doing multiple rows simultaneously
DATA_Final<- rename(DATA,"Hx_Attempt" = "Hx.Attempt", "Age_First.Attempt" = "Age.First.Attempt","Number

# Let's now save our new data sheet using the writexl library
library(writexl)
write_xlsx(DATA_Final, "Example_DATA_Export.xlsx")
```

Quick Check!

Johanna gave a very useful tip for quickly checking your data (certain columns, variables) before running anlyses or continuing with your exploratory analysis. Using View() and data.frame() in conjuction with one another gives you a quick glimpse at a quickly-made df of your specified columns.

```r
# This is a great way of double-checking that your preparation hasn't changed the data.
# Think of it as a way of looking twice before turning!
View(data.frame(DATA_Final$ID, DATA_Final$CDRSR_total, DATA_Final$Sex, DATA_Final$Puberty))
```

Start Exploring

```r
# First we will use colnames() to see the names of the variables of our data set
# Assign the colnames to a variable through a data frame, to be accessed later saves
# this as a data frame
Column_indexes <- data.frame(colnames(DATA))
str(Column_indexes)
```

```
## 'data.frame':    42 obs. of  1 variable:
##  $ colnames.DATA.: chr  "ID" "Group.x" "RSFC1" "RSFC2" ...
```

```r
# which() allows us to know the position of a specific variable, ex "Gender" is our
# 10th column variable
which(colnames(DATA) == "Gender")
```

```
## [1] 10
```

```
# rowMeans()
# We will now use rowMeans() to get the average of multiple variables. rowMeans can
# also have the 'na.rm' logical argument, which, when TRUE, will go through your data
# and ignore any rows that have NAs in your specified columns.As you can
# see, we assign a new column with the '<-' and '$'
DATA$RADS_average<- rowMeans(DATA[,c("RADS_DM", "RADS_AN",  "RADS_NS",  "RADS_SC")],
                             na.rm=TRUE)

# WIDE vs LONG format CONVERSION , common in longitudinal studies/data

# Long format will mean there are several rows per participant that are differentiated
# by time point. Wide is when you have one row per participant and the repeated measures
# and different column. Certain functions require the longitudinal data to be in wide or
# long format.
# Here is an example of a simple reshape from long to wide where time points already
# designated. For the sake of an example, let's say we called for View() after reading
# in our data, and we realize that our data is in long format and it needs to be in wide.
## CCTG_Data_Long <- read.csv("fake_longitudinal_data.csv", header=T, sep=",", na.strings=c("NA", "888"

## CCTG_Data_Wide <- reshape(CCTG_Data_Long, idvar = "pid", timevar = "visit_cycle", direction = "wide"

# FORLOOPS !!!

# Complex example of reshape where "instances" of a measure need to be assigned with
# for loop!
# Let's break down each point in this long function.Let's say our problem is that we
# are trying to convert long to wide, but we don't have a 'marker' or way of assigning
# our data to a certain visit, whether it be their first or fourth. In order to convert
# to wide format, the computer needs to know

# reading in data
## Contact_Log_Long <- read.csv("fake_contact_log.csv", header=T, sep=",",
##                              na.strings=c("NA", "888", "999"))
## turning Contact_Num into missing values
## Contact_Log_Long$Contact_Num <- NA

## for(x in 1:length(Contact_Log_Long$pid)){
##   if (x == 1)
##     {Contact_Log_Long$Contact_Num[x] = 1}
##  else if (Contact_Log_Long$pid[x] == Contact_Log_Long$pid[x-1])
##  {Contact_Log_Long$Contact_Num[x] = Contact_Log_Long$Contact_Num[x-1] + 1}
##  else if (Contact_Log_Long$pid[x] != Contact_Log_Long$pid[x-1])
##  {Contact_Log_Long$Contact_Num[x] = 1}
##  }

## MI_Data_Wide <- reshape(Contact_Log_Long, idvar = "pid", timevar = "Contact_Num", direction = "wide"
```

mutate()

```
# coming from dplyr, the primary purpose of mutate() is to add new columns to a data
# frame based on calculations or transformations applied to existing columns. It allows
# you to perform various operations, such as mathematical calculations, conditional
# operations, string manipulations, and more. LEARN THIS FUNCTION
```

DURING-Session REFLECTION

POST-Session REFLECTION

TBD..