Data Science for the Modern Exercise Physiologist

CSEP 2021

RStudio Tutorial Slides

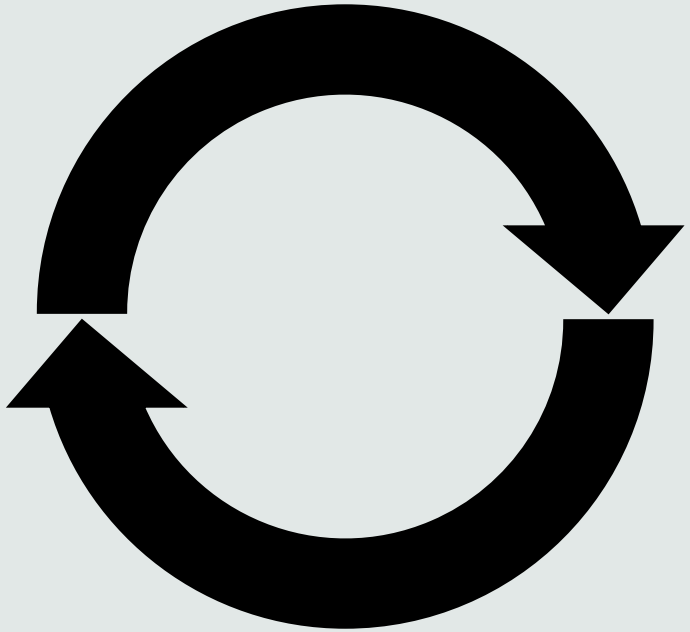
Lesson Plan

1. The Big Picture
2. Overview of the R Studio Interface
3. Basic language
4. Core functions
5. Interactive activity
6. Tidyverse functions
7. Sample dataset exploration

The Big Picture

Data Management

- Data sheets
- Wrangling
- Analysis



Data Collection

- Acquisition systems
- Data types
- Volume

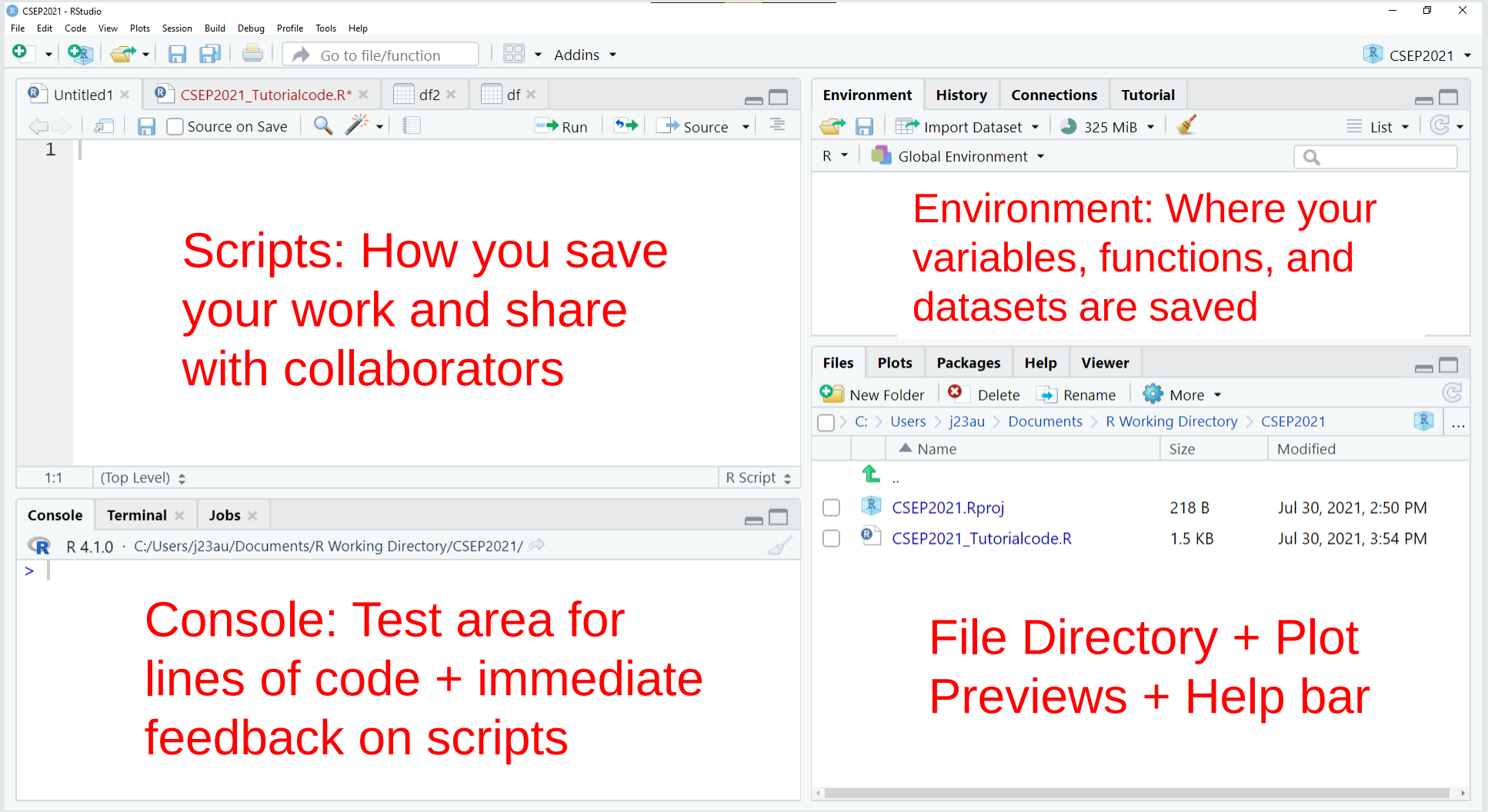
Visualization

- Data exploration
- Figures
- Tables

Reporting

- Repositories
- Data sharing

R Studio Interface



Head to your R Script to follow along

The following slides are reference material for after the symposium

Definitions

Console: Immediately executes code; acts as a test ground for functions

Scripts: Blocks of executable code that can be save and improved over time – bulk of your work

R Notebooks: Allows natural switching between written text and executable code, all within the same file

- Great way to communicate results with colleagues

Projects: File sorting system to organize your studies

- Projects load up with their own workspace and the last scripts you were working on

Basic symbols

Syntax

<- (variable naming)

= (arguments)

( ) (functions)

[ ] (indexing [row,col])

{} (if, else, loops)

Logicals (TRUE = 1, FALSE = 0)

==	exactly equals
!=	does not equal
&&	and
< > >= <=	greater than / less than

Anything following a # is text and will not execute code

?[function] (Help; MOST IMPORTANT!!)

Basic terminology

Values	
a	"words"
b	3.14
c	4
d	TRUE

Variable types:

- **Character**: String-type variables (words)
- **Numeric**: Any number type
- **Integer**: Whole numbers
- **Logicals**: TRUE or FALSE
- **Factor**: A character classifier (e.g., sedentary, active, fit)

Data structure types

- **Vectors**: Group of variables of the same type (i.e., all words, all numbers)
- **Data frames**: Data tables with rows as observations and columns as different variables (i.e., an excel sheet)

Packages and Libraries

R is open source: Anyone can make a package and make it available online

- To access these specialized functions, you need to install the packages and load the specific library
- You only have to install the package once, but you need to access the library each time you start an RStudio session

EXAMPLES

install.packages(“tidyverse”)
library(tidyverse)

install.packages(“gameofthrones”)
library(gameofthrones)

Notice when you do and do not use “quotations”!

Functions

Function: A pre-programmed instruction that takes in input (arguments) and provides an output

- You are likely already familiar with functions from excel!
- Functions work very similar in R, but may be complicated

ID	VO2max
S01	52
S02	36
S03	34
S04	43
S05	38
S06	39
S07	32
S08	33
S09	42
S10	37
MEAN	=AVERAGE(C3:C12
SD	

AVERAGE(**number1**, [number2], ...)

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)
```

e.g., read.csv() is the function, and everything in the brackets are the ‘arguments’ you need to provide to give it instructions

Core functions (examples)

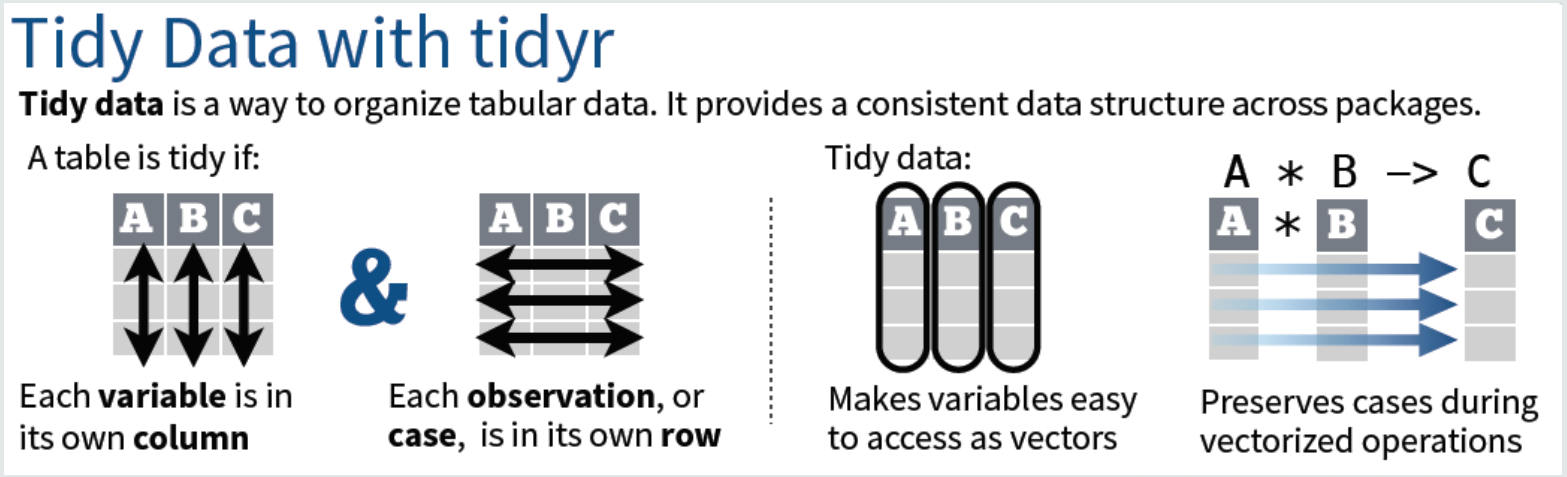
Function	Description	Example
seq()	Create a sequence of numbers	seq(0, 100, by = 10)
c()	Concatenates a bunch of elements	c(2, 4, 6, 8, 10) c(“R “, “is “, “fun!”)
paste()	Concatenates two character strings together	paste(1:3, c(“st”, “nd”, “rd”) - “1st”, “2nd”, “3rd”
rnorm()	Creates a random normal distribution	rnorm(n=100, mean = 250, sd=20) - 100 observations with a mean of 250 and SD of 20)
sample()	Randomly re-organizes a vector	sample(c(“A”,”B”,”C”)) [1] “B”,”C”,”A”
as.integer() as.character() factor()	Converts to integer Converts to character Converts to a factor	
data.frame()	Combines vectors into a data frame object	data.frame(v1,v2)

Tidyverse Functions

The 'Tidyverse' is a collection of very useful functions that are widely used across data science

What is 'tidy' data?

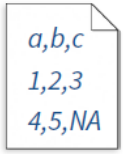
- Each variable is in a column
- Each observation is a row
- Each value is a cell



Readr: Functions for importing data

Read Tabular Data - These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
  n_max), progress = interactive())
```



A	B	C
1	2	3
4	5	NA

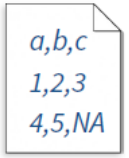
Comma Delimited Files

read_csv("file.csv")

To make file.csv run:

```
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")
```

USEFUL ARGUMENTS



Example file

```
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"
```

A	B	C
1	2	3
4	5	NA

No header

read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header

read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines

read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset

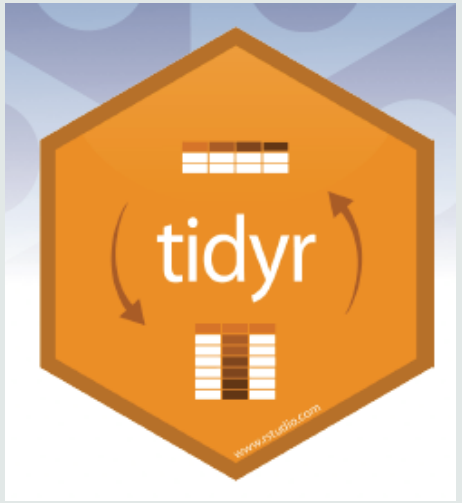
read_csv(f, **n_max = 1**)

A	B	C
NA	2	3
4	5	NA

Missing Values

read_csv(f, **na = c("1", ".")**)

Tidyr: Functions for tidying data



Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Dplyr: Functions for wrangling data

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**
Extract rows that meet logical criteria.

dplyr::**distinct(iris)**
Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**
Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**
Randomly select n rows.

dplyr::**slice(iris, 10:15)**
Select rows by position.

dplyr::**top_n(storms, 2, date)**
Select and order top n entries (by group if grouped data).

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**
Select columns by name or helper function.

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first**
First value of a vector.

dplyr::**last**
Last value of a vector.

dplyr::**nth**
Nth value of a vector.

dplyr::**n**
# of values in a vector.

dplyr::**n_distinct**
# of distinct values in a vector.

IQR
IQR of a vector.

min
Minimum value in a vector.

max
Maximum value in a vector.

mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

sd
Standard deviation of a vector.

Dplyr: Functions for wrangling data

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Make New Variables

dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.

dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

Group Data

dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)
Compute separate summary row for each group.

ggplot2: Grammar of Graphics plots

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

data **geom** x = F · y = A **coordinate system** **plot**

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

data **geom** x = F · y = A color = F size = A **coordinate system** **plot**

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

discrete x , continuous y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight