TEND_Subsetting_Data

Me

2023-05-28

R Markdown

```
library(here)
```

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

```
#tells you full file path of script in it current location
here::i_am("TEND_Subsetting_Data.Rmd")
```

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

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

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

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

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

R Markdown

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 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
```

Subsetting

```
# We will first create a df that allows us to visualize subsets.
# Create a data frame
df <- data.frame(
  Name = c("John", "Emily", "David", "Sarah", "Michael"),
  Age = c(25, 32, 28, 35, 30),
  City = c("New York", "London", "Paris", "Tokyo", "Sydney")
)

# Print the data frame
print(df)
```

The first code chunk will be examples of subsetting on a simple data structure that is created within the chunk. In the interest of truly being able to apply these skills, I will use a df imported from Kaggle after this code chunk in order to test it in a real-life situation.

```
##      Name Age     City
## 1    John  25 New York
## 2   Emily  32   London
## 3   David  28    Paris
## 4   Sarah  35    Tokyo
## 5 Michael  30   Sydney
```

```
# Notice the use of brackets[] and a dollar sign$ to indicate that you want to subset,
# and that you would like to subset a specific column, respectively. In this example,
# we also pass a condition that the Age must be greater than 30.In other words, I want
# to see the data for participants over 30 years old.

# Subset the data frame by age, specifically ppl over the age of 30
subset_df <- df[df$Age > 30, ]

# Print the subsetted data frame
print(subset_df)
```

```
##    Name Age   City
## 2 Emily  32 London
## 4 Sarah  35  Tokyo
```

Quick note: is.na()

a logical function that checks the given the data structure and returns TRUE if there are missing values

na.rm()

```
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"))
DATA <- merge(Brain_Data, Behavioral_Data, "ID", all=T)

# Now, we will use na.rm(), which effectively reads over the data and still averages
# the variables that are there by requiring that it only averages values that are "numeric". We use
# c() within our subset to average across multiple columns
DATA$RADS_average<- rowMeans(DATA[,c("RADS_DM", "RADS_AN",  "RADS_NS",  "RADS_SC")], na.rm=TRUE)
```

creating NAs

```
# This is one way to specifically change certain data into NA. More specifically, I want
# to change any missing values into NAs that are greater than 21. Other conditions can
# be made, of course.
DATA$Bx_to_Scan_Days[DATA$Bx_to_Scan_Days > 21] <- NA

## If you want to take out a specific variable (like an outlier). You can use "which"
## or "grep" (grep additionally allows for partial variable matches). grep() was used
## because it will catch patterns, this one being '107', the subID we are interested
## in removing. This is one of many ways to solve the same problem
badid <- grep('107', DATA$ID)
# assign variable to the subsetted DATA, which is essentially your data w/out outlier
# was specified in the former line.
DATAsub <- DATA[-badid,]
```

Removing outliers w grep()

```
# grep() is being used here to remove an outlier, assuming we know the
# characters/numbers etc of the data.
# In this example, we know that case (or row) 107 is an outlier, a piece of data that
# has an unwanted effect on our descriptive stats, such as the mean
badid <- grep('107', DATA$ID)
DATAsub <- DATA[-badid,]
```

Renaming Variables

```
# If you are wanting to rename multiple variables all a the same time you can use the
# following rename()
DATA_Final <- rename(DATA,"Hx_Attempt" = "Hx.Attempt", "Age_First.Attempt" =
                        "Age.First.Attempt","Number_of_Attempts" = "Number.of.Attempts")
```

Operators

```
# first, let's look at is.na(), '&', and '!' in action on the TEND example data
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"))
DATA <- merge(Brain_Data, Behavioral_Data, "ID", all=T)

# With this line of code, we are extracting all of the cases that did not have missing
# values in the specified column.
SUBdata<- subset(DATA, !is.na(DATA$CDRSR_total) & !is.na(DATA$RSFC1))
View(SUBdata)
```

operators like '!' and '&' can be useful when used properly, like in conjuction with functions within a code chunk. While they don't necessarily do anything powerful, they are very useful when used !

Let's try subsetting by variables and conditions on a more realistic data set found on Kaggle. We will explore the same data set found in my "Importing_Data_Sets" repo, the Depression Dataset found at this link on Kaggle: https://www.kaggle.com/datasets/arashnic/the-depression-dataset

```
# First, I will load in the scores from the selected dataset. It can be done many
# ways, and should be well understood at this point.
dep_scores <- read_csv("scores.csv")
```

This data set is represented as 'scores.csv'in my Files

```
## Rows: 55 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): number, age, edu
## dbl (9): days, gender, afftype, melanch, inpatient, marriage, work, madrs1, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
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>
```

```
# By using str(), I can view the different columns by name. This is essentially a
# list of the variables that you can subset by, and we will try our best to
# navigate through this data via subsetting and adding conditions. Off the bat,
# we notice that there is indeed an age column. HOWEVER, it is actually in
# character form, not numeric!! This is because the creator of this data set has
# made columns that represent age ranges rather than putting the age as a column
# for any number. This slightly changes the code we will use to subset certain
# ages, but we will be doing effectively the same job.
dep_subset <- dep_scores[dep_scores$age == "40-44", ]
print(dep_subset)
```

```
## # A tibble: 5 x 12
##   number        days gender age   afftype melanch inpatient edu   marriage  work
##   <chr>        <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
## 1 condition_2     18      2 40-44       1       2         2 6-10         2     2
## 2 condition_12    12      2 40-44       1       2         2 6-10         2     2
## 3 condition_18    13      2 40-44       3       2         2 11-15        2     2
## 4 control_8       13      2 40-44      NA      NA        NA <NA>        NA    NA
## 5 control_16      13      2 40-44      NA      NA        NA <NA>        NA    NA
## # i 2 more variables: madrs1 <dbl>, madrs2 <dbl>
```

```
# Notice that we used '==" instead of the logical condition '<' or '>'. Rather than
# calling for subjects older than 30 years, we are calling for an age range 40-44 by
```

```r
# calling for the specific character string '40-44'.
#
# What if I wanted to call for a specific ID? There is a 'number' column, and each
# subject is either represented by 'condition_X' or 'control_X', for example. This
# will actually be done by the same syntax as the age range subset example above,
# because they are represented by character strings.
test_subject1 <- dep_scores[dep_scores$number == "condition_1", ]
control_subject1 <- dep_scores[dep_scores$number == "control_1", ]

View(control_subject1)
View(test_subject1)

# After looking further into the data, I noticed that the 'gender' column has 1's
# and 2's, 1 presumably representing male subjects. This allows us to create a new
# variable, 'male_subjects' for example.
male_subject_dep <- dep_scores[dep_scores$gender == 1, ]

# Let's go one step further and subset by age AND gender by using the '&' operator
# in the brackets.Make sure to assign variables to your subsets, so that they can be
# referenced later in the code.
new_subset <- dep_scores[dep_scores$gender == 2 &
                           dep_scores$age == "45-49", ]
View(new_subset)
```

This next code chunk will look into subsetting list-wise and case-wise!

What are the key differences between these methods of subsetting, and when would you use them? To quote GPT-4, "In summary, case-wise subsetting involves selecting specific rows based on conditions, while list-wise subsetting involves selecting specific columns."

```r
# Case-wise refers to subsetting for a specific cases, or ROWS, of the data frame. You
# could set criteria for the rows you would like returned, or specify one case you are
# interested in
# List-wise, on the other hand, is for when we would like to include all of the data
# from specific variables, or columns. Almost as if you chop down unwanted columns from
# the data frame to create a new, slimmer one.

# Let's use the Kaggle depression data set
dep_scores <- read_csv("scores.csv")
```

```
## Rows: 55 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): number, age, edu
## dbl (9): days, gender, afftype, melanch, inpatient, marriage, work, madrs1, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
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>
```

```r
# First, we will look at CASE-WISE subsetting. There is a 'days' column, which we will
# assume refers to the number of days that any given subject has been in the study.
# Under this assumption, we will subset the subjects who have only been in the study
# for at least 2 weeks
subset_twowksubs <- dep_scores[dep_scores$days > 13, ]
View(subset_twowksubs)
# Now, it will return cases that are 2 weeks old or older! Thus, case-wise subsetting

# It's time to look at LIST-WISE subsetting, which is more focused on certain columns,
# or variables, within your df. Our Kaggle data set contains many different variables,
# allowing us to explore this method.When using this method, we use the c() function
# which stands for 'concatenate', or join together. Essentially, it is a function that
# allows you to create a vector or string of columns, the ones you are most interested
# in seeing.

# You will notice that we start the [] section of the following code with a ','. This
# simply means that we will be including all of the rows in the data set. The same
# goes for the opposite regarding columns, as you may have seen earlier.
listwise_subs <- dep_scores[ , c("age", "afftype", "edu")]
View(listwise_subs)
```

```
# For our Kaggle data set, there appears to be two prefixes in the 'number' column.
# This chunk will show you how to subset the 'condition_' prefix group and the
#'control_' group out of the data set. This might serve useful later on!**
dep_scores <- read_csv("scores.csv")
print(dep_scores)
```

grep! this function is from base R, and is used when pattern maqtching is needed, in this case, for a subset of prefixed rows.

```
## # A tibble: 55 x 12
##    number       days gender age   afftype melanch inpatient edu   marriage  work
##    <chr>       <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
##  1 condition_1    11      2 35-39       2       2         2 6-10         1     2
##  2 condition_2    18      2 40-44       1       2         2 6-10         2     2
##  3 condition_3    13      1 45-49       2       2         2 6-10         2     2
##  4 condition_4    13      2 25-29       2       2         2 11-15        1     1
##  5 condition_5    13      2 50-54       2       2         2 11-15        2     2
##  6 condition_6     7      1 35-39       2       2         2 6-10         1     2
##  7 condition_7    11      1 20-24       1      NA         2 11-15        2     1
##  8 condition_8     5      2 25-29       2      NA         2 11-15        1     2
##  9 condition_9    13      2 45-49       1      NA         2 6-10         1     2
## 10 condition_~     9      2 45-49       2       2         2 6-10         1     2
## # i 45 more rows
## # i 2 more variables: madrs1 <dbl>, madrs2 <dbl>
```

```
# Subset only the "condition" rows
condition_df <- dep_scores[grep("^condition_", dep_scores$number), ]

# Subset only the "control" rows
control_df <- dep_scores[grep("^control_", dep_scores$number), ]

# Check the subsetted data frames
str(condition_df)
```

```
## tibble [23 x 12] (S3: tbl_df/tbl/data.frame)
##  $ number   : chr [1:23] "condition_1" "condition_2" "condition_3" "condition_4" ...
##  $ days     : num [1:23] 11 18 13 13 13 7 11 5 13 9 ...
##  $ gender   : num [1:23] 2 2 1 2 2 1 1 2 2 2 ...
##  $ age      : chr [1:23] "35-39" "40-44" "45-49" "25-29" ...
##  $ afftype  : num [1:23] 2 1 2 2 2 2 1 2 1 2 ...
##  $ melanch  : num [1:23] 2 2 2 2 2 2 NA NA NA 2 ...
##  $ inpatient: num [1:23] 2 2 2 2 2 2 2 2 2 2 ...
##  $ edu      : chr [1:23] "6-10" "6-10" "6-10" "11-15" ...
##  $ marriage : num [1:23] 1 2 2 1 2 1 2 1 1 1 ...
##  $ work     : num [1:23] 2 2 2 1 2 2 1 2 2 2 ...
##  $ madrs1   : num [1:23] 19 24 24 20 26 18 24 20 26 28 ...
##  $ madrs2   : num [1:23] 19 11 25 16 26 15 25 16 26 21 ...
```

```
str(control_df)
```

```
## tibble [32 x 12] (S3: tbl_df/tbl/data.frame)
```

```
##  $ number   : chr [1:32] "control_1" "control_2" "control_3" "control_4" ...
##  $ days     : num [1:32] 8 20 12 13 13 13 13 13 13 8 ...
##  $ gender   : num [1:32] 2 1 2 1 1 1 1 2 2 1 ...
##  $ age      : chr [1:32] "25-29" "30-34" "30-34" "25-29" ...
##  $ afftype  : num [1:32] NA NA NA NA NA NA NA NA NA NA ...
##  $ melanch  : num [1:32] NA NA NA NA NA NA NA NA NA NA ...
##  $ inpatient: num [1:32] NA NA NA NA NA NA NA NA NA NA ...
##  $ edu      : chr [1:32] NA NA NA NA ...
##  $ marriage : num [1:32] NA NA NA NA NA NA NA NA NA NA ...
##  $ work     : num [1:32] NA NA NA NA NA NA NA NA NA NA ...
##  $ madrs1   : num [1:32] NA NA NA NA NA NA NA NA NA NA ...
##  $ madrs2   : num [1:32] NA NA NA NA NA NA NA NA NA NA ...
```

ifelse() *******

```r
# Here we are creating a new variable "Current_Med_String" to have strings of what was
# in numeric 1s (yes) and 0s (no) of whether someone was current on medications. This
# is an if else statement, It will look at all rows checking if "current_med" is equal
# to ("==") 1. if that is TRUE, "Current_Med_String" for that row will not equal "Meds",
# if it is not true (FALSE), it will return its ELSE statement "noMeds".
DATA$Current_Med_String <- ifelse(DATA$Current_Med == 1, "Meds", "noMeds")
```

Subsetting Alternatives

```r
# This code chunk contains many different ways to extract rows and columns with more
# simple syntax.
dep_scores[,] # all rows and all columns
```

Below are some very useful examples that Johanna offered as alternatives to the more complex syntax or functions previously mentioned

```
## # A tibble: 55 x 12
##    number       days gender age   afftype melanch inpatient edu   marriage  work
##    <chr>       <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
##  1 condition_1    11      2 35-39       2       2         2 6-10         1     2
##  2 condition_2    18      2 40-44       1       2         2 6-10         2     2
##  3 condition_3    13      1 45-49       2       2         2 6-10         2     2
##  4 condition_4    13      2 25-29       2       2         2 11-15        1     1
##  5 condition_5    13      2 50-54       2       2         2 11-15        2     2
##  6 condition_6     7      1 35-39       2       2         2 6-10         1     2
##  7 condition_7    11      1 20-24       1      NA         2 11-15        2     1
##  8 condition_8     5      2 25-29       2      NA         2 11-15        1     2
##  9 condition_9    13      2 45-49       1      NA         2 6-10         1     2
## 10 condition_~     9      2 45-49       2       2         2 6-10         1     2
## # i 45 more rows
## # i 2 more variables: madrs1 <dbl>, madrs2 <dbl>
```

```r
dep_scores[1,] #first row and all columns
```

```
## # A tibble: 1 x 12
##   number       days gender age   afftype melanch inpatient edu   marriage  work
##   <chr>       <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
## 1 condition_1    11      2 35-39       2       2         2 6-10         1     2
## # i 2 more variables: madrs1 <dbl>, madrs2 <dbl>
```

```r
dep_scores[1:10,] #first 10 rows and all columns
```

```
## # A tibble: 10 x 12
##    number       days gender age   afftype melanch inpatient edu   marriage  work
##    <chr>       <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
##  1 condition_1    11      2 35-39       2       2         2 6-10         1     2
##  2 condition_2    18      2 40-44       1       2         2 6-10         2     2
##  3 condition_3    13      1 45-49       2       2         2 6-10         2     2
##  4 condition_4    13      2 25-29       2       2         2 11-15        1     1
##  5 condition_5    13      2 50-54       2       2         2 11-15        2     2
##  6 condition_6     7      1 35-39       2       2         2 6-10         1     2
##  7 condition_7    11      1 20-24       1      NA         2 11-15        2     1
##  8 condition_8     5      2 25-29       2      NA         2 11-15        1     2
##  9 condition_9    13      2 45-49       1      NA         2 6-10         1     2
## 10 condition_~     9      2 45-49       2       2         2 6-10         1     2
## # i 2 more variables: madrs1 <dbl>, madrs2 <dbl>
```

```r
dep_scores[c(1,3),1:10] #first and third row and the first 10 columns
```

```
## # A tibble: 2 x 10
##   number       days gender age   afftype melanch inpatient edu   marriage  work
##   <chr>       <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
## 1 condition_1    11      2 35-39       2       2         2 6-10         1     2
## 2 condition_3    13      1 45-49       2       2         2 6-10         2     2
```

Substringing

```r
# To stay consistent, we will first create a simple data set wi9thin our code to visualize
# and practice substringing. After that, we will explore the Kaggle depression data set.
# Substr() takes three arguments: the character vector being modified, the starting
# position, and the ending position. the starting/ending position i8s pre-determined by
# the specific range/position of the charcter vector you are interested in. In this case,
# we want to start at'5' and end at '8'.

#First, create the df
substr_df <- data.frame(
  ParticipantID = c("sub_1234", "sub_5678", "sub_9101", "sub_2345"),
  Score = c(15, 20, 18, 22),
  stringsAsFactors = FALSE
)
```

```
# Remove the prefix with substr(), no need to assign a variable!
substr_df$ParticipantID <- substr(substr_df$ParticipantID, start = 5, stop = 8)
print(substr_df)
```

Let's say we had a data frame, but in our subject's row3, there is a prefix called "sub_". This means that each case is represented as 'sub_XXXX'. If I wanted to remove that prefixd, or any prefixes for that matter, I would use substringing! Substringing is useful when you know the specific ranges or positions that you want to call for. In this case, we know that the last 4 characters are the ID, and we don't need the prefix. This code will show you how to solve this problem, so that subjects are only represented by integers, not a prefix as well.

```
##   ParticipantID Score
## 1          1234    15
## 2          5678    20
## 3          9101    18
## 4          2345    22
```

```
dep_scores <- read_csv("scores.csv")
```

Now that we have looked at a basic example of substringing to remove unwanted prefixes, let's try it on our Kaggle depression data set

```
## Rows: 55 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): number, age, edu
## dbl (9): days, gender, afftype, melanch, inpatient, marriage, work, madrs1, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
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>
```

```
# Using str() allows me to peek at the names of the rows that pertain to certain subjects.
# What I notice about the Kaggle data set is that there is a prefix "condition_" that
# comes before each subject id, as well as a "control_" prefix. I will provide the
# method of using sub() and substr().

# First, the method using sub()
dep_scores$number <- sub("^condition_|^control_", "", dep_scores$number)
print(dep_scores)
```

```
## # A tibble: 55 x 12
##    number  days gender age   afftype melanch inpatient edu   marriage  work
##    <chr>  <dbl>  <dbl> <chr>   <dbl>   <dbl>     <dbl> <chr>    <dbl> <dbl>
##  1 1         11      2 35-39       2       2         2 6-10         1     2
##  2 2         18      2 40-44       1       2         2 6-10         2     2
##  3 3         13      1 45-49       2       2         2 6-10         2     2
##  4 4         13      2 25-29       2       2         2 11-15        1     1
##  5 5         13      2 50-54       2       2         2 11-15        2     2
##  6 6          7      1 35-39       2       2         2 6-10         1     2
##  7 7         11      1 20-24       1      NA         2 11-15        2     1
##  8 8          5      2 25-29       2      NA         2 11-15        1     2
##  9 9         13      2 45-49       1      NA         2 6-10         1     2
## 10 10         9      2 45-49       2       2         2 6-10         1     2
## # i 45 more rows
## # i 2 more variables: madrs1 <dbl>, madrs2 <dbl>
```

```
# I want to emphasize that we only practiced that method for the sake of practice. It
# may be counterproductive to remove the prefixes because they offer info about the
# subject data that could be crucial later on. Think before using this method what the
# programmatic implications are.
```

DURING-Session REFLECTION

I want to thank Krupali, who helped me find my way when navigating this software. I was in need of some short-term goals, and she helped contextualize RStudio in a way that inspired me to get back to the computer and keep going! Thank you, Krupali!

I need to look more into ifelse() statements and ordering/levelling

I noticed myself deviating from Johanna's videos and using the internet for some troubleshooting which I think is totally fine, however it ended up costing me time and cognitive load, because had I finished watching the video from Week 2, I wouldv'e learned many functions or concepts that I had trouble understanding. Instead of following her videos or even the Rmd files, I was just asking the internet for answers to questions I didn't end up needing an answer to. While I don't think it isn wise to follow only one source of content, I must realize that these skills are only useful if they make sense in your specific environment, TEND lab. If I receive advice from the internet, they won't understand the full context of the probloems i am trying to solve. THAT, is why I believe following the recordings/markdown/schedule will allow me to set realistic, useful, and short-term goals for learning RStudio and how to be of use to my peers!

My within-codechunk comments are becoming very long, and I ran into a problem when knitting that made the comments continue running off the page. While this wasn't a problem, it made me think about my comments in general and i think they are running long. Don't add words just to add words. the more effective you are with your words, the more emphasis you can put on certain ideas.

Margin columns, soft-wrapping, and other aesthetic issues came up when knitting during the making of this markdown, and I seem to understand it a lot better. However, we haven't even started diving deep on data visualization, where I imagine this problem is only exacerbated and more cause for attention/concern

POST-Session REFLECTION

TBD....