TEND_Group_Testing

Me

2023-05-28

R Markdown

Setting Working Directory

```
# Trying here(), which will also set working directory based off your Rmd file
library(here)
```

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

```
here::i_am("TEND_Group_Testing.Rmd") # tells you full file path script's current location
```

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

```
here() # returns file path of where the script is current saved
```

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

```
setwd(here()) # sets the working directory to be wherever your source file or Rmd is

getwd() # why not double check?
```

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

Adding libraries/packages

```
library(tidyverse)
```

```
## -- 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
```

```
# Loads 'ggplot2' for data visualization. 'dplyr' for data manipulation, 'tidyr' for
# data tidying, 'readr' for data import, 'purrr' for functional programming, 'tibble'
# for (tibbles) a modern re-imagining of data frames, 'stringr' for strings
library(writexl)
# Allows yoyu to save an explored data fram as an Excel file. It will go to your wd.
library(foreign)
# Read data stored by SPSS and Stata
library(psych)
```

```
##
## Attaching package: 'psych'
##
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
# For personality, psychometric, and psychological research, includes describe function
# and error bars
library(stargazer)
```

```
##
## 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
```

```
# Handy regression tables
library(readr)
```

Load and Prepare Data

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.

```
# 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

```
# 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" ...
```

```
# 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", "999"))

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

FORLOOPS !!!

forloop example

```
# First, we create the necessary variables to construct the long format data frame.
# 'subID' is a vector that represents exactly what you think, subjects!
subID <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
# 'vis_ID' is a vector that represents exactly what you think, events!
vis_ID <- c("Event1", "Event2", "Event3")
# Random values for each event, rnorm() creates a random generation of values, which can
# be manipulated by conditions or arguments
ran_values <- rnorm(9)

# Now we use data.frame() to create the data frame for practice.
practice_long_df <- data.frame(SubID = rep(subID, each = 3),
                               Vis_Number = rep(vis_ID, times = 3),
                               Value = ran_values)
```

```
View(practice_long_df)

# Create a function to add visit number based on subject ID
# unique() is used to extract the unique elements from a vector or column of a data
# frame. It will essentially go through the rows and find the 'distinct elements', or
# the values that may or not be duplicated, but exist in the rows. It essentially makes
# a call to return a vector with no dups
# we use a for loop to iterate over each unique subject ID and assign visit numbers to
# the corresponding rows in the 'VisitNumber' column
add_visit_number <- function(data) {
  unique_subjects <- unique(data$SubID)
  for (subject in unique_subjects) {
    subject_rows <- data$SubID == subject
    data$Vis_Number[subject_rows] <- 1:length(data$Vis_Number[subject_rows])
  }
  return(data)
}

# Add visit number using the function we previously defined. This will put the data
# frame in the proper orientation to convert to wide format
practice_long_df <- add_visit_number(practice_long_df)
# Remove incomplete rows
practice_long_df <- practice_long_df[complete.cases(practice_long_df), ]
View(practice_long_df)

# Convert to wide format using reshape()
practice_wide_df <- reshape(practice_long_df, idvar = "SubID", timevar = "Vis_Number",
                            direction = "wide")
View(practice_wide_df)
```

Due to the importance of understanding forloops beyond the input-output, I will perform a similar function to the one that Johanna made, but on a different data frame either made by me or found online (or... both?)

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..