object of type closure is not subsettable

rstd.io/debugging

Jennifer Bryan

RStudio

@jennybc

@JennyBryan

```r
dat ← data.frame(x = 1, y = 2)

df$x
#> Error in df$x: object of type 'closure' is not subsettable

cat("Perhaps you wanted `dat$x`?\n")
#> Perhaps you wanted `dat$x`?
```

Fun!


Fun!


Less fun.

Poll at slido.com (event code = **HEXAGON**)

WHAT'S YOUR MAIN DEBUGGING METHOD?

A. I don’t have a method. Why do you think I’m here?!?

B. I ask in an online forum or consult my local expert.

C. Google is my BFF.

D. Print debugging is my jam.

E. I use R's debugging tools, like `browser()`.

1. RESET

2. REPREX

3. DEBUG

4. DETER

1 RESET

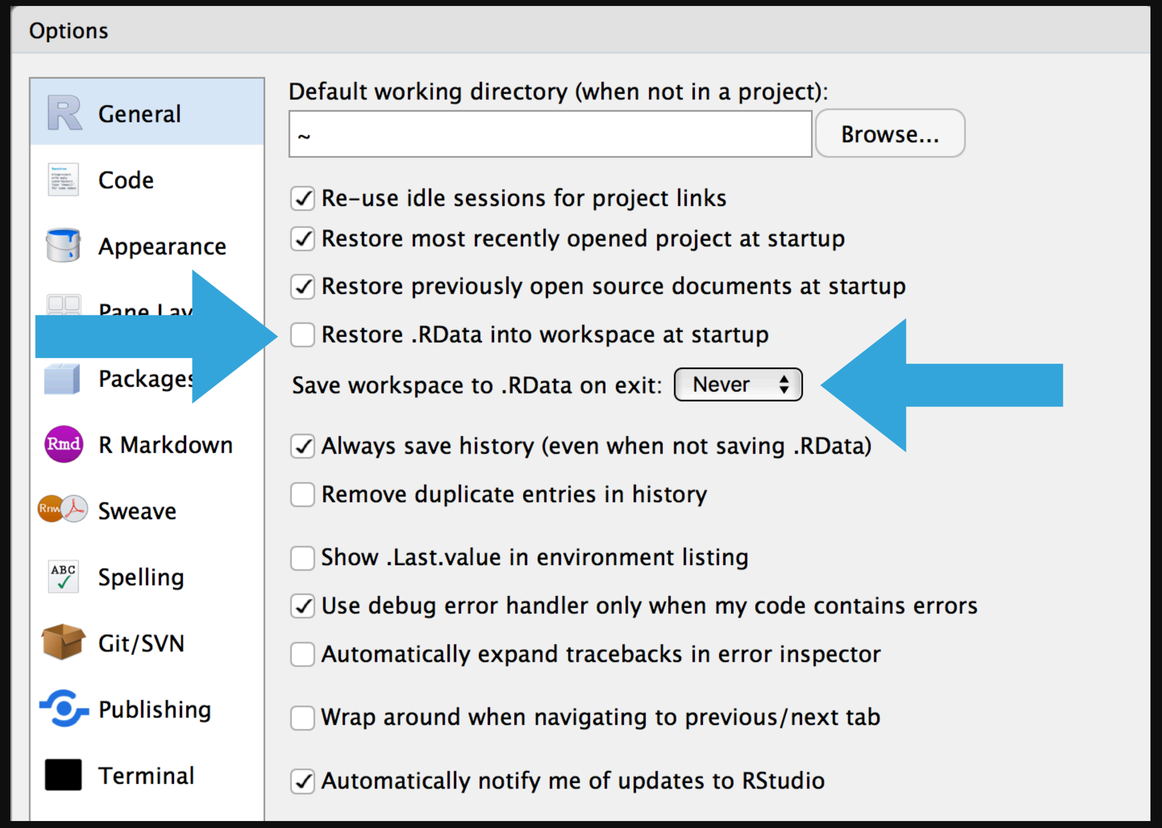

If at first you don’t succeed...
try exactly the same thing again.

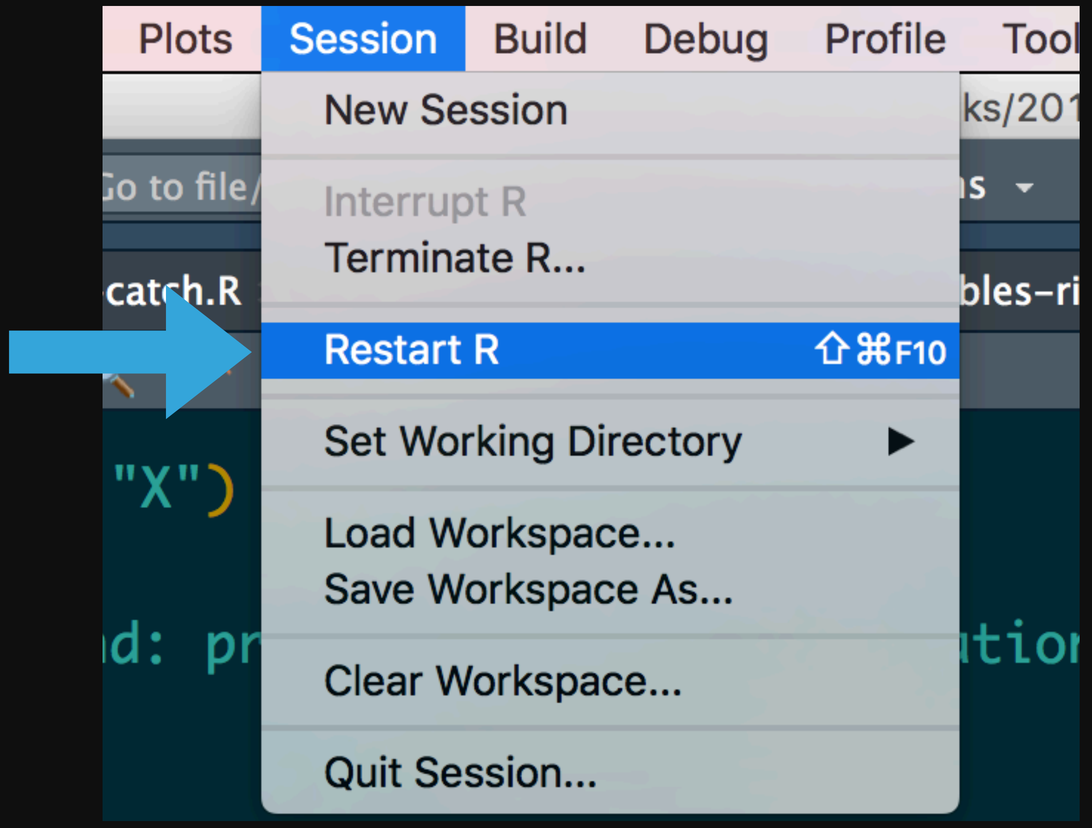
Have you tried turning it OFF and ON again?

“If you love something, set it free.
If it comes back, it is yours.
If it does not, it never was.

RESTART R

Especially when things get weird


Options
General
Code
Appearance
Pane Lay
Packages
R Markdown
Sweave
Spelling
Git/SVN
Publishing
Terminal
Default working directory (when not in a project):
~
Browse...
Re-use idle sessions for project links
Restore most recently opened project at startup
Restore previously open source documents at startup
Restore .RData into workspace at startup
Save workspace to .RData on exit:
Never
Always save history (even when not saving .RData)
Remove duplicate entries in history
Show .Last.value in environment listing
Use debug error handler only when my code contains errors
Automatically expand tracebacks in error inspector
Wrap around when navigating to previous/next tab
Automatically notify me of updates to RStudio


Plots
Session
Build
Debug
Profile
Tool
New Session
Interrupt R
Terminate R...
Restart R
⇧⌘F10
Set Working Directory
Load Workspace...
Save Workspace As...
Clear Workspace...
Quit Session...

```
R --no-save --no-restore-data
```

```
R --no-save --no-restore-data
```

!=

```
rm(list = ls())
```

Poll at slido.com (event code = **HEXAGON**)

WHICH PERSIST AFTER `rm(list = ls())`?

A. `library(dplyr)`

B. `summary <- head`

C. `options(stringsAsFactors = FALSE)`

D. `Sys.setenv(LANGUAGE = "fr")`

E. `x <- 1:5`

F. `attach(iris)`

Poll at slido.com (event code = **HEXAGON**)

WHICH PERSIST AFTER `rm(list = ls())`?

A. `library(dplyr)` ✔

B. `summary <- head` ✘

C. `options(stringsAsFactors = FALSE)` ✔

D. `Sys.setenv(LANGUAGE = "fr")` ✔

E. `x <- 1:5` ✘

F. `attach(iris)` ✔

```
R --no-save --no-restore-data

>>>

rm(list = ls())
```

A FRESH START

Cleans workspace

Resets options and env vars

Clears search path


R sessions are like crops,


not houseplants.


2
REPREX


If at first you don’t succeed...
brood, dither, and fret.

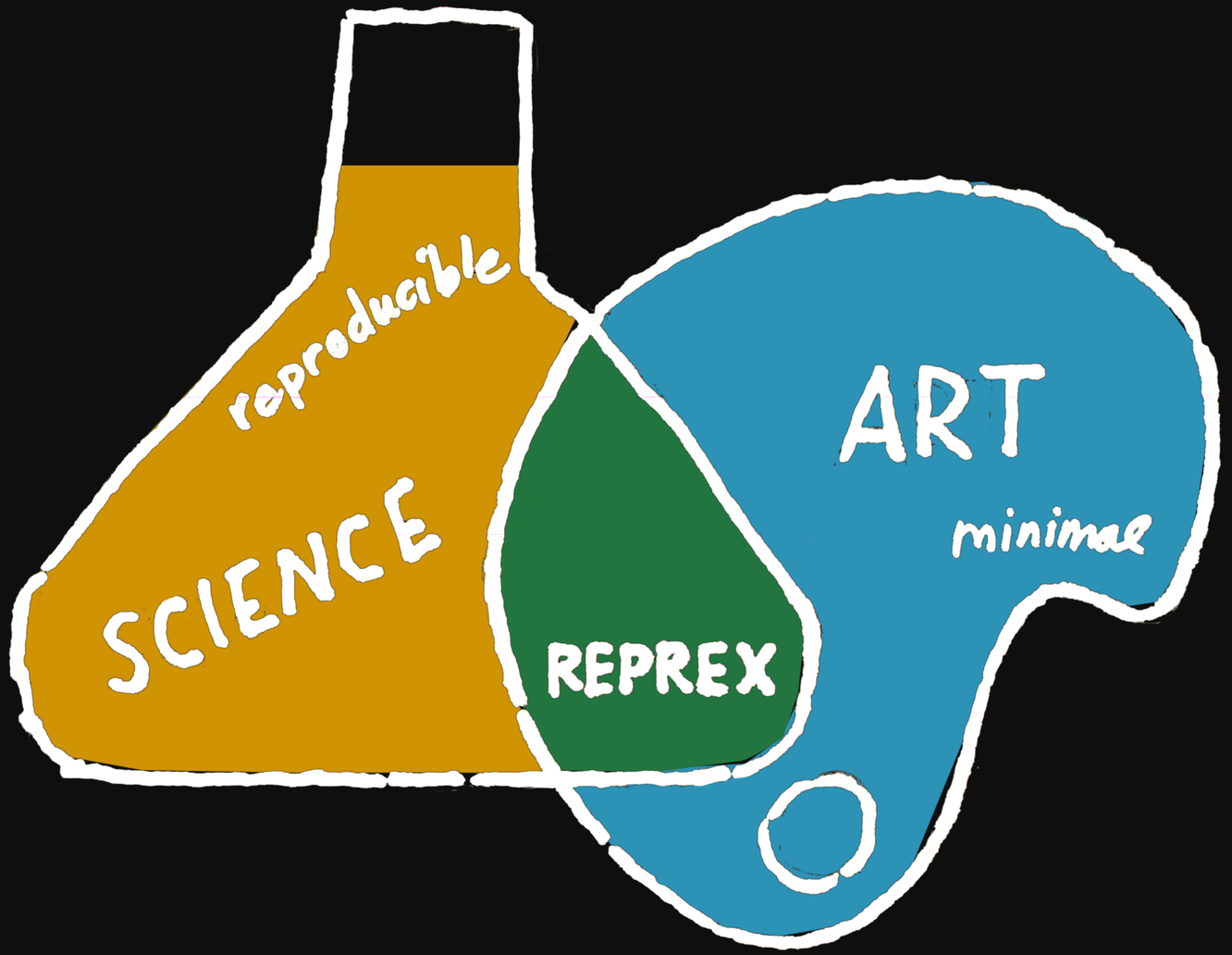
> Don't wring hands and speculate. Work a small concrete example that reveals, confirms, or eliminates something.

reprex

minimum

reproducible

example


reproducible
SCIENCE
REPREX
ART
minimal

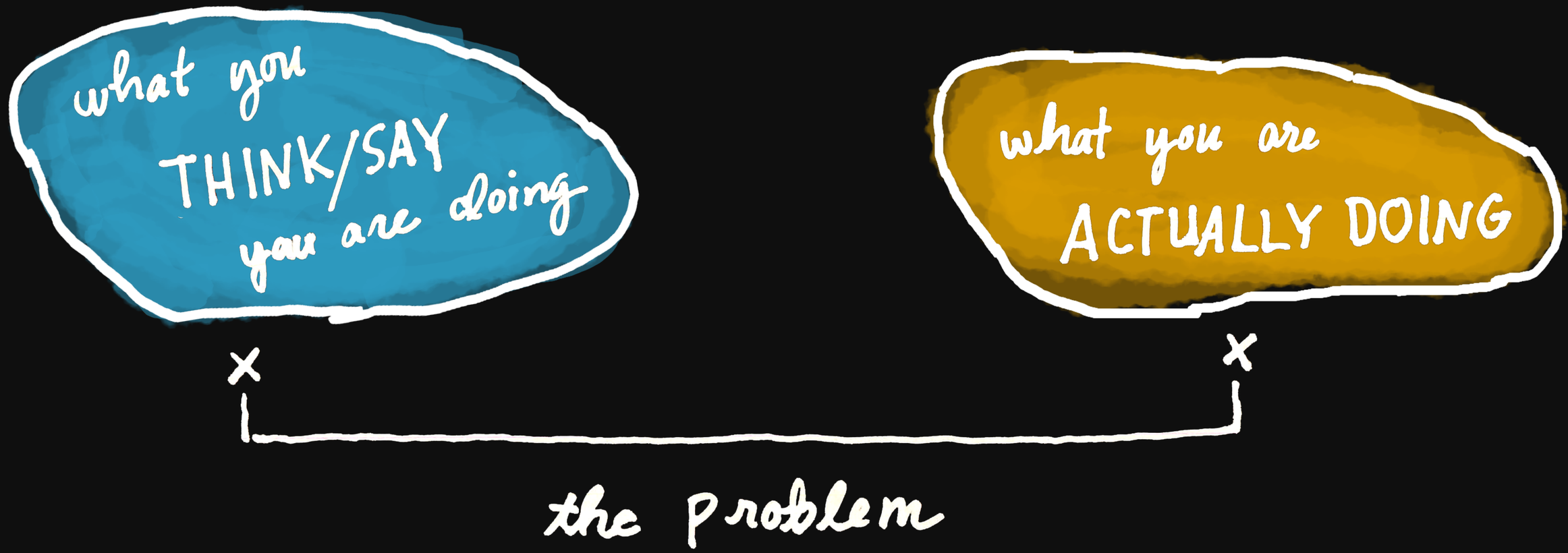
```
template ← "${EXCLAMATION} - your reprex is ${adjective}!"
praise(template)
#> Error in praise(template): could not find function "praise"
```

```
library(praise)
praise(template)
#> Error in grepl(template_pattern, x): object 'template' not found
```

```
library(praise)
template ← "${EXCLAMATION} - your reprex is ${adjective}!"
praise(template)
#> [1] "WOWIE - your reprex is glorious!"
```

REPRODUCIBLE

No reliance on hidden state


what you
THINK/SAY
you are doing
what you are
ACTUALLY DOING
the problem

To FIND a NEEDLE

Try looking in a

SMALLER STACK

```
fs::dir_info(path = raw_data_path,          # list xls[s] files
             regexp = "[.]xls[x]?$") %>%

  dplyr::mutate(sheets = purrr::map(        # create list-col of
    path, ~ readxl::excel_sheets)) %>%      # worksheets

  tidyr::unnest(sheets) %>%                 # get one row per worksheet

  dplyr::mutate(data = purrr::map2(         # read data into a list-col
    path, sheets,                           # of data frames
    ~ readxl::read_excel(.x, .y) %>%        # call `as.character()`
      dplyr::mutate_all(as.character)       # on each column
  ))
#> New names:
#> * `` → `..2`
#> * `` → `..3`
#> * `` → `..4`
#> * `` → `..5`
#> * `` → `..6`
#> Error: the ... list does not contain 3 elements
```

```
fs::dir_info(path = raw_data_path,          # list xls[s] files
             regexp = "[.]xls[x]?$") %>%

  dplyr::mutate(sheets = purrr::map(        # create list-col of
    path, ~ readxl::excel_sheets)) %>%      # worksheets

  tidyr::unnest(sheets) %>%                 # get one row per worksheet

  dplyr::mutate(data = purrr::map2(         # read data into a list-col
    path, sheets,                           # of data frames
    ~ readxl::read_excel(.x, .y) %>%        # call `as.character()`
      dplyr::mutate_all(as.character)       # on each column
  ))
#> New names:
#> * `` → `..2`
#> * `` → `..3`
#> * `` → `..4`
#> * `` → `..5`
#> * `` → `..6`
#> Error: the ... list does not contain 3 elements
```

```
fs::dir_info(path = raw_data_path,
             regexp = "[.]xls[x]?$") %>%

  dplyr::mutate(sheets = purrr::map(
    path, ~ readxl::excel_sheets)) %>%

  tidyr::unnest(sheets) %>%

  dplyr::mutate(data = purrr::map2(
    path, sheets,
    ~ readxl::read_excel(.x, .y) %>%
      dplyr::mutate_all(as.character)
  ))
#> New names:
#> * `` → `..2`
#> * `` → `..3`
#> * `` → `..4`
#> * `` → `..5`
#> * `` → `..6`
#> Error: the ... list does not contain 3 elements
```

private xlsx files

~10 lines of code

5 packages

8 functions

```
dat ← data.frame(`..1` = 1)

dplyr::mutate_all(dat, as.character)
#> Error: Column 1 must not have names
#> of the form ... or ..j.
#> Use .name_repair to specify repair.
```

inlined data

2 lines of code

1 package

1 function

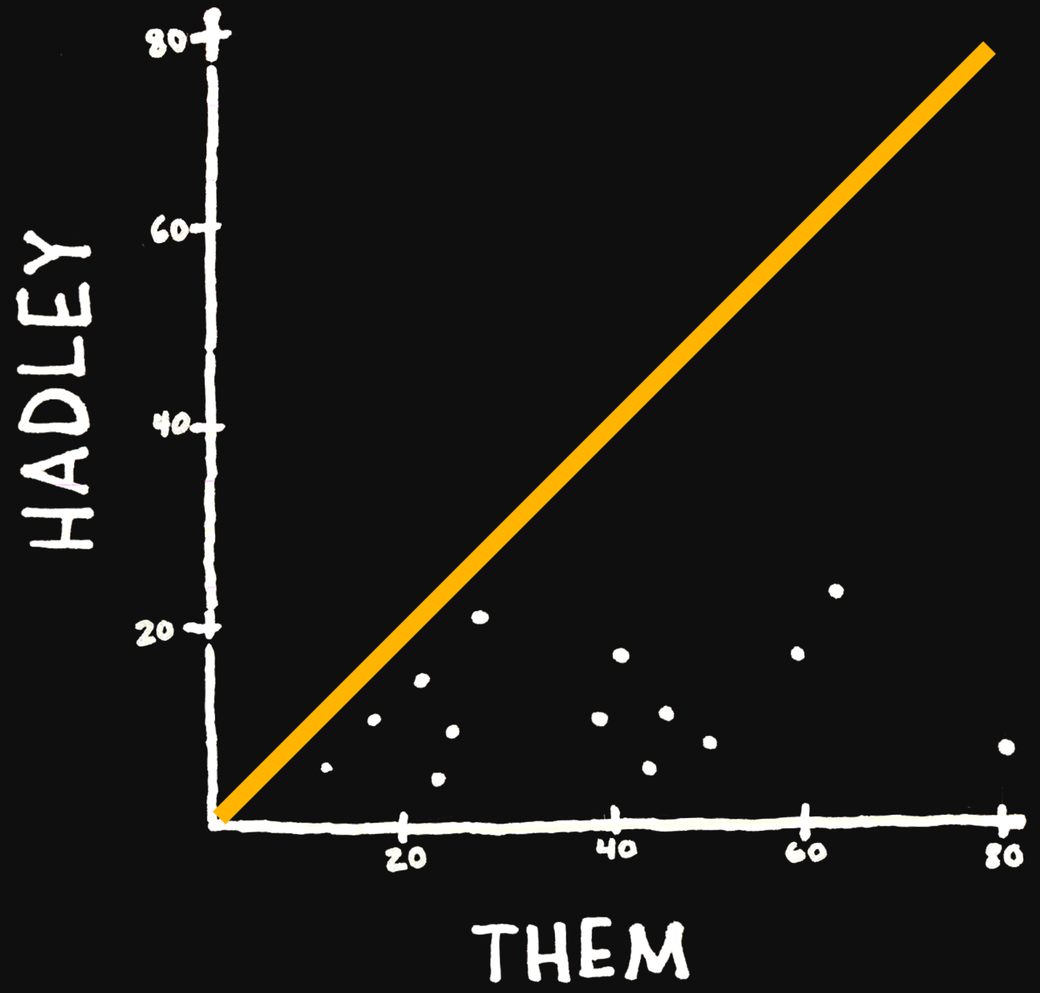
MINIMAL

Inputs are small and simple

~~Extraneous packages~~

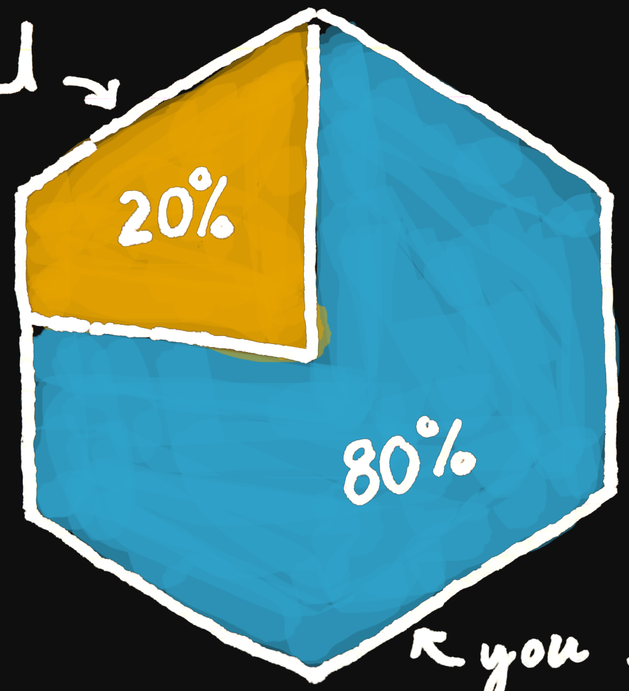
~~Unnecessary function calls~~

SIZE of REPREX



WHY make a REPREX?

helps them
HELP YOU



you SOLVE
your own
PROBLEM


3
DEBUG

Poll at slido.com (event code = **HEXAGON**)

HAVE YOU GOTTEN STUCK IN THE DEBUGGER?

A. R has a debugger? What is a debugger?

B. No.

C. Yes.

D. Yes and I'm not even sure how I got there.

```
# Error : .onLoad failed in loadNamespace() for 'rJava', details:
# call: dyn.load(file, DLLpath = DLLpath, ...)
# error: unable to load shared object '/Users/janedoe/Library/R/3.6/library/rJava/libs/rJava.so':
# libjvm.so: cannot open shared object file: No such file or directory
# Error: loading failed
# Execution halted
# ERROR: loading failed
# * removing '/Users/janedoe/Library/R/3.6/library/rJava/'
# Warning in install.packages :
# installation of package 'rJava' had non-zero exit status
```

```
# Error : blah bl failed blah blah blah blah blah blah blah blah:
# blah: bla.blah(blah, blahbla = blahbla, ...)
# error: unable to blah blah blah bla '/blahb/blahbla/blahbla/b/b.b/blahbla/blahb/blah/blahb.so':
# blahbl.so: cannot blah blah blah blah bla: No blah blah blah blah bl
# Error: blah bl failed
# Blah blah blah b
# ERROR: blah bl failed
# * removing '/blahb/blahbla/blahbla/b/b.b/blahbla/blahb/'
# Warning in blah blah blah b :
# blahblahblah bl blahbla 'blahb' bla bla-blah blah blahbl
```

```
dat
#>            blackberry blueberry peach plum
#> calories            4         1    59   30
#> weight              9         2   150   78
#> yumminess           6         8    10    5

fruit_avg(dat, pattern = "berry")
#> Found 2 fruits!
#>  calories    weight yumminess
#>       2.5       5.5       7.0
```

```
dat
#>            blackberry blueberry peach plum
#> calories            4         1    59   30
#> weight              9         2   150   78
#> yumminess           6         8    10    5

fruit_avg(dat, pattern = "melon")
#> Found 0 fruits!
#>  calories    weight yumminess
#>       NaN       NaN       NaN
```

```
dat
#>            blackberry blueberry peach plum
#> calories            4         1    59   30
#> weight              9         2   150   78
#> yumminess           6         8    10    5

fruit_avg(dat, pattern = "black")
#> Found  fruits!
#> Error in rowMeans(mini_dat): 'x' must be an array
#> of at least two dimensions
```

```
dat
#>            blackberry blueberry peach plum
#> calories            4         1    59   30
#> weight              9         2   150   78
#> yumminess           6         8    10    5

fruit_avg(dat, pattern = "black")
#> Found  fruits!
#> Error in rowMeans(mini_dat): 'x' must be an array
#> of at least two dimensions
```

Certificate of Death

: 16

WILLIE SC

First Name | Middle Name | Last

RTICULARS
edical Examiner.)

N.Y.

Town Springfield
City

Ave.
St.

sion)
City
to death Life

Married

MEDICAL CERTI
(To be filled in by Me

16. PLACE OF DEATH:

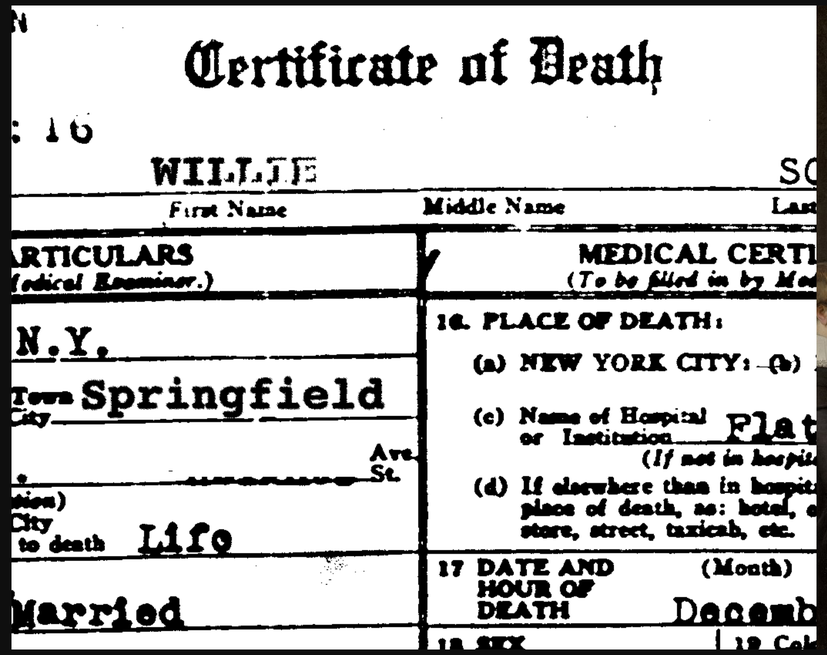
(a) NEW YORK CITY: (b)

(c) Name of Hospital or Institution Flat
(If not in hospit

(d) If elsewhere than in hospit place of death, as: hotel, store, street, taxicab, etc.

17 DATE AND HOUR OF DEATH (Month) Decemb

18 SEX | 19 Col



traceback()

browser()

options(error = recover)

```
fruit_avg(dat, pattern = "black")
#> Found  fruits!
#> Error in rowMeans(mini_dat): 'x' must be an array
#> of at least two dimensions

traceback()
#> 3: stop("'x' must be an array of at least two dimensions")
#> 2: rowMeans(mini_dat) at fruit_avg.R#5
#> 1: fruit_avg(dat, pattern = "black")
```

call
stack
trace
back

call stack
trace back
stack trace
back trace
call back

call

stack

trace

back

call stack

trace back

stack trace

back trace

~~call back~~

```
fruit_avg(dat, pattern = "black")
#> Found  fruits!
#> Error in rowMeans(mini_dat): 'x' must be an array
#> of at least two dimensions

rlang::last_trace()
#> <error/rlang_error>
#> 'x' must be an array of at least two dimensions
#> Backtrace:
#>     ▆
#>  1. └─global::fruit_avg(dat, pattern = "black")
#>  2.   └─base::rowMeans(mini_dat) R/fruit_avg.R:5:2
```

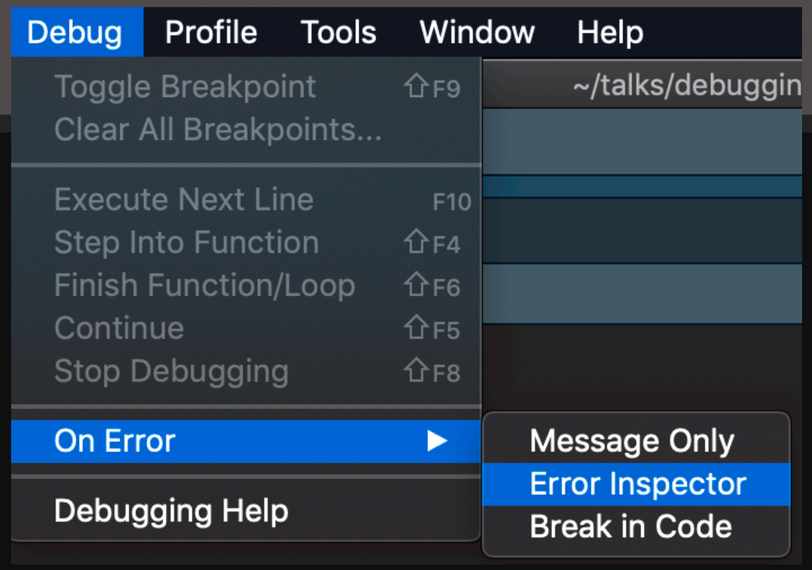
```
> fruit_avg(dat, pattern = "black")
Found  fruits!
```

```
Error in rowMeans(mini_dat) :
  'x' must be an array of at least two dimensions
```

Hide Traceback

Rerun with Debug

```
 3. stop("'x' must be an array of at least two dimensions")
 2. rowMeans(mini_dat) at fruit_avg.R#5
 1. fruit_avg(dat, pattern = "black")
```



Video of a tiny room hidden behind an electrical outlet.

By Mozu Studios

https://www.mozustudios.com

https://www.instagram.com/p/B6NvxK7JnpU

```
options(error = recover)

fruit_avg(dat, "black")
#> Found  fruits!
#> Error in rowMeans(mini_dat): 'x' must be an array
#> of at least two dimensions

Enter a frame number, or 0 to exit
1: fruit_avg(dat, "black")
2: fruit_avg.R#5: rowMeans(mini_dat)

Selection: 1
```

```
Enter a frame number, or 0 to exit
1: fruit_avg(dat, "black")
2: fruit_avg.R#5: rowMeans(mini_dat)

Selection: 1

Browse[1]> ls.str()
cols :  int 1
dat : 'data.frame':3 obs. of  4 variables:
 $ blackberry: int  4 9 6
 $ blueberry : int  1 2 8
 $ peach     : int  59 150 10
 $ plum      : int  30 78 5
mini_dat :  int [1:3] 4 9 6
pattern :  chr "black"
```

```
Enter a frame number, or 0 to exit
1: fruit_avg(dat, "black")
2: fruit_avg.R#5: rowMeans(mini_dat)

Selection: 1

Browse[1]> ls.str()
cols :  int 1
dat : 'data.frame':3 obs. of  4 variables:
 $ blackberry: int  4 9 6
 $ blueberry : int  1 2 8
 $ peach     : int  59 150 10
 $ plum      : int  30 78 5
mini_dat :  int [1:3] 4 9 6
pattern :  chr "black"
```

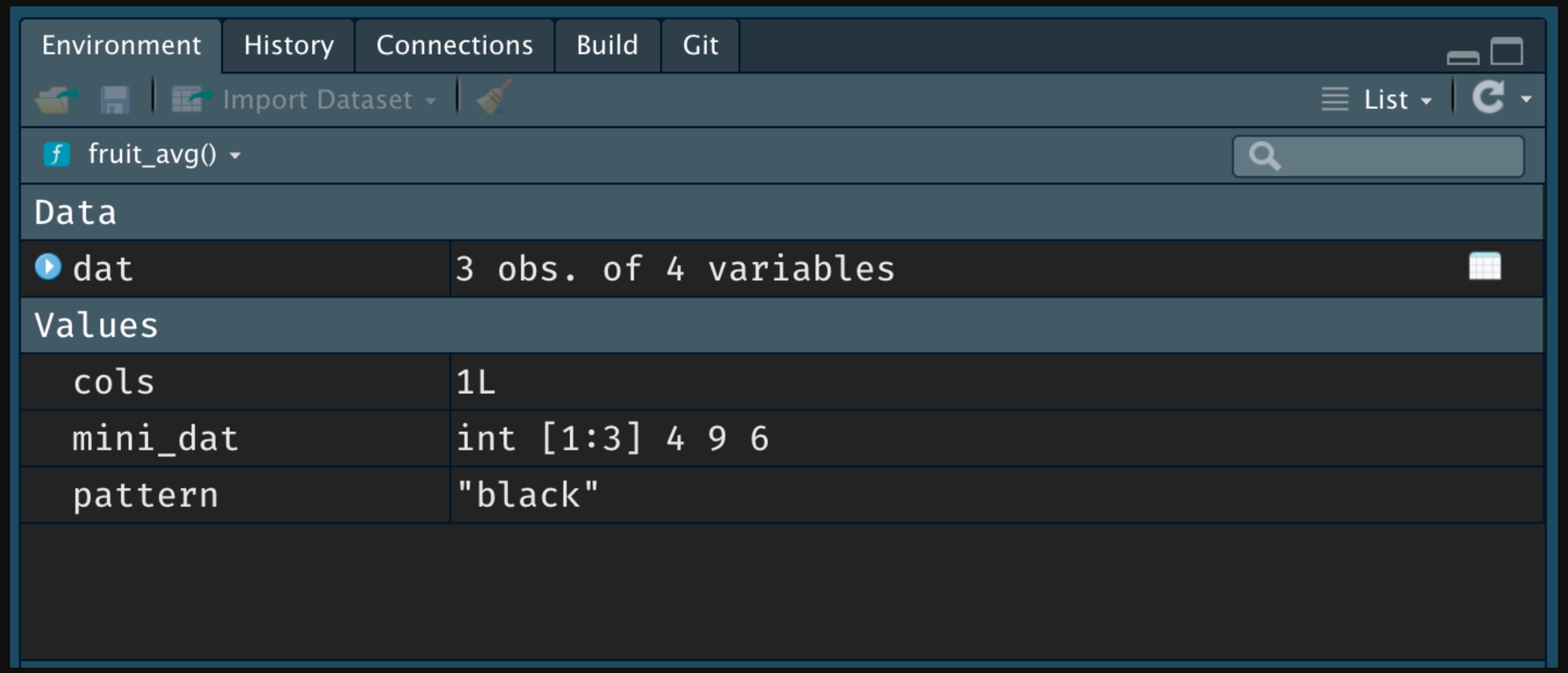
```
Enter a frame number, or 0 to exit
1: fruit_avg(dat, "black")
2: fruit_avg.R#5: rowMeans(mini_dat)

Selection: 1

Browse[1]> ls.str()
cols :  int 1
dat : 'data.frame':3 obs. of  4 variables:
 $ blackberry: int  4 9 6
 $ blueberry : int  1 2 8
 $ peach     : int  59 150 10
 $ plum      : int  30 78 5
mini_dat :  int [1:3] 4 9 6
pattern :  chr "black"
```

```
Enter a frame number, or 0 to exit
1: fruit_avg(dat, "black")
2: fruit_avg.R#5: rowMeans(mini_dat)

Selection: 1
```



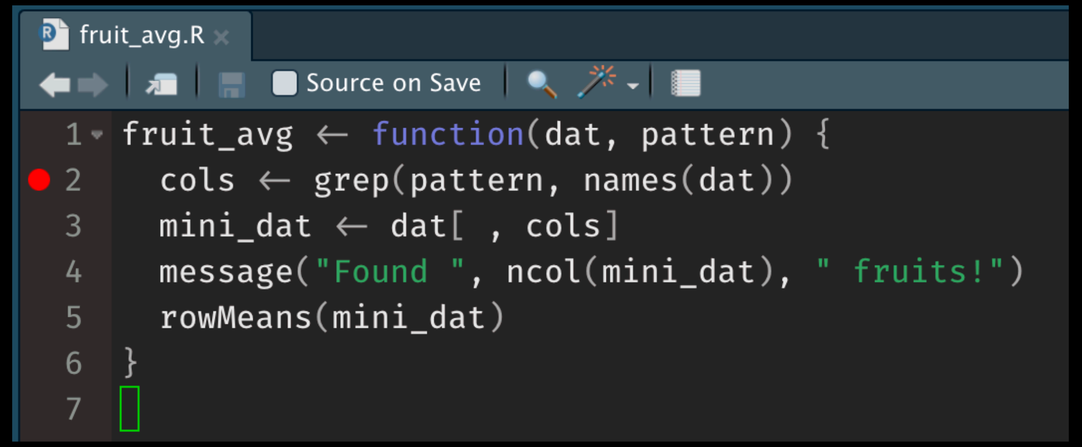
```
fruit_avg ← function(dat, pattern) {
  cols ← grep(pattern, names(dat))
  mini_dat ← dat[ , cols]
  message("Found ", ncol(mini_dat), " fruits!")
  rowMeans(mini_dat)
}
```

```r
fruit_avg ← function(dat, pattern) {
  browser()
  cols ← grep(pattern, names(dat))
  mini_dat ← dat[ , cols]
  message("Found ", ncol(mini_dat), " fruits!")
  rowMeans(mini_dat)
}
```

```
fruit_avg ← function(dat, pattern) {
  browser()
  cols ← grep(pattern, names(dat))
  mini_dat ← dat[ , cols]
  message("Found ", ncol(mini_dat), " fruits!")
  rowMeans(mini_dat)
}
```

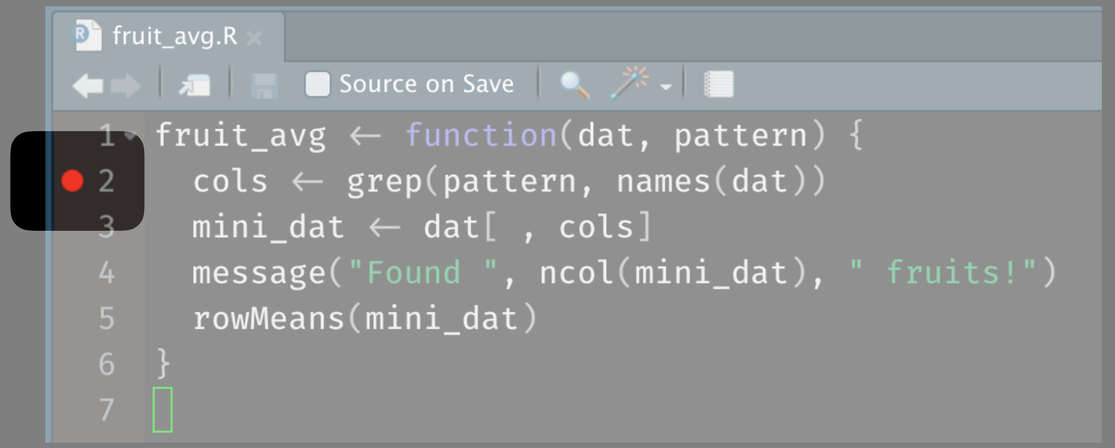
```r
fruit_avg ← function(dat, pattern) {
  browser()
  cols ← grep(pattern, names(dat))
  mini_dat ← dat[ , cols]
  message("Found ", ncol(mini_dat), " fruits!")
  rowMeans(mini_dat)
}
```

```r
debug(fruit_avg)
```

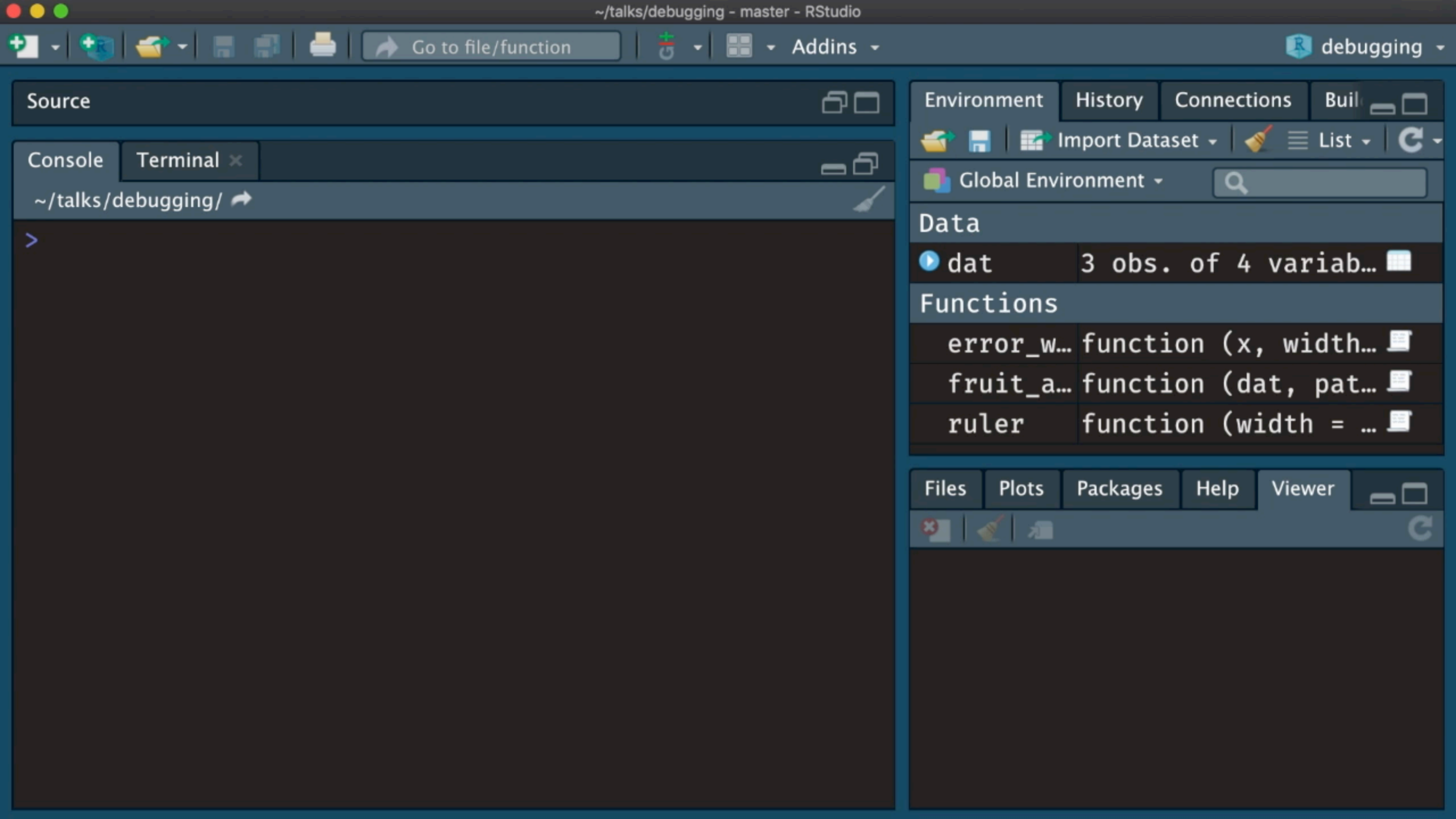


```r
fruit_avg <- function(dat, pattern) {
  browser()
  cols <- grep(pattern, names(dat))
  mini_dat <- dat[ , cols]
  message("Found ", ncol(mini_dat), " fruits!")
  rowMeans(mini_dat)
}
```

```r
debug(fruit_avg)
```



next slide is a GIF of debugging with browser() inside RStudio


~/talks/debugging - master - RStudio
Go to file/function
Addins
debugging
Source
Console
Terminal
~/talks/debugging/
>
Environment
History
Connections
Import Dataset
List
Global Environment
Data
dat
3 obs. of 4 variab…
Functions
error_w…
function (x, width…
fruit_a…
function (dat, pat…
ruler
function (width = …
Files
Plots
Packages
Help
Viewer

```
fruit_avg ← function(dat, pattern) {
  cols ← grep(pattern, names(dat))
  mini_dat ← dat[ , cols, drop = FALSE]
  message("Found ", ncol(mini_dat), " fruits!")
  rowMeans(mini_dat)
}
```

Poll at slido.com (event code = **HEXAGON**)

HELP!

- Q
- RStudio ■ stop button
- `debug()` + `undebug()`
- `debugonce()`

4 DETER

FIX IT ONCE? KEEP IT FIXED

ADD A TEST

```
# https://github.com/OWNER/REPO/issues/666
test_that("fruit_avg() works for 0, 1, ⩾2 matches", {
  dat ← data.frame(a = 1:2, ab = 3:4, row.names = c("one", "two"))
  expect_equal(fruit_avg(dat, "abc"), c(one = NaN, two = NaN))
  expect_equal(fruit_avg(dat,  "ab"), c(one = 3,   two = 4))
  expect_equal(fruit_avg(dat,   "a"), c(one = 2,   two = 3))
})
```

ADD AN ASSERTION

```
dat ← read.csv("fruit.csv")

if (!all(vapply(dat, is.numeric, logical(1)))) {
  stop("All columns of `dat` must be numeric")
}

fruit_avg(dat, pattern = "berry")
```

AUTOMATE YOUR CHECKS

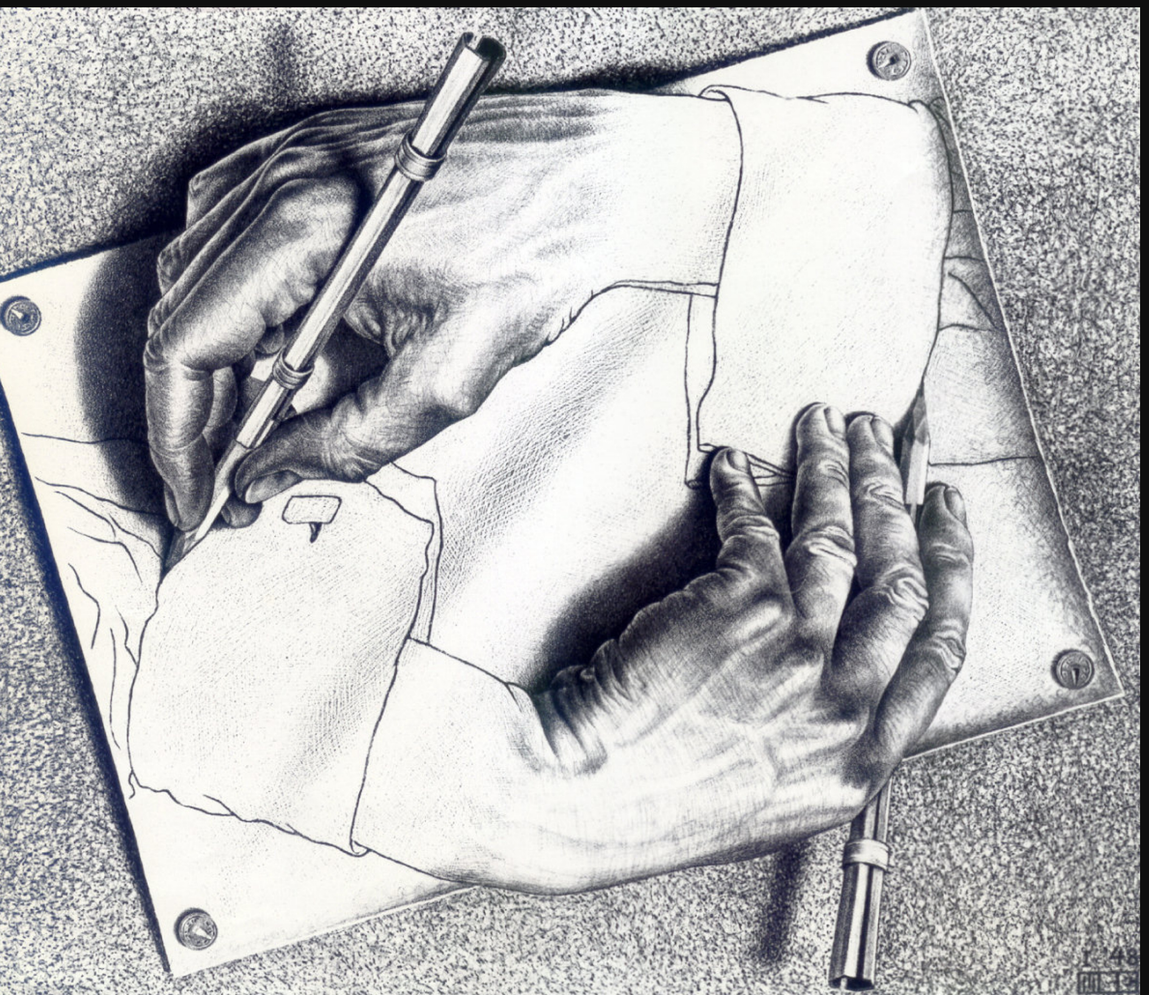
R CMD check

testthat::test_check()

RUN YOUR CHECKS ON THEIR MACHINE

Continuous integration

USE MIND BENDY STUFF IN MODERATION



> The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at and repair.

Douglas Adams

LEAVE ACCESS PANELS

```
readxl + xls2csv, excelgesis

httr::with_verbose()

rlang::qq_show()

options(future.debug = TRUE)

options(gargle_quiet = FALSE)

curl::handle_setopt(h, verbose = TRUE)

options(internet.info = 0)
```

WRITE ERROR MESSAGES FOR HUMANS

```
dat ← data.frame(x = 1, y = 2)

df$x
#> Error: object of type 'closure' is not subsettable
```

WRITE ERROR MESSAGES FOR HUMANS

```
dat ← data.frame(x = 1, y = 2)

df$x
#> Error: object of type 'function' is not subsettable
```

WRITE ERROR MESSAGES FOR HUMANS

```
dat ← data.frame(x = 1, y = 2)

df$x
#> Error: Can't subset a function.
#> Have you forgotten to define a variable named `df`?
```

WRITE ERROR MESSAGES FOR HUMANS

```
library(dplyr)

filter(iris, Species = "setosa")
#> Error: `Species` (`Species = "setosa"`) must not be
#> named, do you need `==`?
```

Turn it off and on again

Make a reprex

Dig into the error

Plan for the unexpected

Thanks:

Tidyverse team

Christine Kuper

rstd.io/debugging

Jennifer Bryan

RStudio

@jennybc

@JennyBryan

IMAGE SOURCES

- Fret: https://unsplash.com/photos/OsC8HauR0e0
- Do same thing again: https://unsplash.com/photos/uxUUENpp01I
- Diver: https://unsplash.com/photos/wVvxjiLJr-g
- Ocean horizon background: https://unsplash.com/photos/sYzFIusQp3Q
- Calm sea background: https://unsplash.com/photos/IZ01rjX0XQA
- Coral reef background: https://unsplash.com/photos/T1Wru10gKhg
- Seaweed background: https://unsplash.com/photos/nAkC-KS444M
- Orchid: https://unsplash.com/photos/Ug6z9PCwr58
- Corn field: https://unsplash.com/photos/nCQXxsSg3oo
- On/off key: https://unsplash.com/photos/cw_uvISXkCI

IMAGE SOURCES CONTINUED

- Sunlight under water background: https://unsplash.com/photos/K785Da4A_JA
- Garnishing with sauce: https://unsplash.com/photos/YaiY50wzWzI
- death certificate (modified): Public Domain, https://commons.wikimedia.org/w/index.php?curid=214170
- washing pot: https://unsplash.com/photos/-VhH4S1Lur8
- The Night King: https://cnet4.cbsistatic.com/img/vugy5MvUVBvwcJf0JvKIBd1RwJE=/1200x675/2019/04/22/2b2fee8d-111a-4d19-ae83-4e61899cfd47/1nightking.jpg (probably copyright HBO)
- Autopsy painting by Rembrandt: https://www.mauritshuis.nl/en/explore/the-collection/artworks/the-anatomy-lesson-of-dr-nicolaes-tulp-146/detailgegevens/ Public Domain, https://commons.wikimedia.org/w/index.php?curid=64281722
- Gray cube abstract wallpaper vector art https://unsplash.com/photos/1CVy8JStf3A