TEND_Subsetting_Data

Me

2023-05-26

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

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

R Markdown

This markdown will serve as a reference sheet for different subsetting methods, packages, applications, and more. More specifically, how to subset data casewise, listwise, by ID, or

by variable. Having a large and complex df could prevent you from gaining insight about specific data. Having the ability to call for certain ID numbers or variables, like gender or age, allows you to use a magnifying glass on your large df. I also plan on learning and practicing substringing values, and being able to remove the "sub" from "sub-xxxx" in the subject ID column. The key differences between substringing and subsetting is that substringing is applied to character strings like text files, and can output specific ranges or positions in a long character string. A subset is more widely used because it can be applied to data frames and other data structures. Based off of certain arguments or criteria, a subset can give you a closer look on specific points in your data structure. Subsets are represented by brackets[], and more of the syntax will be learned with time and practice.

Subsetting

```
# We will first create a df that allows us to visualize subsetting, and learn the syntax used to call f

# 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 wou

# 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

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 sp
SUBdata<- subset(DATA, !is.na(DATA$CDRSR_total) & !is.na(DATA$RSFC1))
View(SUBdata)
```

operaqtors 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 w
dep_scores <- read_csv("scores.csv", show_col_types = FALSE)
str(dep_scores)
```

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

```
## 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
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 '>'. So, rather than calling for sub
# What if I wanted to call for a specific ID? There is a 'number' column, and each subject is either re
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
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
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 criter
# List-wise, on the other hand, is for when we would like to include all of the data from specific vari

# Let's use the Kaggle depression data set from the previous chunk to observe these subset methods.
dep_scores <- read_csv("scores.csv", show_col_types = FALSE)
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
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 columms, or variables, wi

# You will notice that we start the [] section of the following code with a ','. This simply means that
listwise_subs <- dep_scores[ , c("age", "afftype", "edu")]
View(listwise_subs)
```

```r
# For our Kaggle data set, there appears to be two prefixes in the 'number' column. This chunk will sho
dep_scores <- read_csv("scores.csv", show_col_types = FALSE)
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 ...
```

Subsetting Alternatives

```
# 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>
```

```
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>
```

```
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>
```

```
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

```
# To stay consistent, we will first create a simple data set wi9thin our code to visualize and practice
# Substr() takes three arguments: the character vector being modified, the starting position, and the e

#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", show_col_types = FALSE)
str(dep_scores)
```

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

```
## 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 notic

# 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 practed that method for the sake of practice. It may be counterprodu
```

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 gouing! Thank you, Krupali!

POST-Session REFLECTION

TBD....