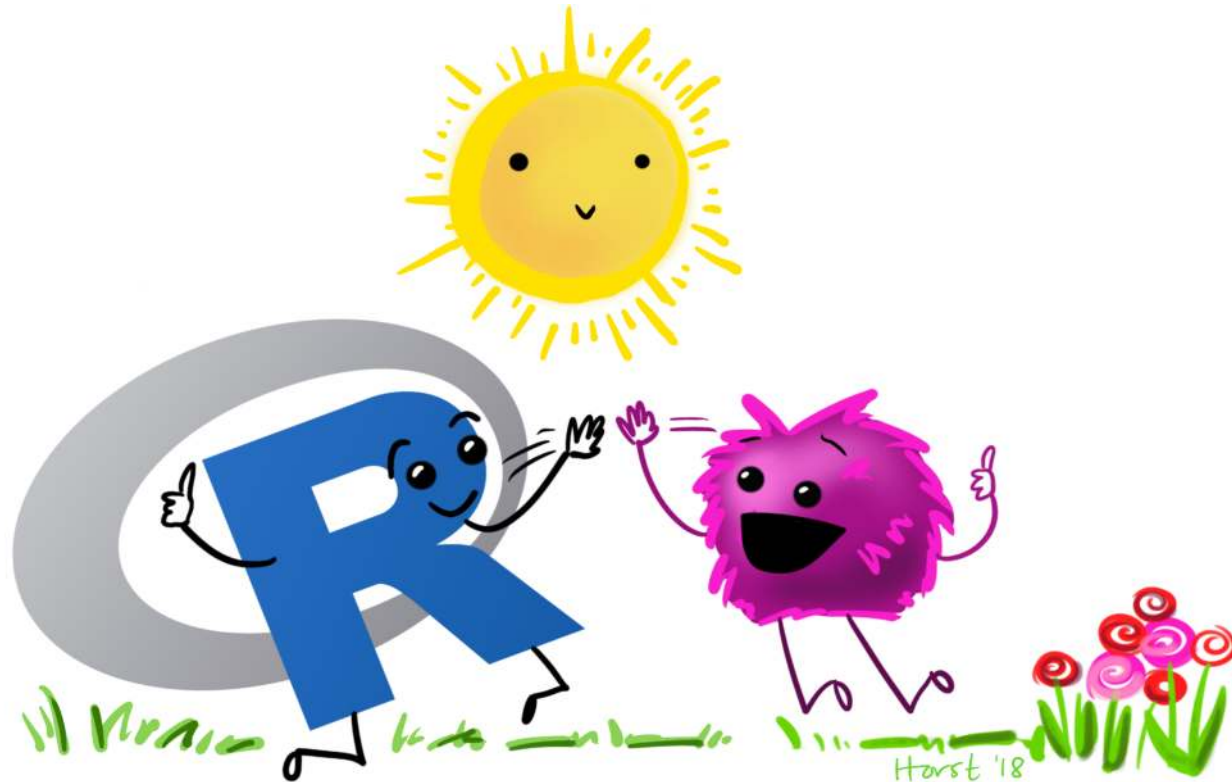


Getting started with R



Check-in

- Everyone getting emails?
 - Email about these slides?
- Everyone have access to these slides?
[https://steffilazerte.ca/NRI 7350/slides.html](https://steffilazerte.ca/NRI_7350/slides.html)

About these Labs

Format

- I will provide you tools and workflow to get started with R
- I will go over specific statistical functions
 - How to run them
 - How to interpret the results
- We'll have hands-on, lecture, and demonstrations

R is hard: But have no fear!

- Don't expect to remember everything!
- Copy/Paste is your friend (never apologize for using it!)
- Consider these labs a resource to return to

About these Labs

Format

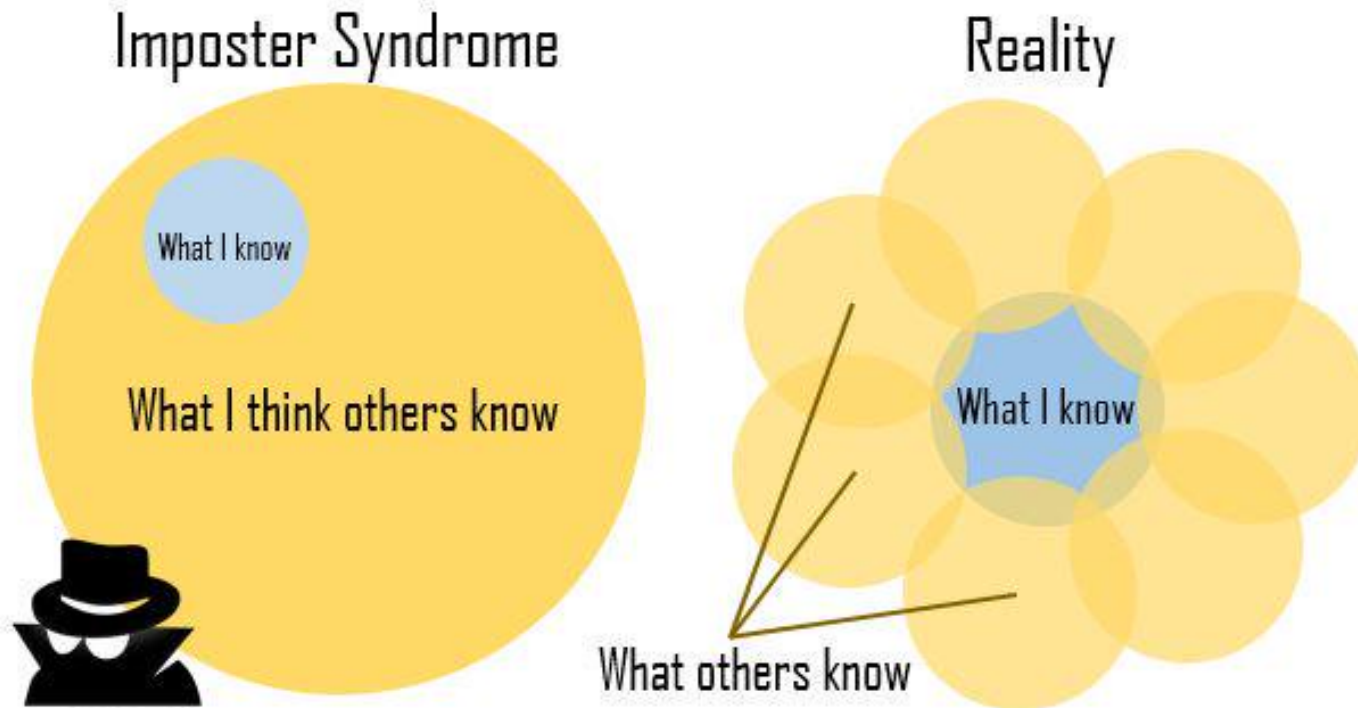
- I will provide you tools and workflow to get started with R
- I will go over specific statistical functions
 - How to run them
 - How to interpret the results
- We'll have hands-on, lecture, and demonstrations

R is hard: But have no fear!

- **Don't expect to remember everything!**
- Copy/Paste is your friend (never apologize for using it!)
- Consider these labs a resource to return to

Impost Syndrome

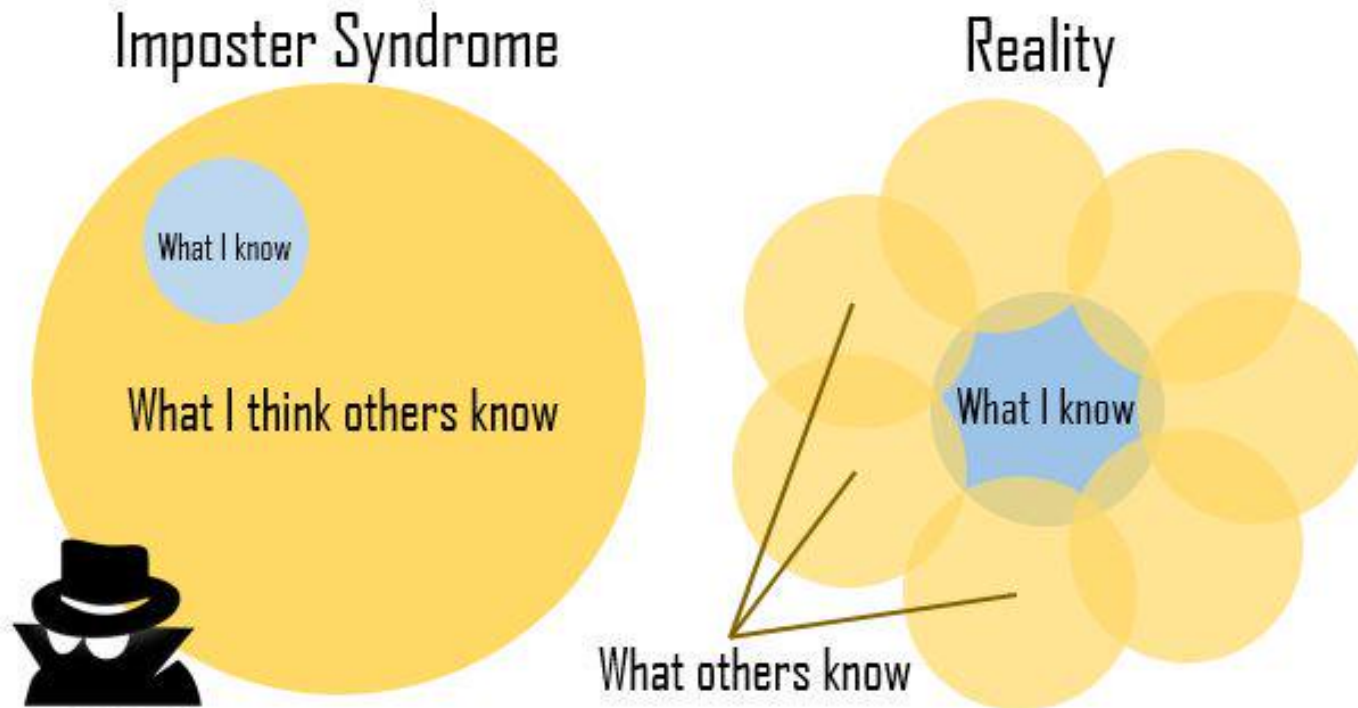
ImpostR Syndrome



David Whittaker

ImpostR
Syndrome

ImpostR Syndrome



David Whittaker

ImpostR Syndrome

Moral of the story?

Make friends, code in groups, learn together and don't beat yourself up

at first I was like...



...but now it's like...



About R

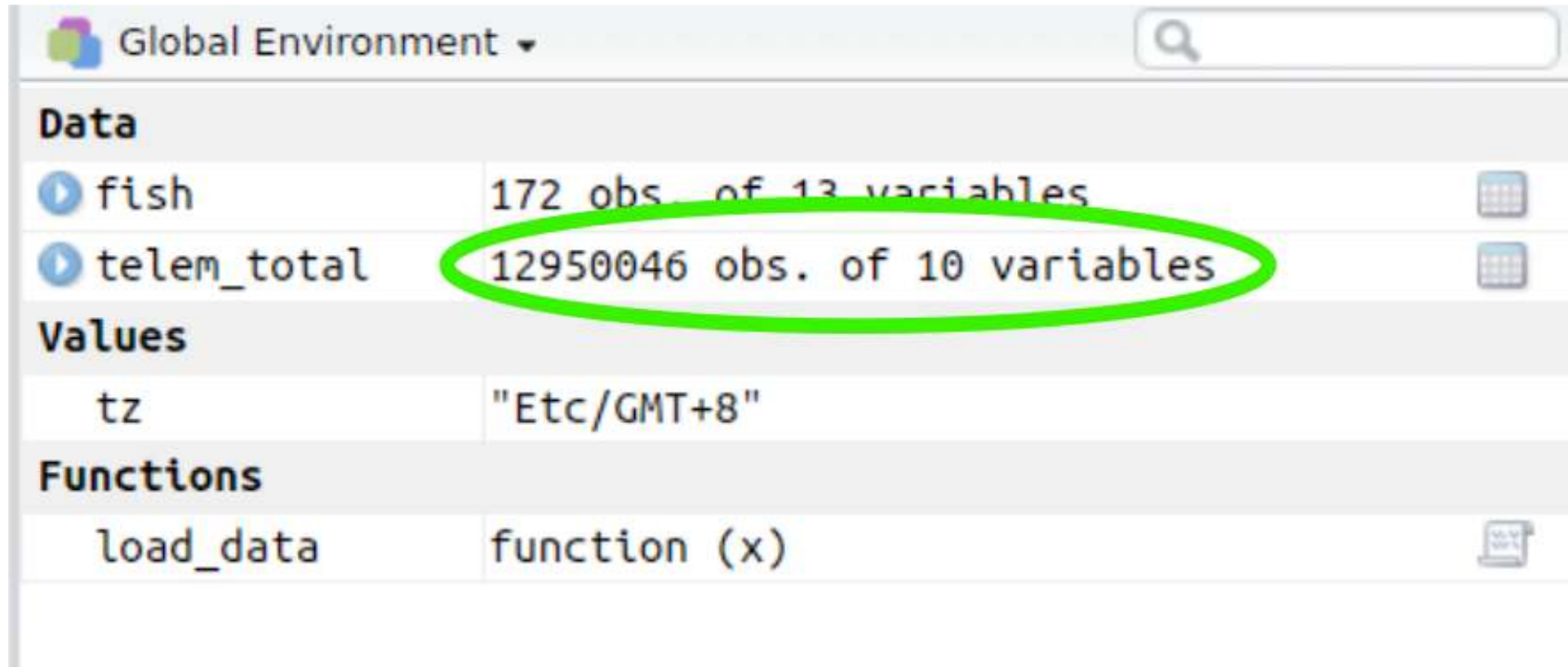
Why R?

R is hard

```
# Get in circle around city
circle <- data.frame()
cutoff <- 10
for(i in unique(gps$region)) {
  n <- nrow(gps[gps$region == i,]) ##number of IDs
  if(i == "wil") tmp <- geocode("Williams Lake, Canada")
  if(i == "kam") tmp <- geocode("Kamloops, Canada")
  if(i == "kel") tmp <- geocode("Kelowna, Canada")
  temp <- data.frame()
  for(a in 1:n){
    if(a <= cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                         lat = tmp$lat,
                                                         bearing = (a*(360/(cutoff))-360/(cutoff)),
                                                         dist = 20,
                                                         dist.units = "km",
                                                         model = "WGS84"))
    if(a > cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                         lat = tmp$lat,
                                                         bearing = ((a-cutoff)*(360/(max(table(gps$region))-10))-360/(max(table(gps$region))-cutoff)),
                                                         dist = 35,
                                                         dist.units = "km",
                                                         model = "WGS84"))
  }
  circle <- rbind(circle, cbind(temp,
                                region = i,
                                hab = gps$hab[gps$region == i],
                                spl = gps$spl.orig[gps$region == i],
```

Why R?

But R is powerful (and reproducible)!

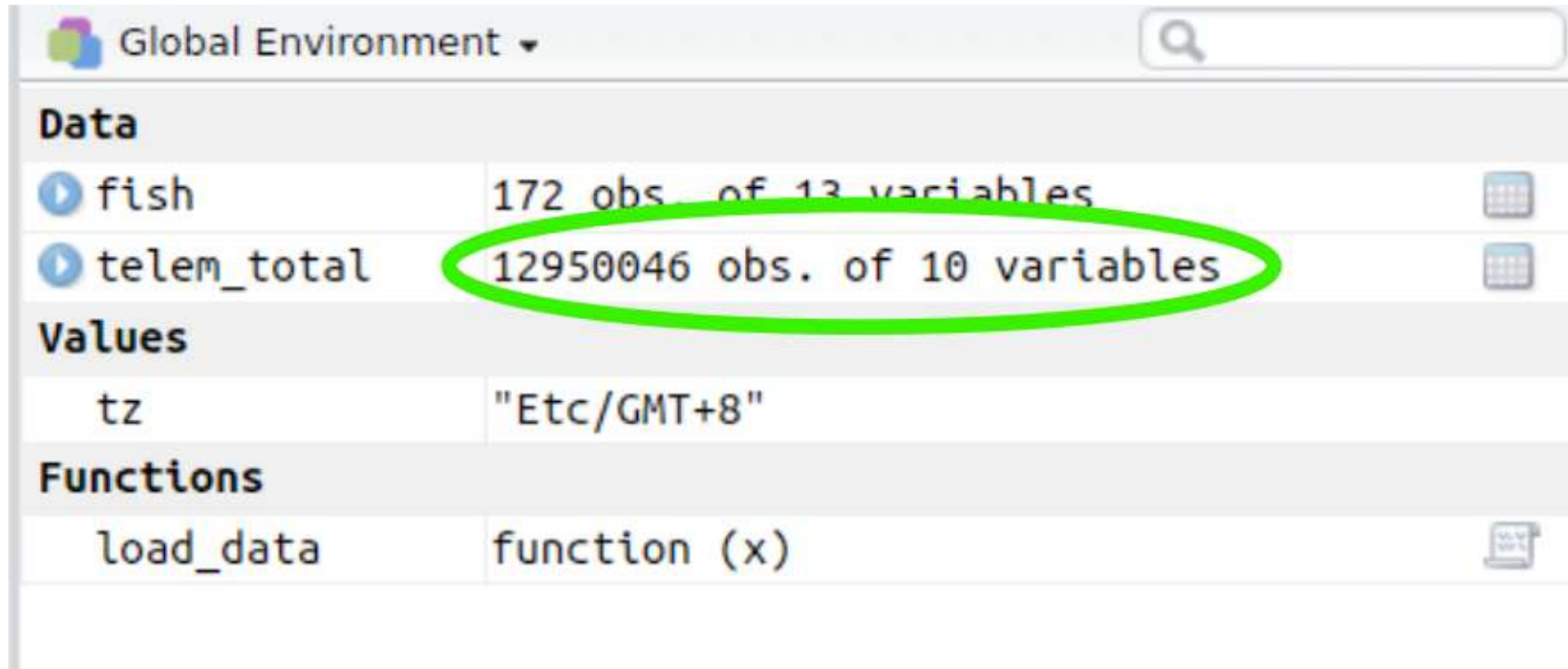


The screenshot shows the 'Global Environment' pane in an R IDE. It is divided into four sections: 'Data', 'Values', 'Functions', and 'load_data'. The 'Data' section lists two objects: 'fish' (172 obs. of 13 variables) and 'telem_total' (12950046 obs. of 10 variables). The 'Values' section shows 'tz' with the value 'Etc/GMT+8'. The 'Functions' section shows 'load_data' as a function (x). The 'telem_total' entry is circled in green.

| Global Environment | |
|--------------------|-------------------------------|
| Data | |
| fish | 172 obs. of 13 variables |
| telem_total | 12950046 obs. of 10 variables |
| Values | |
| tz | "Etc/GMT+8" |
| Functions | |
| load_data | function (x) |

Why R?

But R is powerful (and reproducible)!

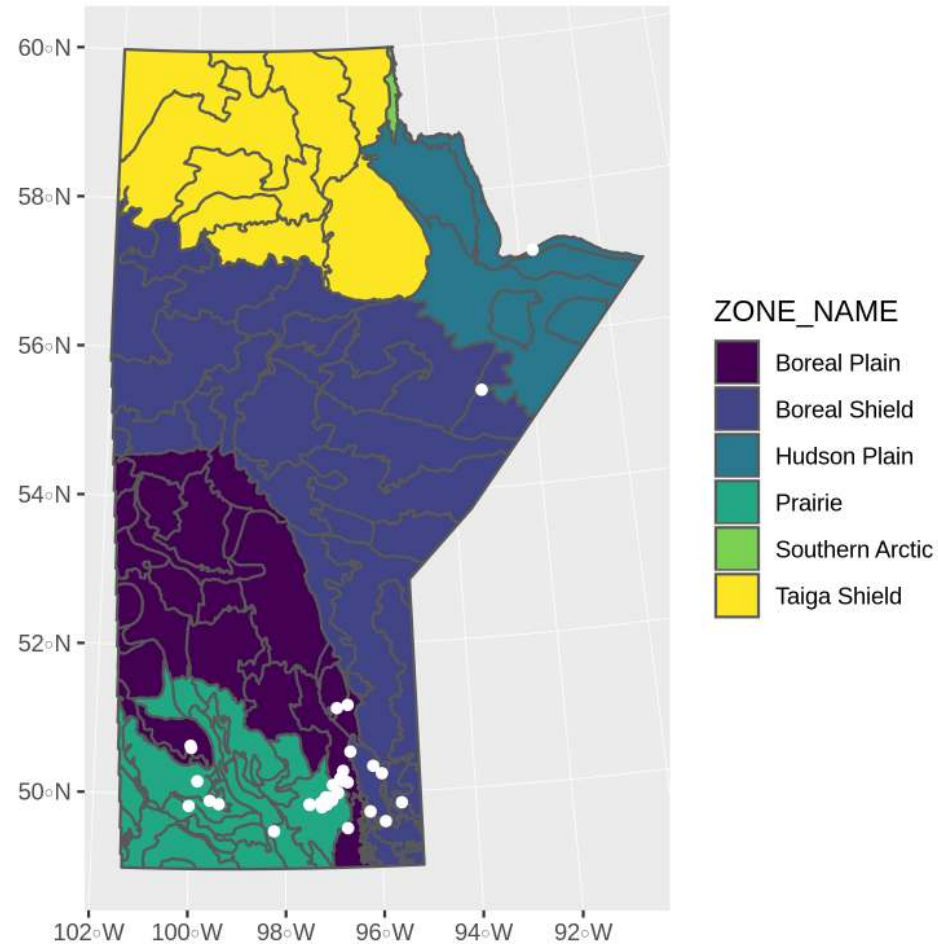


The screenshot shows the R Global Environment pane. It is divided into sections: Data, Values, and Functions. In the Data section, two objects are listed: 'fish' with 172 observations and 13 variables, and 'telem_total' with 12950046 observations and 10 variables. The 'telem_total' entry is circled in green. In the Values section, the 'tz' variable is shown with the value 'Etc/GMT+8'. In the Functions section, the 'load_data' function is shown as a function of 'x'.

| Global Environment | |
|--------------------|-------------------------------|
| Data | |
| fish | 172 obs. of 13 variables |
| telem_total | 12950046 obs. of 10 variables |
| Values | |
| tz | "Etc/GMT+8" |
| Functions | |
| load_data | function (x) |

Why R?

R is also beautiful



Why R?

R is affordable (i.e., free!)

R is available as Free Software under the terms of the [Free Software Foundation's GNU General Public License](#) in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

What is R?

R is Programming language

A programming **language** is a way to give instructions in order to get a computer to do something

- You need to know the language (i.e., the code)
- Computers don't know what you mean, only what you type (unfortunately)
- Spelling, punctuation, and capitalization all matter!

For example

R, what is 56 times 5.8?

```
56 * 5.8
```

```
## [1] 324.8
```


Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

R, multiply this value by 6

```
steffis_mean * 6
```

```
## [1] 15
```

Code, Output, Scripts

Code

- The actual commands

Output

- The result of running code or a script

Script

- A text file full of code that you want to run
- You should always keep your code in a script

Code, Output, Scripts

Code

- The actual commands

Output

- The result of running code or a script

Script

- A text file full of code that you want to run
- You should always keep your code in a script

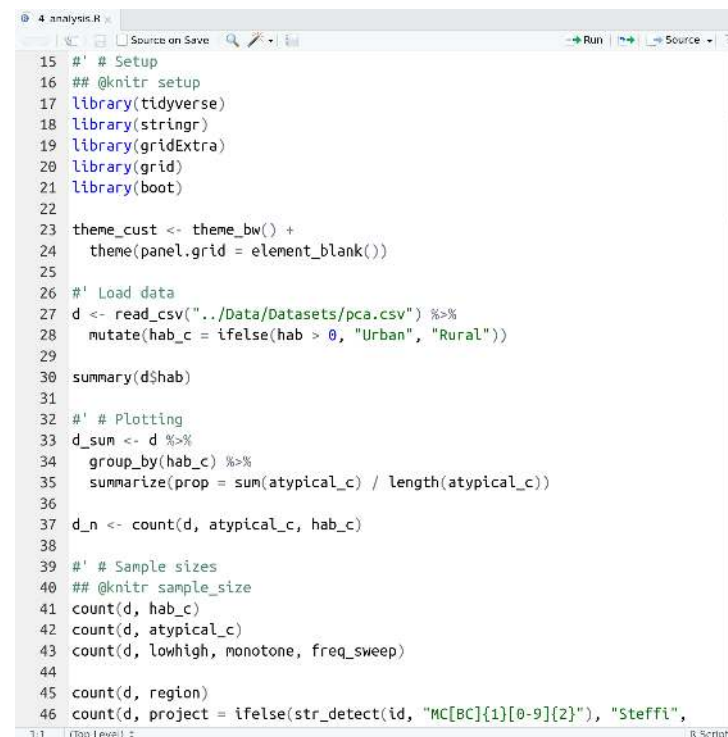
For example:

```
mean(c(1, 2, 3, 4))
```

Code

```
## [1] 2.5
```

Output



```
15 #' # Setup
16 ## @knitr setup
17 library(tidyverse)
18 library(stringr)
19 library(gridExtra)
20 library(grid)
21 library(boot)
22
23 theme_cust <- theme_bw() +
24   theme(panel.grid = element_blank())
25
26 #' Load data
27 d <- read_csv("../Data/Datasets/pca.csv") %>%
28   mutate(hab_c = ifelse(hab > 0, "Urban", "Rural"))
29
30 summary(d$hab)
31
32 #' # Plotting
33 d_sum <- d %>%
34   group_by(hab_c) %>%
35   summarize(prop = sum(atypical_c) / length(atypical_c))
36
37 d_n <- count(d, atypical_c, hab_c)
38
39 #' # Sample sizes
40 ## @knitr sample_size
41 count(d, hab_c)
42 count(d, atypical_c)
43 count(d, lowhigh, monotone, freq_sweep)
44
45 count(d, region)
46 count(d, project = ifelse(str_detect(id, "MC[BC]{1}[0-9]{2}"), "Steffi",
```

Script

RStudio vs. R



RStudio



R

- **RStudio** is not **R**
- RStudio is a User Interface or IDE (integrated development environment)
 - (i.e., Makes coding simpler)
- But sometimes tries to be **too** helpful

RStudio Features

Changing Options: Tools > Global Options

- General > Restore RData into workspace at startup (NO!)
- General > Save workspace to on exit (NEVER!)
- Code > Insert matching parens/quotes (Personal preference)

Projects

- Handles working directories
- Organizes your work

Packages

- Can use the package manager to install packages
- Can use the manager to load them as well, but not recommended
 - Load packages in your script so you remember which ones you used!

Let's take a look at RStudio

Set up a Project for this course

Your first *real* code!

First Code

```
# First load the package
library(tidyverse)

# Now create the figure
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point()
```

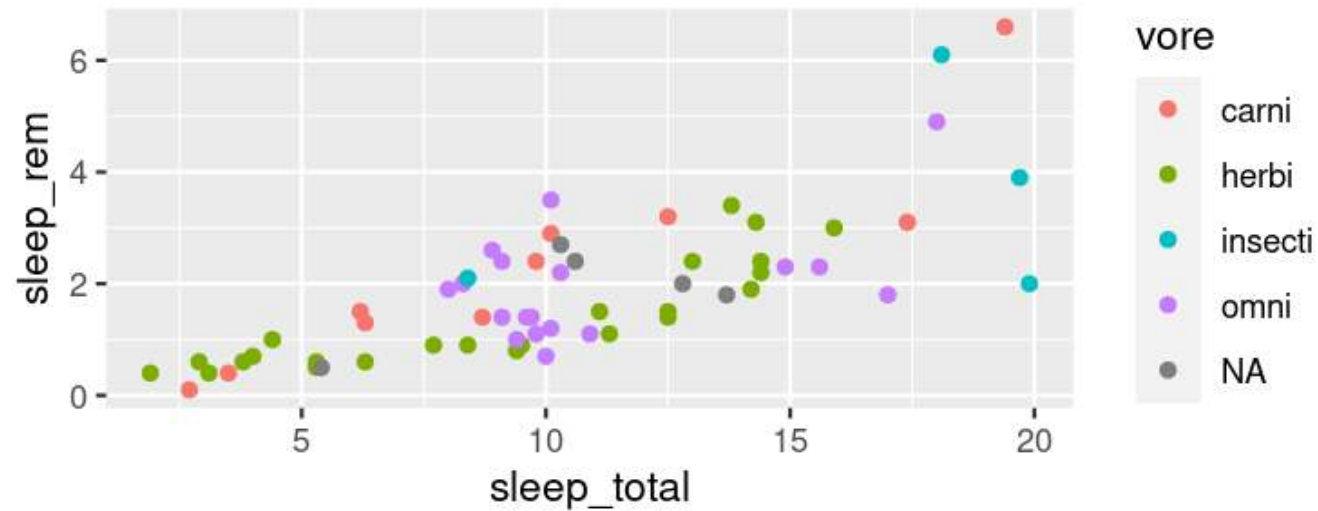
- Copy/paste or type this into the script window in RStudio
 - You may have to go to File > New File > R Script
- Click anywhere on the first line of code
- Use the 'Run' button to run this code, **or** use the short-cut **Ctrl-Enter**
 - Repeat until all the code has run

First Code

```
# First load the package  
library(tidyverse)
```

```
# Now create the figure  
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

```
## Warning: Removed 22 rows containing missing values (geom_point).
```



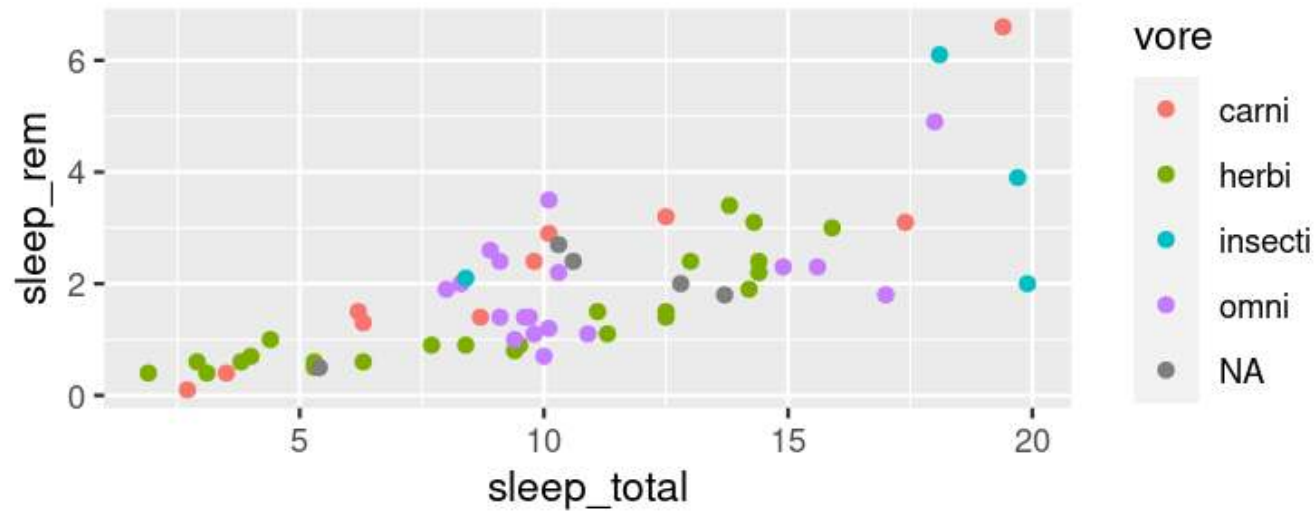
First Code

```
# First load the package  
library(tidyverse)
```

Package
tidyverse

```
# Now create the figure  
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

```
## Warning: Removed 22 rows containing missing values (geom_point).
```



First Code

```
# First load the package
```

```
library(tidyverse)
```

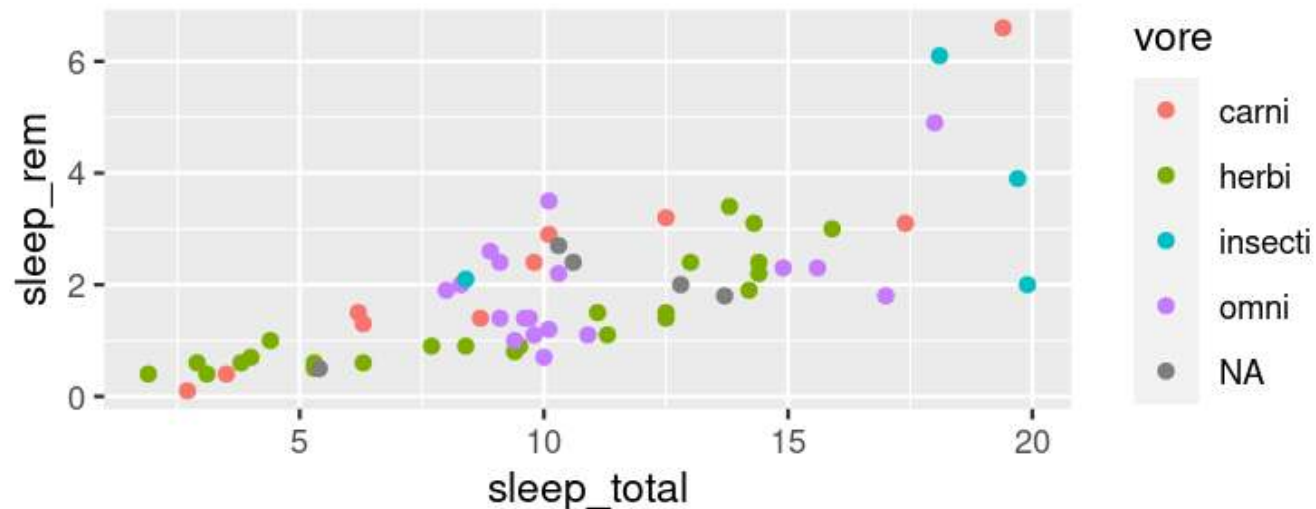
```
# Now create the figure
```

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

```
## Warning: Removed 22 rows containing missing values (geom_point).
```

Functions:

library(), **ggplot()**
aes(), and **geom_point()**



First Code

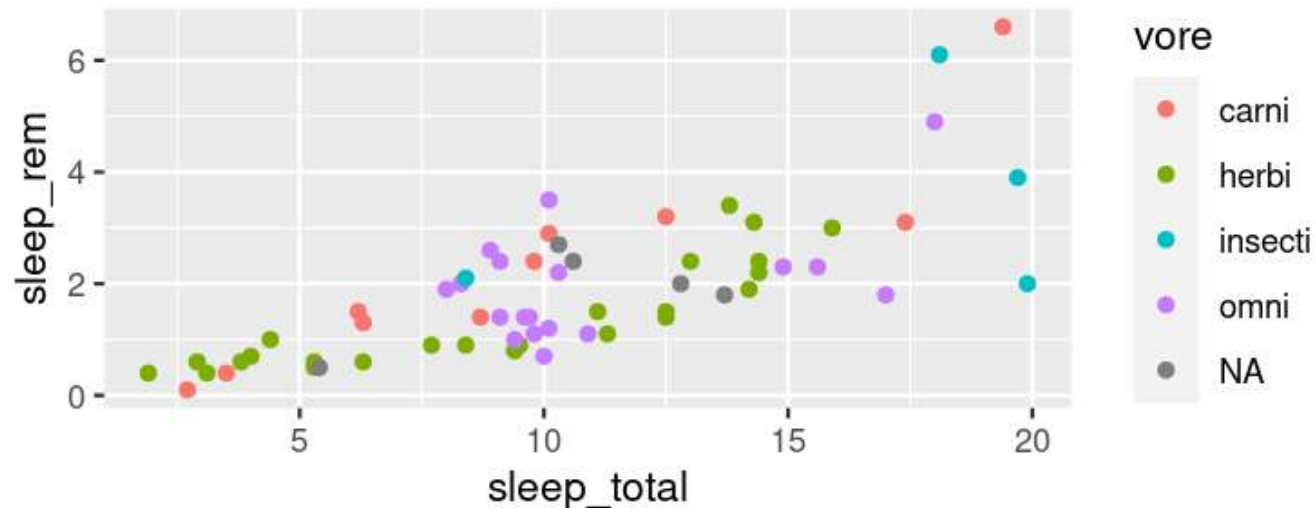
```
# First load the package  
library(tidyverse)
```

```
# Now create the figure
```

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

+
(Specific to **ggplot**)

```
## Warning: Removed 22 rows containing missing values (geom_point).
```

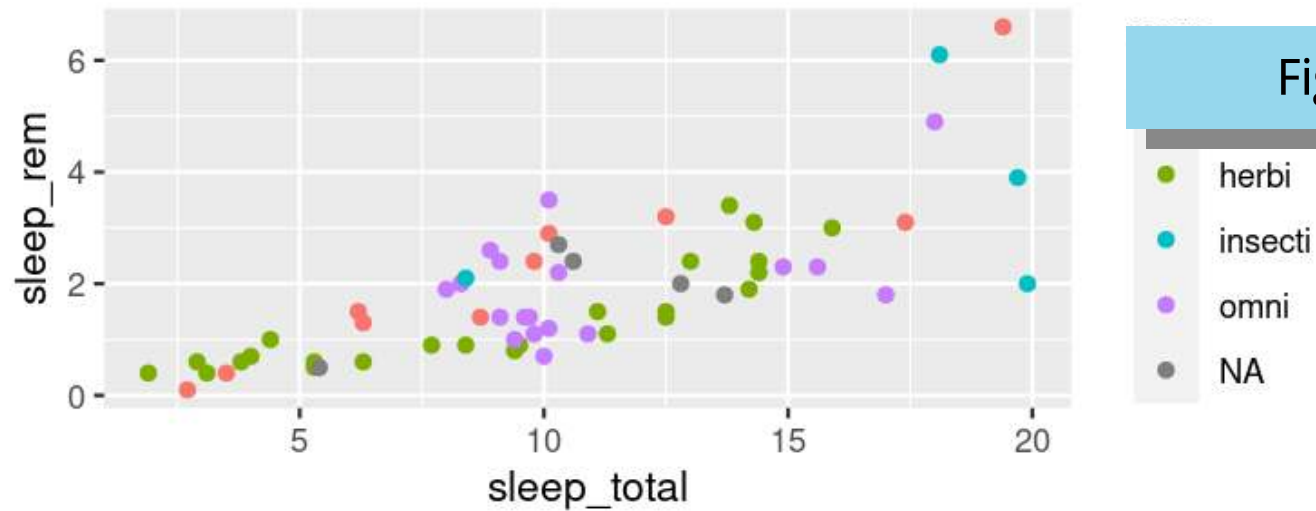


First Code

```
# First load the package  
library(tidyverse)
```

```
# Now create the figure  
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

```
## Warning: Removed 22 rows containing missing values (geom_point).
```



Figure!

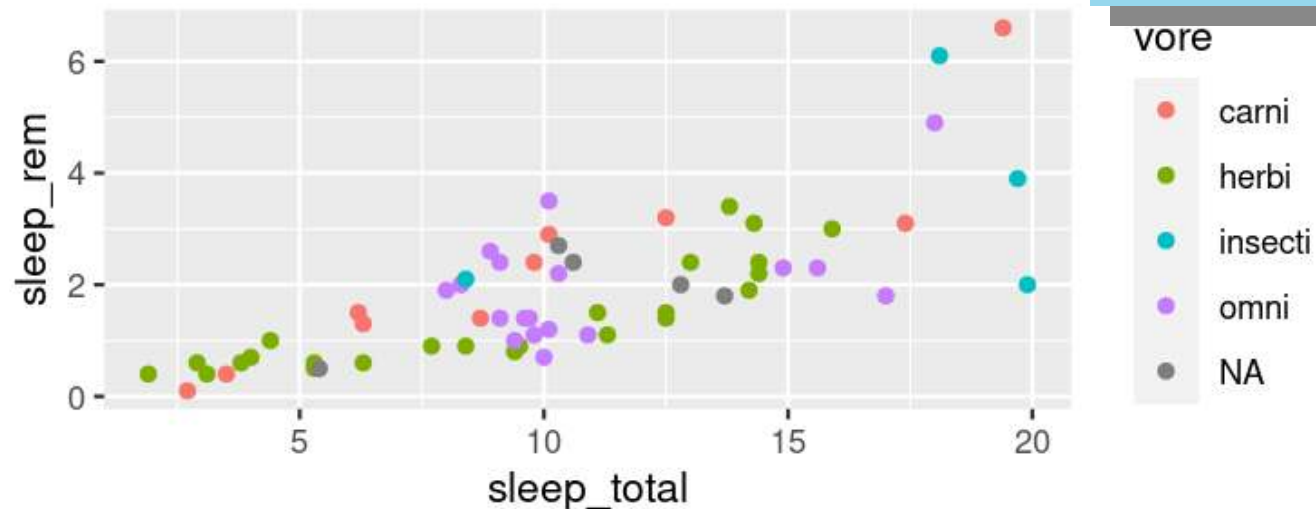
First Code

```
# First load the package  
library(tidyverse)
```

```
# Now create the figure  
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

```
## Warning: Removed 22 rows containing missing values (geom_point).
```

Warning



First Code

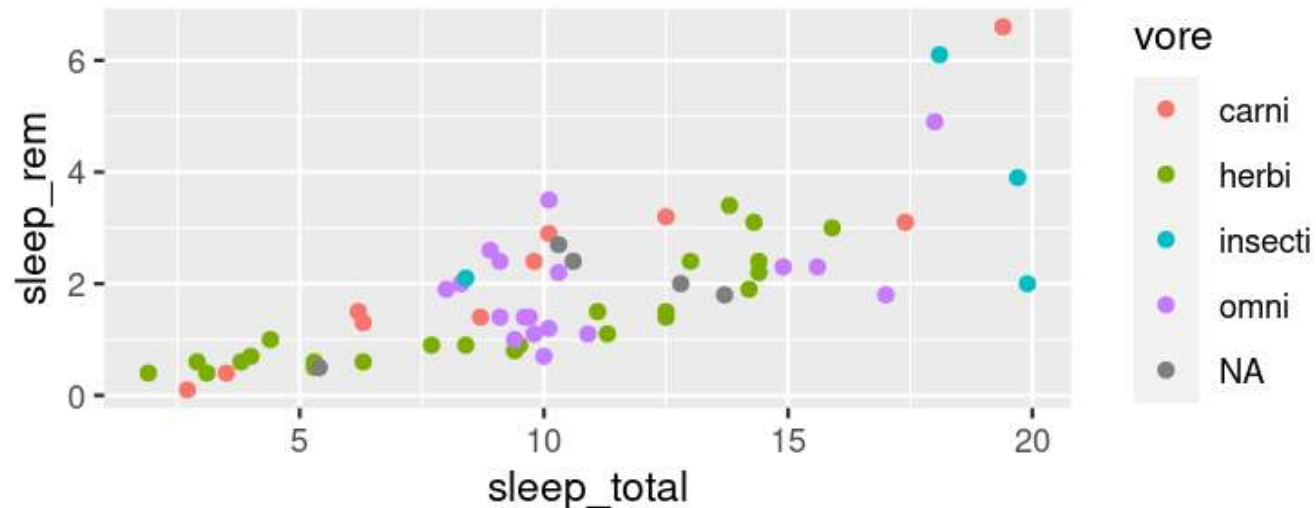
Comments

```
# First load the package  
library(tidyverse)
```

```
# Now create the figure
```

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

```
## Warning: Removed 22 rows containing missing values (geom_point).
```



R Basics: Objects

Objects are *things* in the environment
(Check out the **Environment** pane in RStudio)

functions()

Do things, Return things

Does something but returns nothing

e.g., `write_csv()` - Saves the `mtcars` data frame as a csv file

```
write_csv(mtcars, path = "mtcars.csv")
```

Does something and returns something

e.g., `sd()` - returns the standard deviation of a vector

```
sd(c(4, 10, 21, 55))
```

```
## [1] 22.78157
```

functions()

- Functions can take **arguments** (think 'options')
- **data, x, y, colour**

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

functions()

- Functions can take **arguments** (think 'options')
- **data, x, y, colour**

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

- Arguments defined by **name** or by **position**
- With correct position, do not need to specify by name

By name:

```
mean(x = c(1, 5, 10))
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10))
```

```
## [1] 5.333333
```

functions()

- Functions can take **arguments** (think 'options')
- **data, x, y, colour**

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

- Arguments defined by **name** or by **position**
- With correct position, do not need to specify by name

By name:

```
mean(x = c(1, 5, 10))
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10))
```

```
## [1] 5.333333
```

Note that **c()** is also a function: combine or concatenate

functions()

Watch out for 'hidden' arguments

By name:

```
mean(x = c(1, 5, 10, NA),  
      na.rm = TRUE)
```

```
## [1] 5.333333
```

functions()

Watch out for 'hidden' arguments

By name:

```
mean(x = c(1, 5, 10, NA),  
      na.rm = TRUE)
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10, NA),  
      TRUE)
```

```
## Error in mean.default(c(1, 5, 10, NA), TRUE): 'trim'  
must be numeric of length one
```


functions()

Watch out for 'hidden' arguments

By name:

```
mean(x = c(1, 5, 10, NA),  
      na.rm = TRUE)
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10, NA),  
      TRUE)
```

```
## Error in mean.default(c(1, 5, 10, NA), TRUE): 'trim'  
must be numeric of length one
```

This error states that we've assigned the argument **trim** to a non-valid argument

Where did **trim** come from?

R documentation

```
?mean
```

Your Turn:

Run this, what happens?
Do you see the **trim** argument?

?mean

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

```
## Default S3 method:
```

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

- `x` An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.
- `trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.
- `na.rm` a logical value indicating whether NA values should be stripped before the computation proceeds.
- `...` further arguments passed to or from other methods.

Data

Generally kept in **vectors** or **data.frames/tibbles**

- These are objects with names (like functions)
- We can use `<-` to assign values to objects (assignment)

Vector (1 dimension)

```
a <- c("a", "b", "c")
```

```
a
```

```
## [1] "a" "b" "c"
```

Data frame (2 dimensions)

rows x columns

```
d <- data.frame(letters = c("a", "b", "c"),  
                numbers = c(1, 2, 3),  
                treat = c("control", "control",  
                          "control"))
```

```
d
```

```
##   letters numbers  treat  
## 1      a        1 control  
## 2      b        2 control  
## 3      c        3 control
```

Vectors

Use `c()` to create a vector

```
a <- c("apples", 12, "bananas")
```

Use `x[index]` to access part of a vector

```
a[3] # [1] "bananas"
```

Vectors contain one type of variable

(Even if you try to make it with more)

```
class(a) # [1] "character"
```

Data frames (also tibbles)

Create with `data.frame()/tibble()`

```
my_data <- tibble(x = c("s1", "s2", "s3",  
  "s4"),  
                 y = c(101, 102, 103, 104),  
                 z = c("a", "b", "c", "d"))  
  
my_data
```

```
## # A tibble: 4 × 3  
##   x           y z  
##   <chr> <dbl> <chr>  
## 1 s1      101 a  
## 2 s2      102 b  
## 3 s3      103 c  
## 4 s4      104 d
```

(**dbl** = "Double" = Computer talk for non-integer number)

Data frames (also tibbles)

Create with `data.frame()/tibble()`

```
my_data <- tibble(x = c("s1", "s2", "s3",  
  "s4"),  
                 y = c(101, 102, 103, 104),  
                 z = c("a", "b", "c", "d"))  
  
my_data
```

```
## # A tibble: 4 × 3  
##   x           y z  
##   <chr> <dbl> <chr>  
## 1 s1      101 a  
## 2 s2      102 b  
## 3 s3      103 c  
## 4 s4      104 d
```

(**dbl** = "Double" = Computer talk for non-integer number)

Cols have different types of variables

```
str(my_data)
```

```
## tibble [4 × 3] (S3: tbl_df/tbl/data.frame)  
## $ x: chr [1:4] "s1" "s2" "s3" "s4"  
## $ y: num [1:4] 101 102 103 104  
## $ z: chr [1:4] "a" "b" "c" "d"
```

Data frames (also tibbles)

x\$colname to pull out column

```
my_data$x
```

```
## [1] "s1" "s2" "s3" "s4"
```

Or use **pull()** (from **tidyverse**)

```
pull(my_data, x)
```

```
## [1] "s1" "s2" "s3" "s4"
```


Data frames (also tibbles)

`x$colname` to pull out column

```
my_data$x
```

```
## [1] "s1" "s2" "s3" "s4"
```

Or use `pull()` (from `tidyverse`)

```
pull(my_data, x)
```

```
## [1] "s1" "s2" "s3" "s4"
```

`x[row, col]` to access rows and columns of a data frame

```
my_data[1:2, 2:3]
```

```
## # A tibble: 2 × 2
```

```
##       y z
```

```
##   <dbl> <chr>
```

```
## 1    101 a
```

```
## 2    102 b
```

Your Turn: Vectors and Data frames

1) Create a vector with 5 numbers and look at it

- Find it in the "Global Environment" pane (upper right)
- Type its name in the console and hit enter

```
my_vec <- c(1, 2, 3, 4, 5)  
my_vec
```

2) Create a data frame with `data.frame()` or `tibble()`

- Click on it's name in the "Global Environment"
- Type its name in the console and hit enter

```
my_df <- data.frame(name = c("John", "Jane", "Bob"),  
                    age = c(30, 25, 40))  
my_df
```

Miscellaneous

R has spelling and punctuation

- R cares about spelling
- R is also case sensitive! (**Apple** is not the same as **apple**)
- Commas are used to separate arguments in functions

For example

This is correct:

```
mean(c(5, 7, 10)) # [1] 7.333333
```

This is **not** correct:

```
mean(c(5 7 10))
```

```
## Error: <text>:1:10: unexpected numeric constant
## 1: mean(c(5 7
##                ^
```

R has spelling and punctuation

- R cares about spelling
- R is also case sensitive! (**Apple** is not the same as **apple**)
- Commas are used to separate arguments in functions

For example

This is correct:

```
mean(c(5, 7, 10)) # [1] 7.333333
```

>80% of learning R is learning to
troubleshoot

This is **not** correct:

```
mean(c(5 7 10))
```

```
## Error: <text>:1:10: unexpected numeric constant
## 1: mean(c(5 7
##                ^
```

R has spelling and punctuation

Spaces usually don't matter unless they change meanings

```
5>=6      # [1] FALSE
5 >=6     # [1] FALSE
5 >= 6    # [1] FALSE
5 > = 6   # Error: unexpected '=' in "5 > ="
```

Periods don't matter either, but can be used in the same way as letters

(But for complex programming reasons... don't)

```
apple.oranges <- "fruit"
```

Assignments and Equal signs

Use `<-` to assign values to objects

```
a <- "hello"
```

Use `=` to set function arguments

```
mean(x = c(4, 9, 10))
```

Use `==` to determine equivalence (logical)

```
10 == 10 # [1] TRUE  
10 == 9  # [1] FALSE
```

Braces/Brackets

Round brackets: ()

- Run functions (even if there are no arguments)

```
Sys.Date() # Get the Current Date
```

```
## [1] "2021-09-15"
```


Braces/Brackets

Round brackets: ()

- Run functions (even if there are no arguments)

```
Sys.Date() # Get the Current Date
```

```
## [1] "2021-09-15"
```

- Without the (), R spits out information on the function:

```
Sys.Date
```

```
## function ()  
## as.Date(as.POSIXlt(Sys.time()))  
## <bytecode: 0x55f6f0eb6cb8>  
## <environment: namespace:base>
```

Braces/Brackets

Round brackets: ()

- Run functions (even if there are no arguments)

```
Sys.Date() # Get the Current Date
```

```
## [1] "2021-09-15"
```

- Without the (), R spits out information on the function:

```
Sys.Date
```

```
## function ()  
## as.Date(as.POSIXlt(Sys.time()))  
## <bytecode: 0x55f6f0eb6cb8>  
## <environment: namespace:base>
```

() must be associated with a **function**

(Well, *almost* always)

Braces/Brackets

Square brackets: []

- Extract parts of objects

```
LETTERS
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
```

```
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
LETTERS[1]
```

```
## [1] "A"
```

```
LETTERS[26]
```

```
## [1] "Z"
```

Braces/Brackets

Square brackets: []

- Extract parts of objects

```
LETTERS
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
```

```
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
LETTERS[1]
```

```
## [1] "A"
```

```
LETTERS[26]
```

```
## [1] "Z"
```

[] have to be associated with an **object**
that has dimensions

(Always)

Improving code readability

Use spaces like you would in sentences:

```
a <- mean(c(4, 10, 13))
```

is easier to read than

```
a<-mean(c(4,10,13))
```

(But they are equivalent, coding-wise)

Improving code readability

Don't be afraid to use line breaks ('Enters') to make the code more readable

```
a <- data.frame(exp = c("A", "B", "A", "B", "A", "B"),  
                sub = c("A1", "A1", "A2", "A2", "A3", "A3"),  
                res = c(10, 12, 45, 12, 12, 13))
```

VS.

```
a <- data.frame(exp = c("A", "B", "A", "B", "A", "B"), sub = c("A1", "A1", "A2", "A2", "A3", "A3"),  
                res = c(10, 12, 45, 12, 12, 13))
```

Reproducible research

What is reproducible research?

Remembering what you've done (and sharing)

- Keep scripts
- Annotate scripts (use comments)
- Date scripts!
- Compile scripts into reports or notebooks
- Include version information
 - `devtools::session_info()`

We can use the "Compile Report" button in RStudio to create an HTML report of your work

tidyverse?

R base vs. tidyverse

R base

- R base is basic R
- Most packages used are installed and loaded by default

R base vs. tidyverse

R base

- R base is basic R
- Most packages used are installed and loaded by default

tidyverse

- Collection of 'new' packages developed by a team closely affiliated with RStudio
- Packages designed to work well together
- Use a slightly different syntax
- Among others, includes packages used for data transformations and visualizations:
 - e.g., **ggplot2**, **dplyr**, **tidyr**

R base vs. tidyverse

R base

- R base is basic R
- Most packages used are installed and loaded by default

tidyverse

- Collection of 'new' packages developed by a team closely affiliated with RStudio
- Packages designed to work well together
- Use a slightly different syntax
- Among others, includes packages used for data transformations and visualizations:
 - e.g., **ggplot2**, **dplyr**, **tidyr**

Can be helpful to understand whether functions are **tidyverse** or R base functions

Wrapping up: Further reading

- <http://www.cookbook-r.com>
- [R for Data Science](#)
- [R base cheatsheet](#)