PSYC 259: Principles of Data Science

Week 2: File Organization and Workflow

Today

1. Project structure principles
   a. File/folder organization
   b. Version control
2. Advice: How to get programming help
3. BREAK
4. R language basics and importing tutorial
5. Getting started on homework

Project Structure Principles

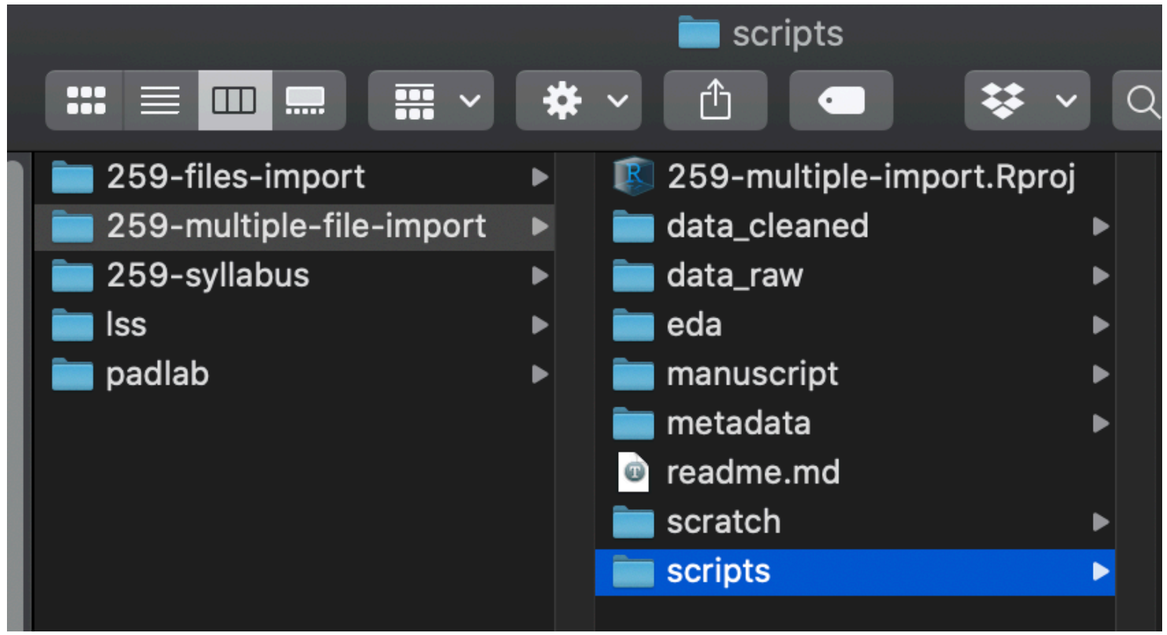
Two principles of project workflow

1. Folder organization creates rules and defines a workflow; establishes a location from which to build relative file paths
2. Version control tracks file history without duplication/clutter; allows for collaboration/derivation/experimentation

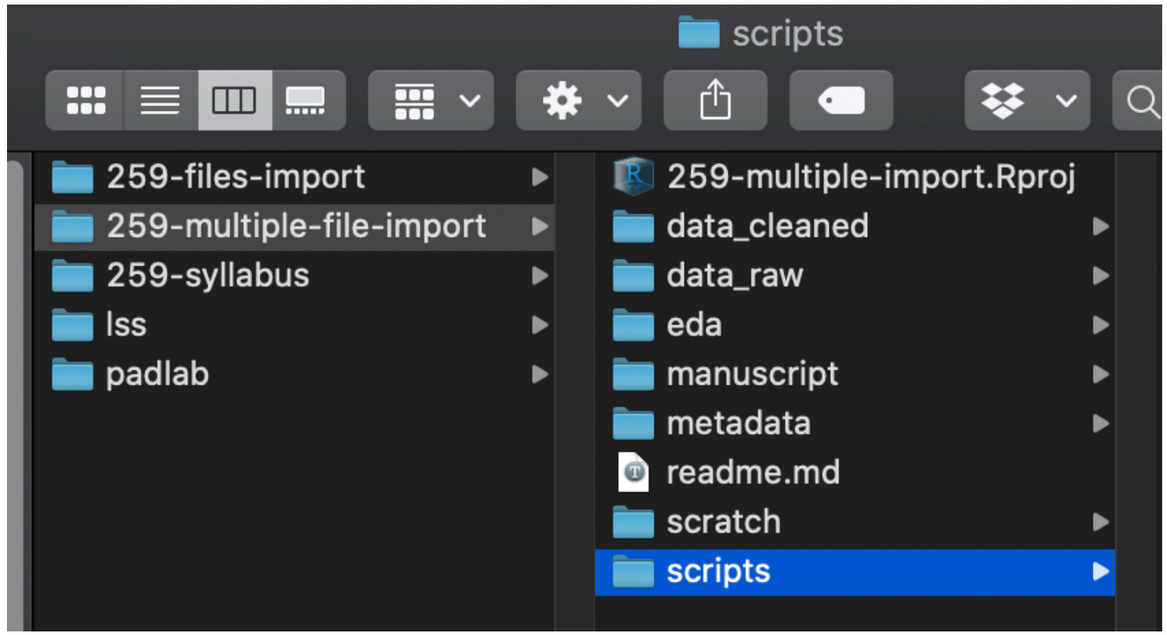
What principles should guide project structure?

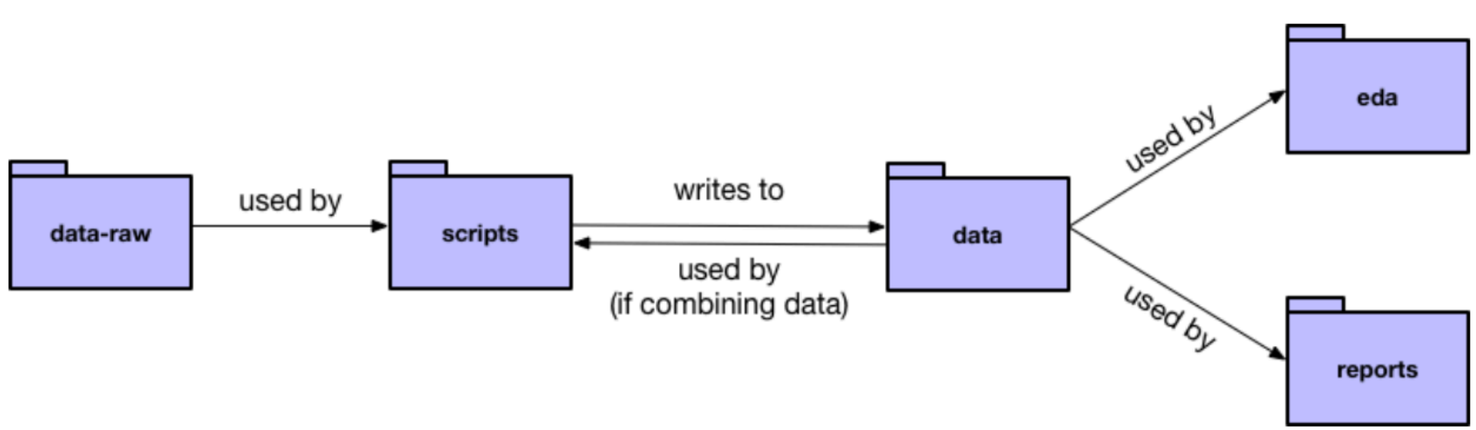
#1 File/folder organization

Folders should organize similar file **types** (w/in a root project folder)



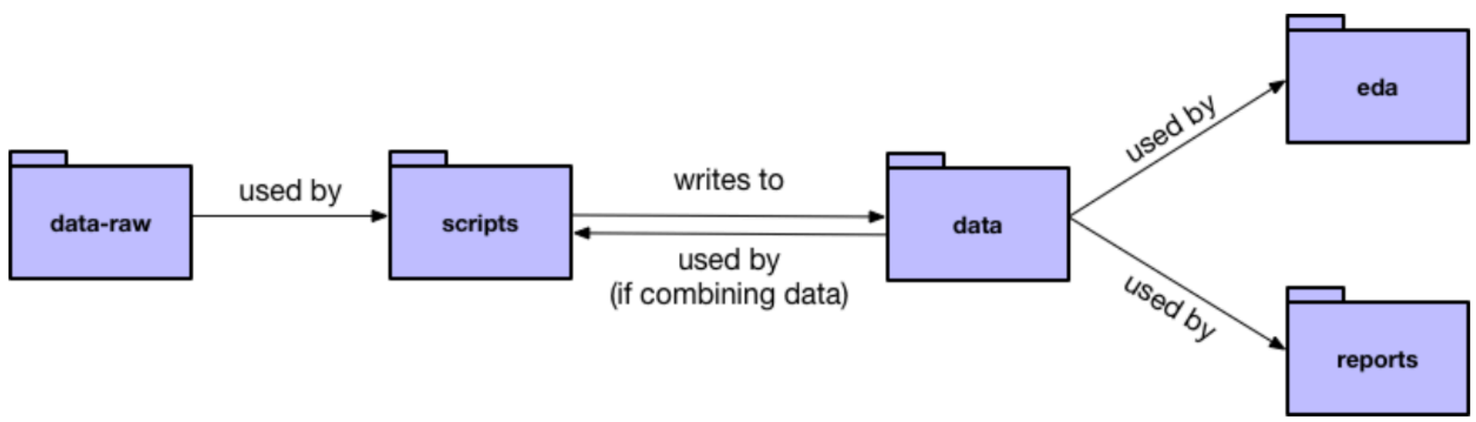
Folders should organize similar file **types** (w/in a root project folder)





Avoid having more nodes in this chart than are necessary

- Every intermediate step incurs a cost of maintenance
  - Think about what has to be re-run if data-raw changes
- Multiple endpoints don't incur cost



What is an R Studio Project?

- A project folder (directory) with some files that keep tabs on your command history (.RProj)
  - Open the RProj file to open the project
  - Can turn existing folders into projects
- What is real (e.g., persists after shutting down R)?
  - Not real: objects/data frames in your workspace
  - Real: data files and the scripts (.R files) used to work with them

Accessing files from your working directory

- Your RStudio Project folder is your default *working directory*
  - working directory: where R can find files
  - getwd() probably gives you something ugly like `/Users/johnfranchak/Documents/GitHub/project_folder`
- *Absolute file paths* like that should be avoided at all costs!
  - Absolute file paths don't transfer between computers with different user names or different operating systems
  - Bad for reproducibility, extensibility, and sharing
- But don't you need them to get into all of those directories you just told me I need to use?

Relative files paths 💪💪💪

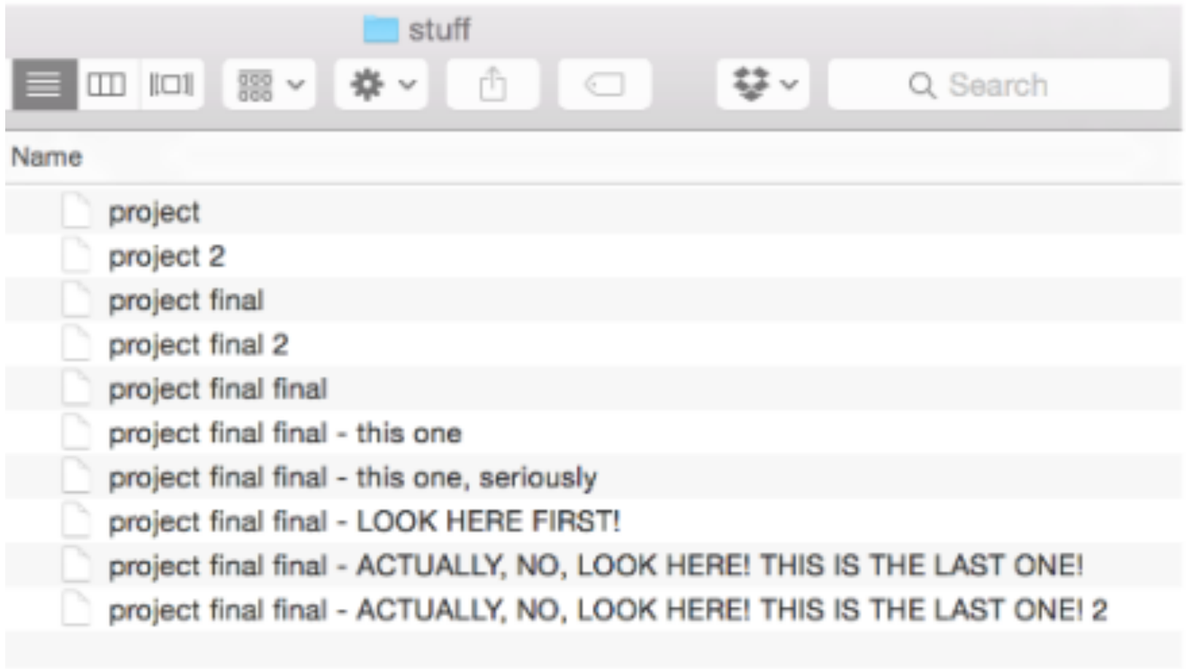
- Just tell R what to look for relative to your project (working) directory!
- append subfolder name to filename
  - “folder1/filename”
- *here* package detects the project directory and composes filenames from/to any folder
  - `here(“folder1”,”folder2”,”filename")`

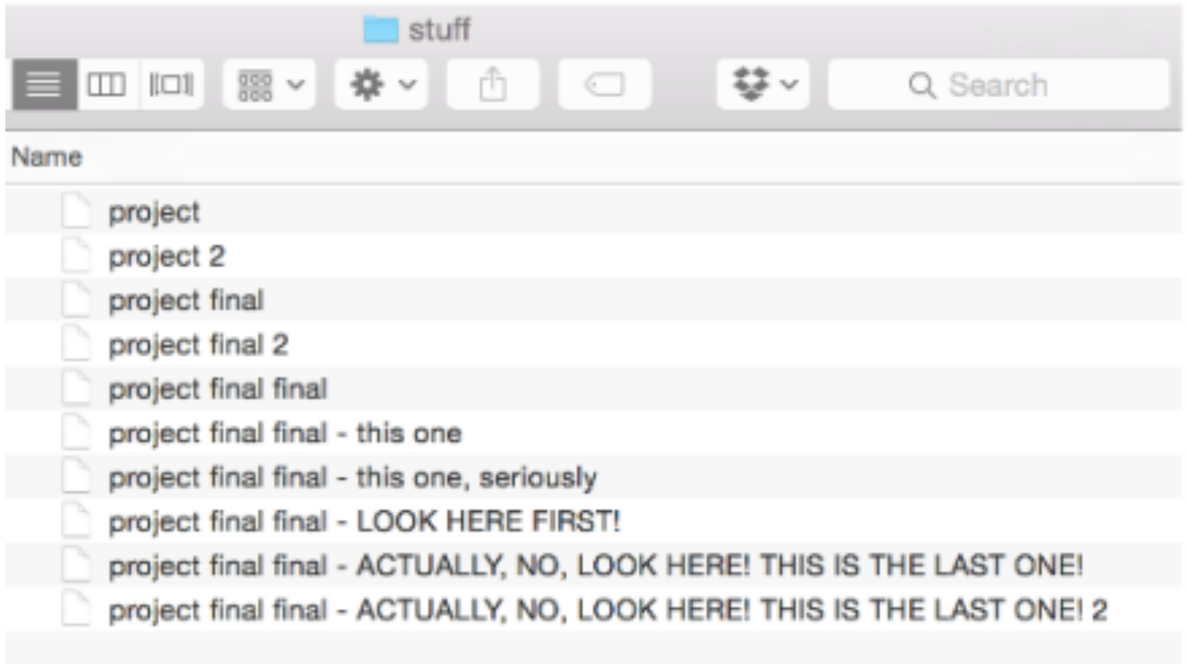
Other considerations

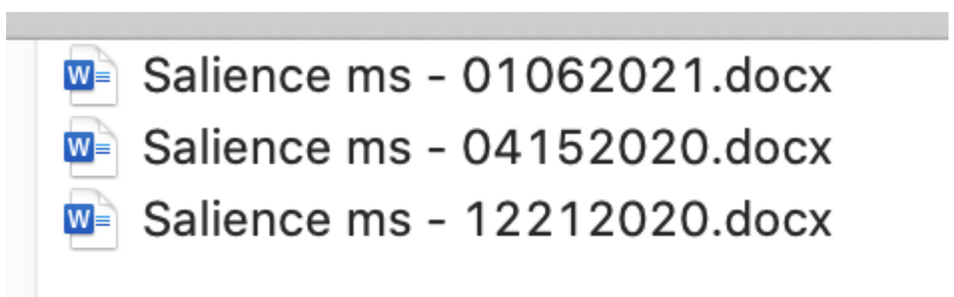
- Not everything can be automated, but you can try to limit human data entry to a single master table
  - Track notes about sessions, inclusion/exclusion info
  - Keep as part of project metadata, and use it to direct your scripts to pull the ‘right’ data
- Not every project can be contained in a local directory on a single computer (or on github)
  - Large datasets might need other solutions
  - “data_raw” might need to be “data_slightly_cooked”

What principles should guide project structure?

#2 Version Control


stuff
Search
Name
project
project 2
project final
project final 2
project final final
project final final - this one
project final final - this one, seriously
project final final - LOOK HERE FIRST!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE! 2


stuff
Search
Name
project
project 2
project final
project final 2
project final final
project final final - this one
project final final - this one, seriously
project final final - LOOK HERE FIRST!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE! 2


Salience ms - 01062021.docx
Salience ms - 04152020.docx
Salience ms - 12212020.docx

stuff

Search

Name

- project
- project 2
- project final
- project final 2
- project final final
- project final final - this one
- project final final - this one, seriously
- project final final - LOOK HERE FIRST!
- project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE!
- project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE! 2

- Salience ms - 01062021.docx
- Salience ms - 04152020.docx
- Salience ms - 12212020.docx

- SUBMISSION1-JEP
- SUBMISSION2-APP
- SUBMISSION3-APP
- SUBMISSION4-APP

- 2015-11-13-Recalibration.docx
- 2015-12-08-Recalibration.docx
- 2015-12-09-Recalibration.docx
- 2015-12-10-Recalibration.docx
- 2016-01-12-Recalibration.docx
- 2016-01-13-Recalibration.docx
- 2016-01-14-Recalibration.docx
- 2016-01-15-Recalibration.docx
- 2016-01-18-Recalibration.docx
- 2016-01-20-Recalibration.docx
- 2016-01-21-Recalibration.docx
- 2016-01-22-Recalibration.docx
- 2016-01-25-Recalibration.docx
- 2016-01-26-Recalibration.docx
- 2016-01-27-Recalibration.docx
- 2016-01-28-Recalibration.docx
- 2016-01-29-Recalibration.docx
- 2016-02-09-Recalibration.docx
- 2016-02-10-Recalibration.docx
- 2016-02-11-Recalibration.docx
- 2016-02-12-Recalibration.docx
- 2016-02-14-Recalibration.docx
- 2016-02-15-Recalibration.docx
- 2016-02-16-Recalibration.docx
- 2016-02-17-Recalibration.docx
- 2016-02-17b-Recalibration.docx
- 2016-02-18-Recalibration.docx
- 2016-02-19-Recalibration-KSK.docx
- 2016-02-19-Recalibration.docx
- 2016-02-23-Recalibration-DR.docx
- 2016-03-01-Recalibration.docx
- 2016-03-01-Recalibrationb.docx
- 2016-03-02-Recalibration.docx
- 2016-03-03-Recalibration-KSK.docx
- 2016-03-03-Recalibration.docx
- 2016-03-04-Recalibration.docx

Why do we do this to ourselves?

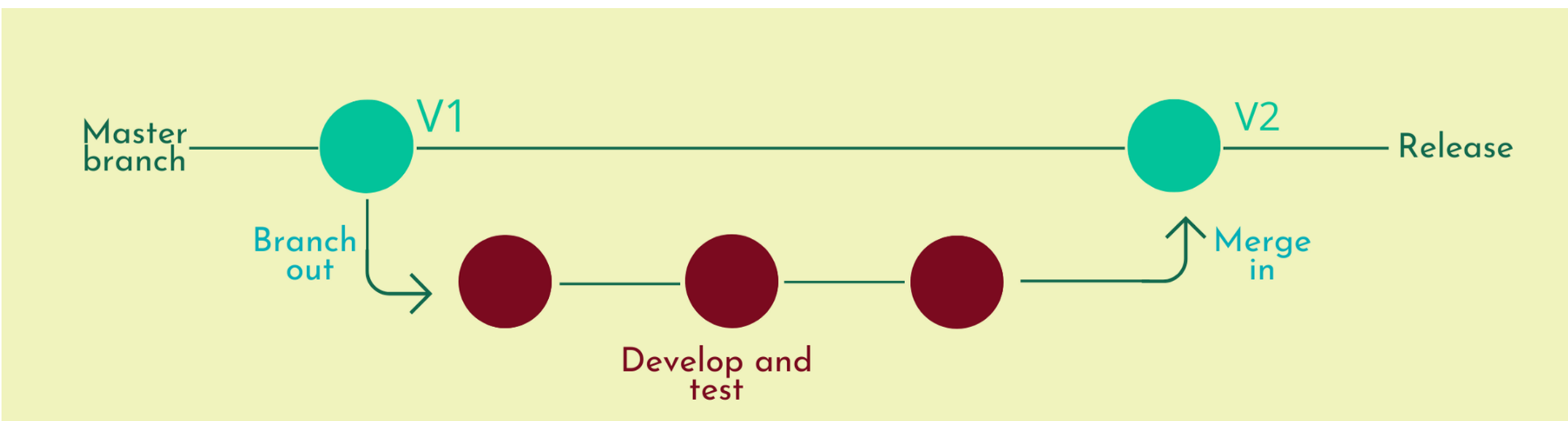
- Want to preserve the document history
  - Easier to take risks, delete things, etc. if you know you can go back (especially for writing code)
  - For collaboration, want to know who made changes
- But it's messy, inefficient, and easy to break
  - Cluttered file folders (imagine if you wanted to track every change to every type of file)
  - Copy -> paste -> rename is tedious
  - No record of *what* the changes were

The solution: Use version control software (e.g., git, svm)

- Define a **repository** (your project folder) and tell git which files to **track** vs **ignore**
- Place your repository on a cloud hub (e.g., GitHub)
- Git tracks what/when changes were made, and you tell git whether to **commit** those changes
- You can **push** those changes to the central hub to store them (now they're saved)
- Other users/computers can **pull** those changes from the hub to keep their local copy up-to-date

More advanced git features

- Create branches (alternate timelines) to develop and test new functions, then merge those changes back to the master branch when they're ready



Other considerations

- Easiest to get started working on a solo project
- Collaborative version control can get messy if people forget to commit/push their changes or to pull the most recent version from the master repo
- git wants your folders out of dropbox or other cloud synced drives, so you have to commit to using it
- Lots of ways to use git (Github website, Github app, RStudio git window, command line)
- Public vs. private repos
- Flat files vs. binary files

Essential git skills to master

- Forking repos - creating a copy of a repo that points to the original [DONE]
- Cloning a repo - pulling a version from the cloud onto your local system [DONE]
- Pulling changes from the cloud
- Committing and pushing changes to the cloud
- Discarding unwanted changes
- Creating a new repo from an existing local directory

https://github.com/2025-PSYC-259/259-langbasics-importing

259-langbasics-importing

Public

forked from psych-259-data-science-2022/259-langbasics-importing

Watch 0 | Fork 1 | Star 0

Fork your own copy of 2025-PSYC-259/259-langbasics-importing

master | 1 Branch | 0 Tags | Go to file t | Add file | Code

This branch is 3 commits ahead of `psych-259-data-science-2022/259-langbasics-importing:master`.

JohnFranchak fixed demos ✓ 88fdf1c · last week 33 Commits

data_cleaned	made a standalone version of the script examples	4 years ago
data_raw	made a standalone version of the script examples	4 years ago
docs	fixed demos	last week
.gitignore	created flipbook	4 years ago
259-langbasics-importing.Rproj	created flipbook	4 years ago
README.md	2025	3 weeks ago
langbasics-importing.R	fixed demos	last week
xaringan-themer.css	2025	3 weeks ago

About

Introduction to R data importing

2025-PSYC-259.github.io/259-langba...

Readme

Activity

Custom properties

0 stars

0 watching

1 fork

Report repository

Releases

No releases published

Packages

No packages published

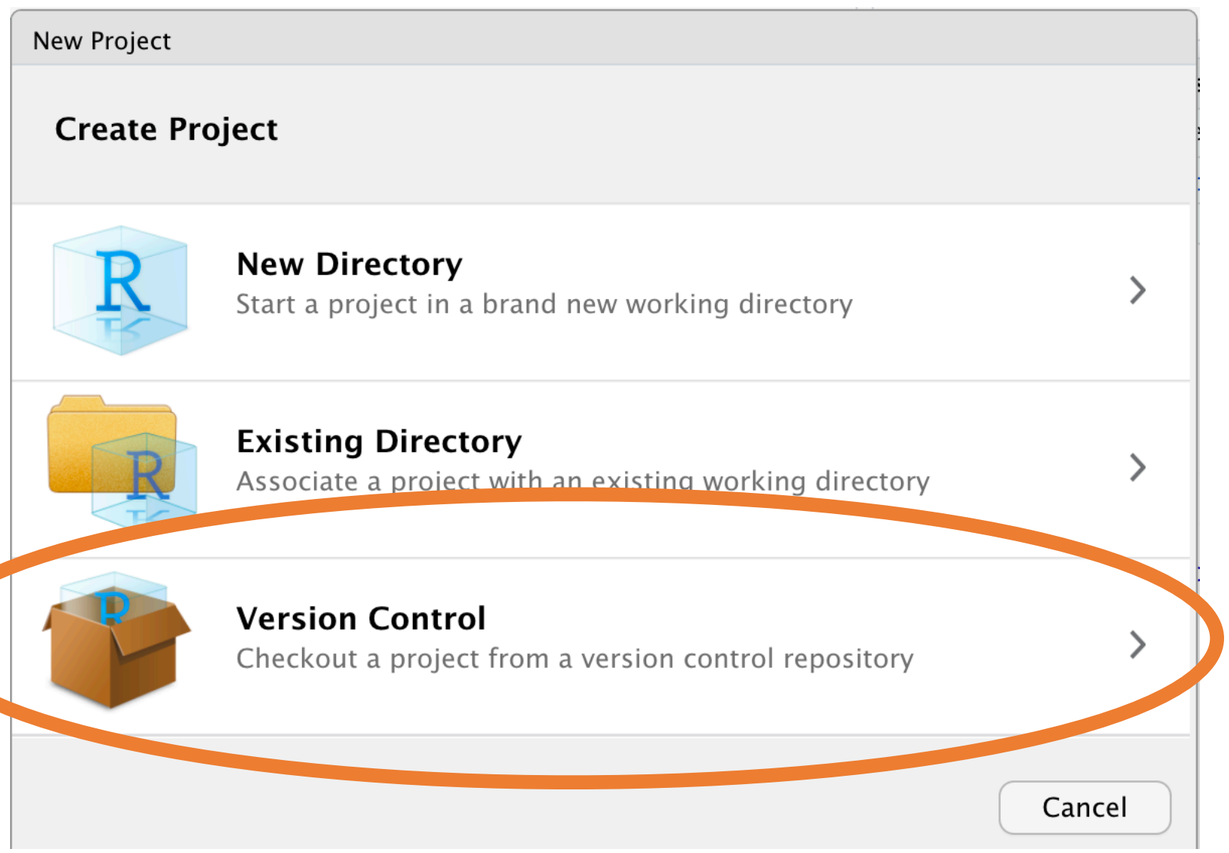
Fork a repository to make your own copy of it

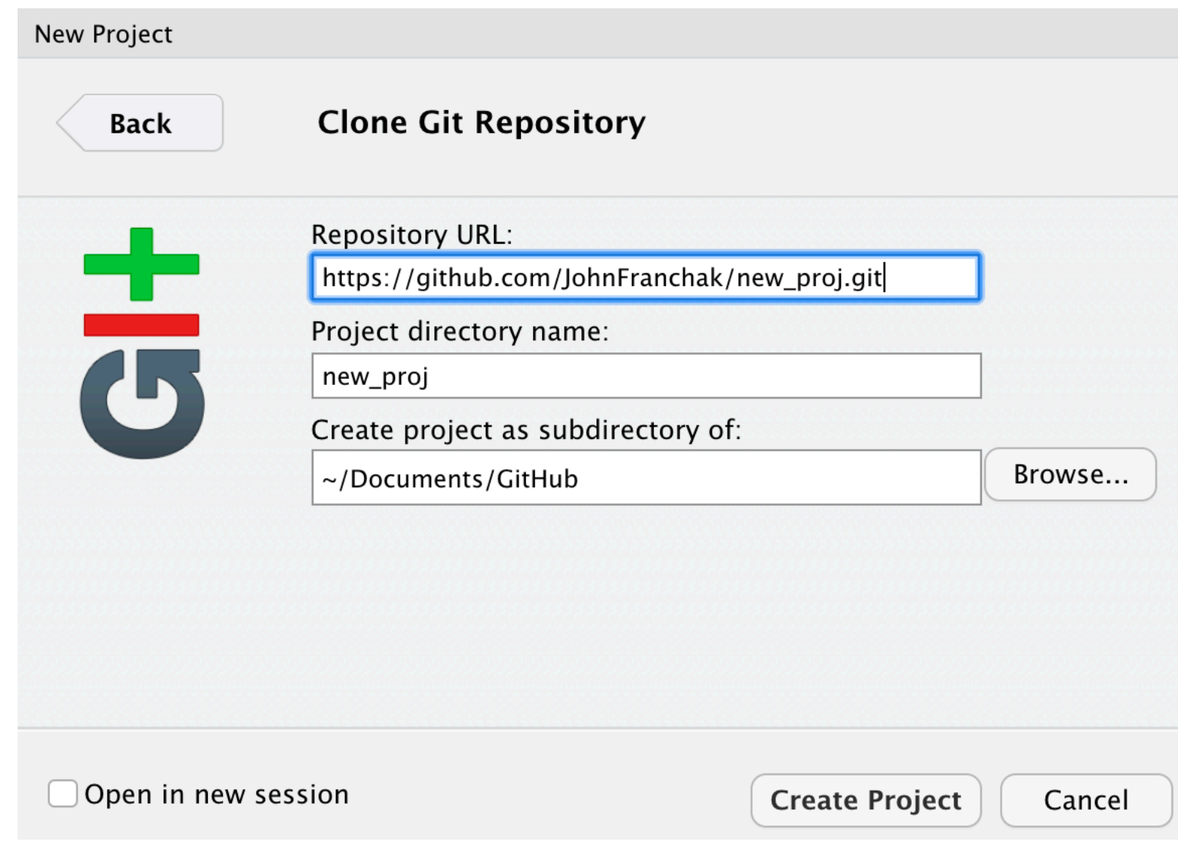
- Log in to github.com with your username
- Go to https://github.com/2025-PSYC-259/259-langbasics-importing
- Now it won't be PSYC-259-Data-Science's anymore, it will be yours!
- Let's get it off the Github website and onto your local computer by **cloning it**

Disclaimer!

- We are not going over how to work with the Github Desktop app here. We will just be working with GitHub online and using RStudio to connect to GitHub

Clone a project directly into RStudio from Github

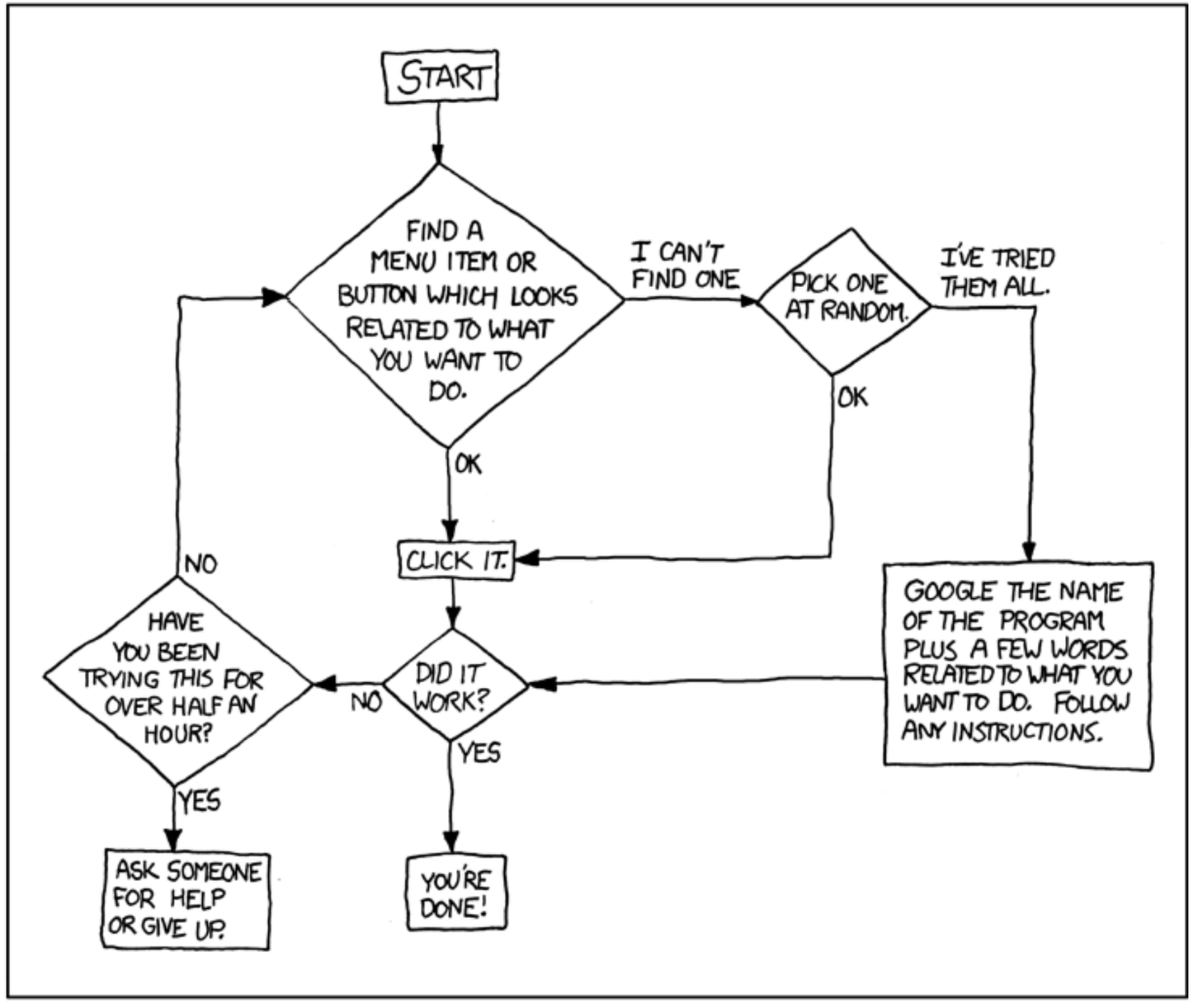




General advice:
How to get help

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE."

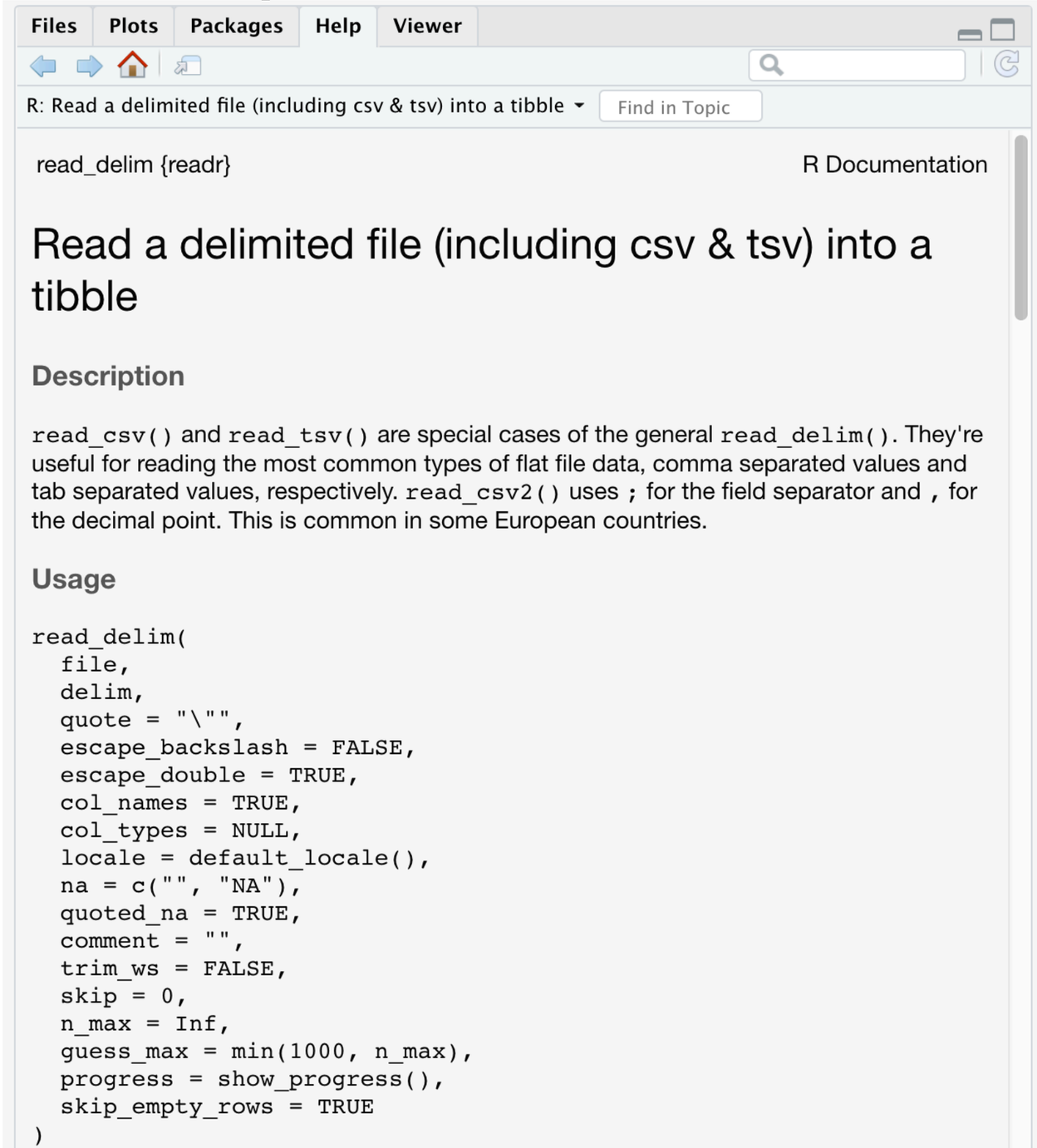
WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

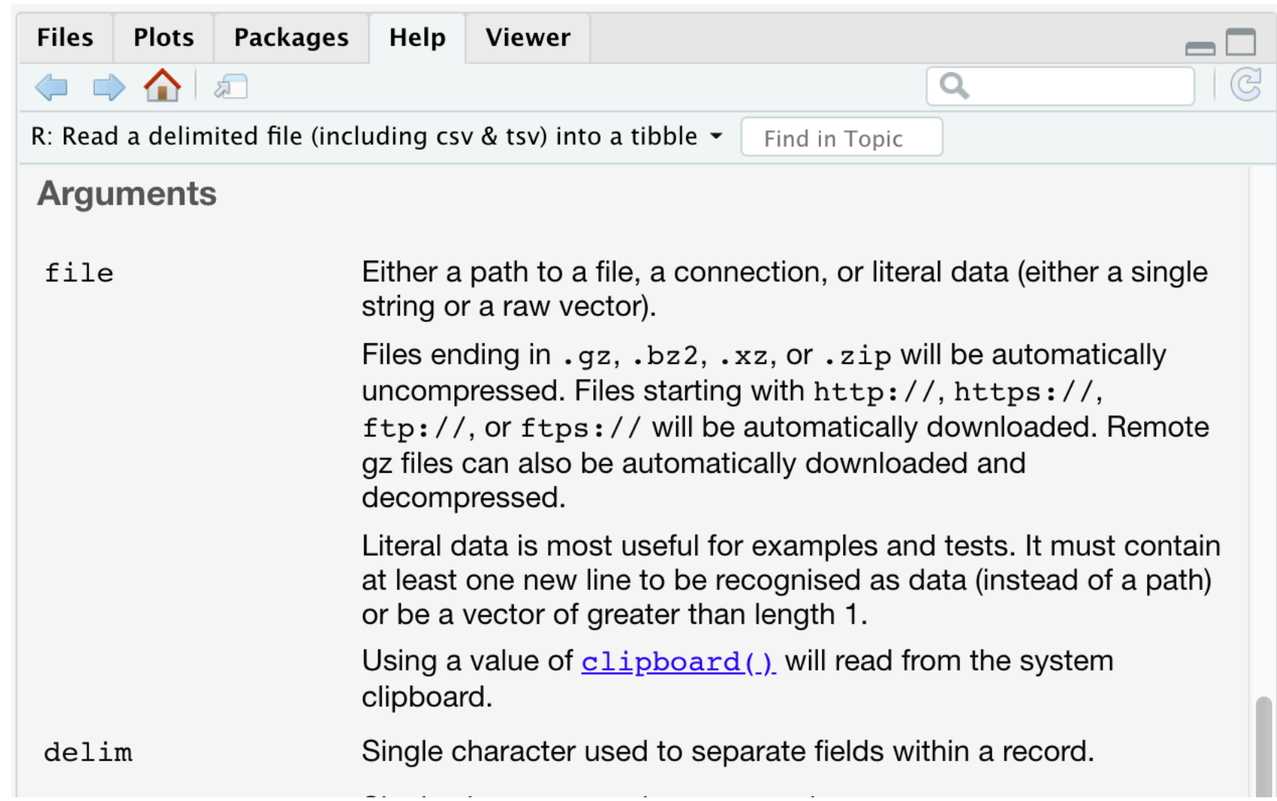
Where to search for help

- R Documentation
  - ?function brings up the documentation for a function (assuming the package is loaded)



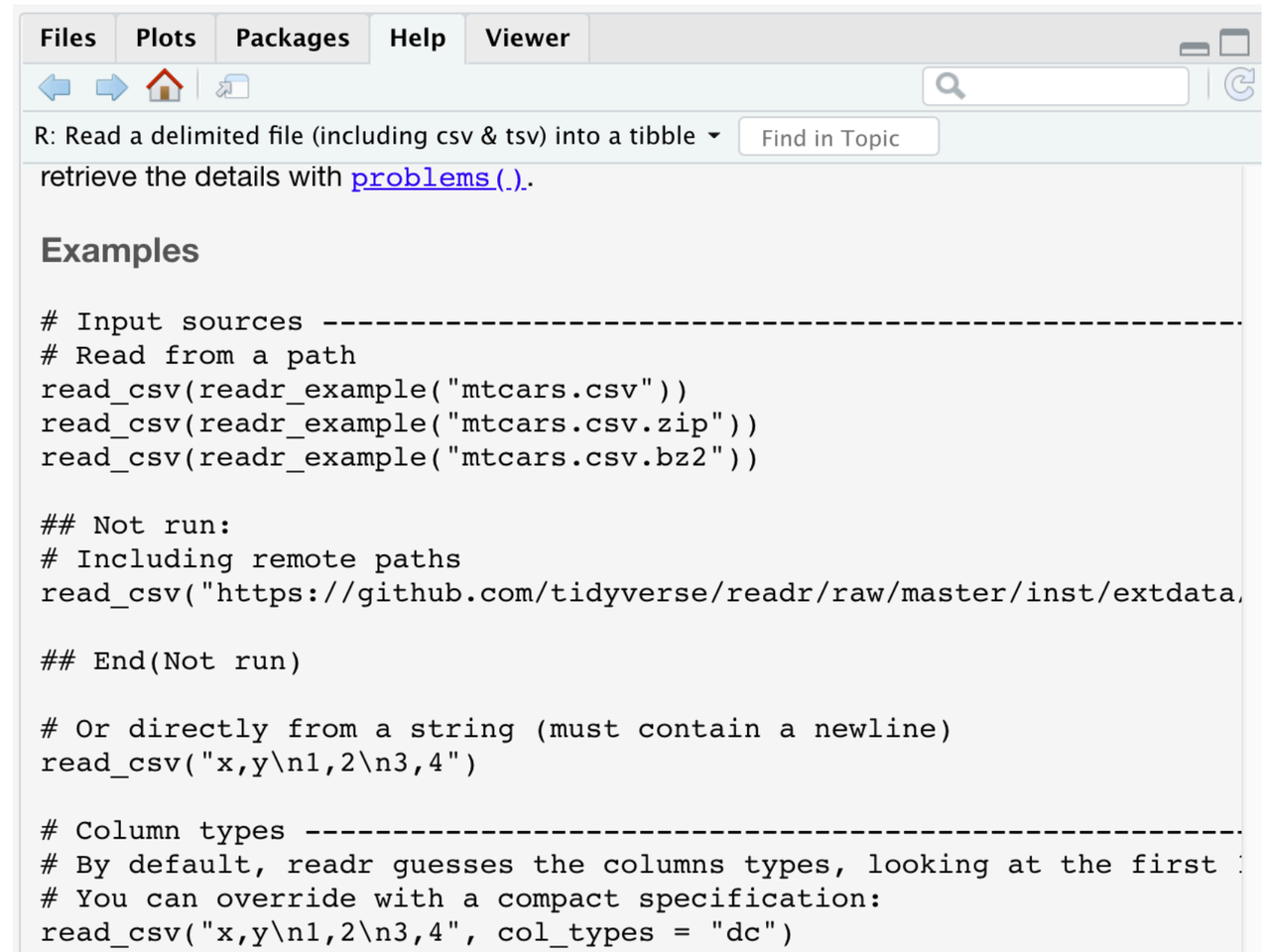
Where to search for help

- R Documentation
  - ?function brings up the documentation for a function (assuming the package is loaded)
  - Scroll down to see the "arguments" definitions



Where to search for help

- R Documentation
  - ?function brings up the documentation for a function (assuming the package is loaded)
  - Scroll down to see the "arguments" definitions
  - Scroll down more to see examples



Where to search for help

- Package vignettes/blogs
  - https://cran.r-project.org/web/packages/readr/vignettes/readr.html
- Google
  - Blogs/instructional sites
  - StackExchange (especially for error text)
  - Tidyverse community
    - https://community.rstudio.com/c/tidyverse/6
- Take working examples and fiddle!

Tidyverse cheatsheets!

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

String to file
write_file(x, path, append = FALSE**)**

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE**)**

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**

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
```

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")

```
a;b;c
1;2;3
4;5;NA
```

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

```
a|b|c
1|2|3
4|5|NA
```

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

```
a b c
1 2 3
4 5 NA
```

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5)**)**
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv"**)** Also **read_table().**
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

```
a,b,c
1,2,3
4,5,NA
```

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", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale()**)**

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive()**)**

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**

Read a file into a raw vector
read_file_raw(file**)**

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer

earn is a double (numeric)

sex is a character

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the default
- **col_character()**
- **col_double(), col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**), col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number(), col_numeric()**
- **col_skip()**

x <- read_csv("file.csv", col_types = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

x$A <- parse_number(x$A)

Tidyverse cheatsheets!

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

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

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	**:**, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	**-**, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

BREAK

After the break: Tutorial

Homework 1

Homework 1 - Data import (no Github yet)

- https://github.com/tspringstein/259-langbasics-importing-hw
- Fork this project
- Clone it to your RStudio
- Complete the homework (at least 4 completed questions counts as completed)
- Commit changes
- Push changes
- Invite Madison to be a collaborator on your project so she can provide comments