

Data I/O, Part 1

Data Wrangling in R

Explaining output on slides

In slides, a command (we'll also call them code or a code chunk) will look like this

```
print("I'm code")
```

```
[1] "I'm code"
```

And then directly after it, will be the output of the code.

So `print("I'm code")` is the code chunk and `[1] "I'm code"` is the output.

These slides were made in R using `knitr` and R Markdown (covered later today when we discuss reproducible research)

Data Input

- 'Reading in' data is the first step of any real project/analysis
- R can read almost any file format, especially via add-on packages
- We are going to focus on simple delimited files first
 - tab delimited (e.g. '.txt')
 - comma separated (e.g. '.csv')
 - Microsoft excel (e.g. '.xlsx')

Data Input

UFO Sightings via Kaggle.com: "Reports of unidentified flying object reports in the last century".

"There are two versions of this dataset: scrubbed and complete. The complete data includes entries where the location of the sighting was not found or blank (0.8146%) or have an erroneous or blank time (8.0237%). Since the reports date back to the 20th century, some older data might be obscured. Data contains city, state, time, description, and duration of each sighting."

<https://www.kaggle.com/NUFORC/ufo-sightings>

Data Input

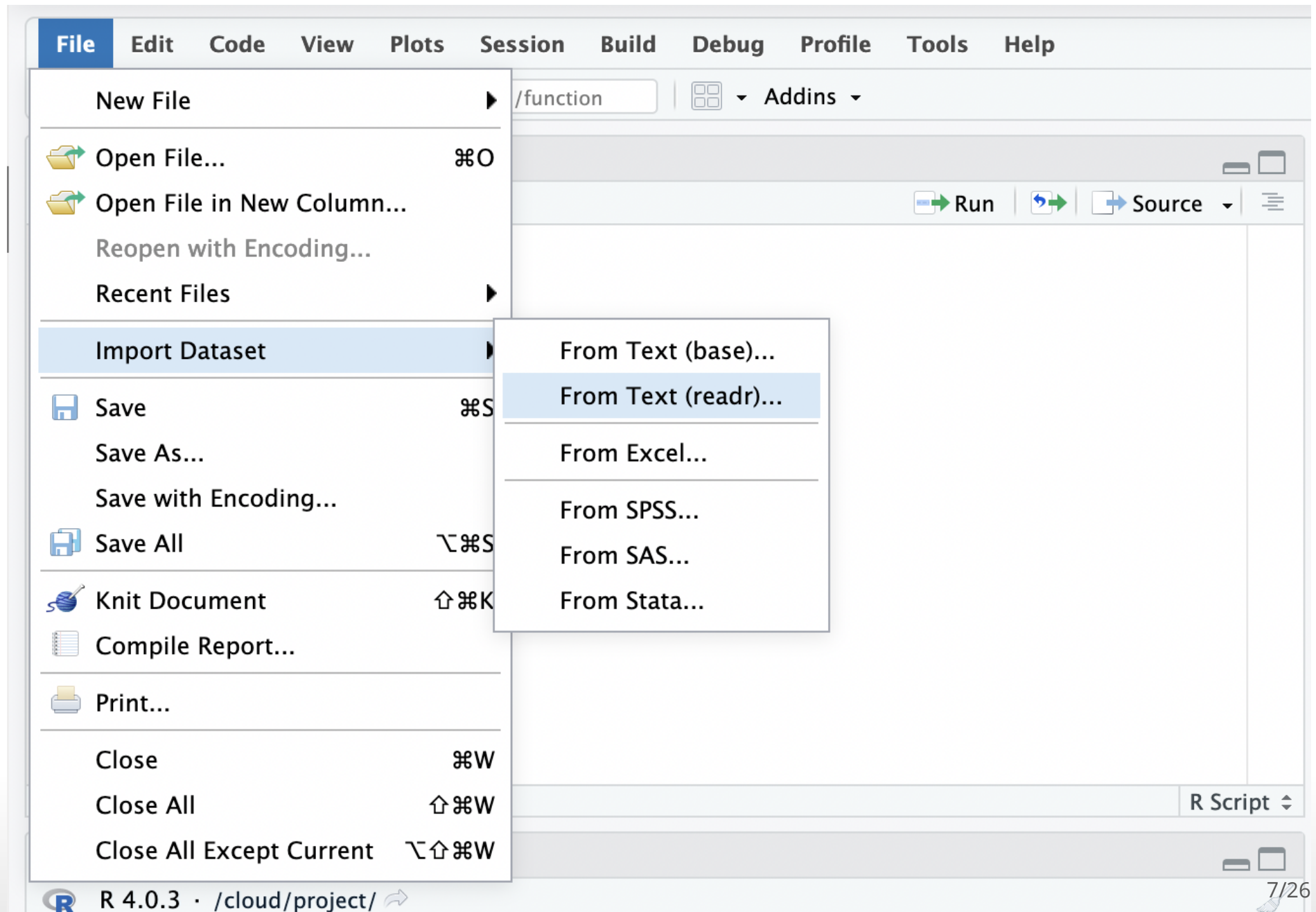
- Download data from http://sisbid.github.io/Module1/data/ufo/ufo_data_complete.csv.gz
- Upload the data to RStudio Cloud

Data Input

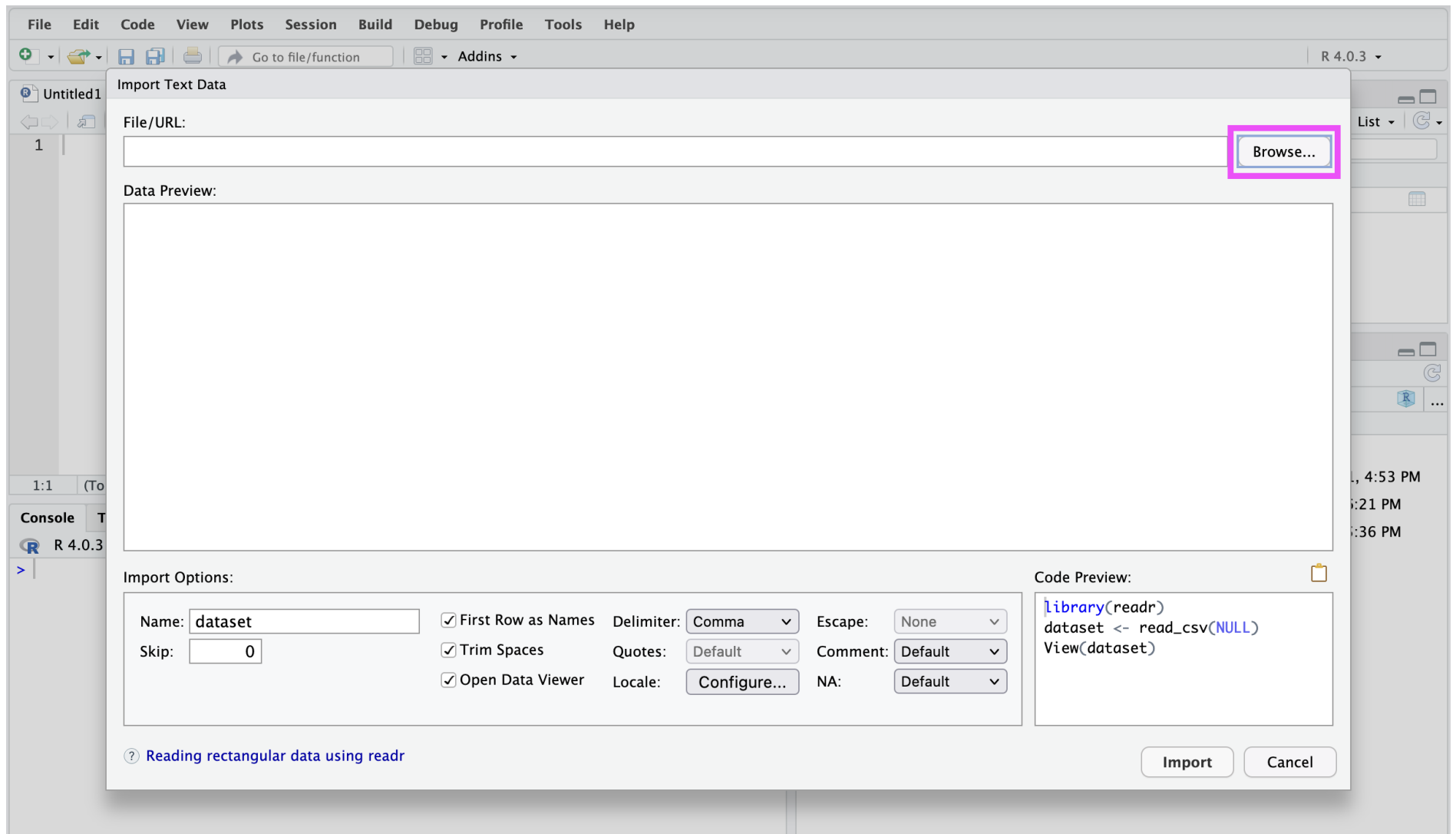
Easy way: import text datasets using the “File -> Import Dataset -> From Text (readr)” command. Selecting this will bring up a new screen that lets you specify the formatting of your text file.

Going through this process enters the corresponding R commands in the console (you can copy these for later!)

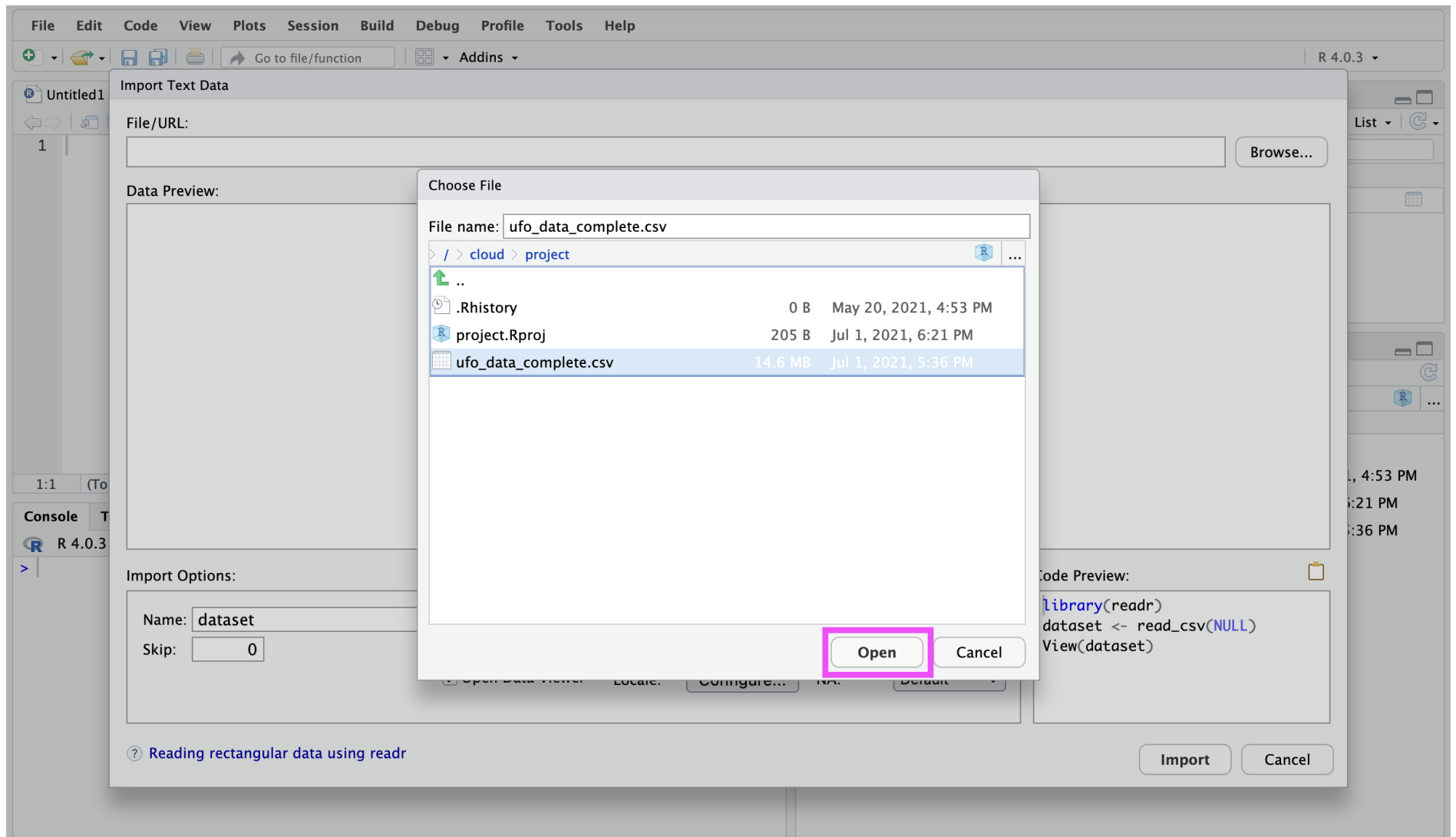
Data Input



Data Input



Data Input



Data Input

The screenshot shows the RStudio 'Import Text Data' dialog box. The 'File/URL' field contains '/cloud/project/ufo_data_complete.csv'. The 'Data Preview' section displays a table of UFO sightings. The 'Import Options' section includes checkboxes for 'First Row as Names', 'Trim Spaces', and 'Open Data Viewer', along with dropdowns for 'Delimiter', 'Escape', 'Quotes', 'Comment', and 'NA'. The 'Code Preview' section shows the R code for reading the CSV file. The 'Import' button is highlighted with a pink box.

File/URL: /cloud/project/ufo_data_complete.csv Browse...

Data Preview:

datetime (character)	city (character)	state (character)	country (character)	shape (character)	duration (seconds) (double)	duration (hours/min) (character)	comments
10/10/1949 20:30	san marcos	tx	us	cylinder	2700	45 minutes	This event took place in early f
10/10/1949 21:00	lackland afb	tx	NA	light	7200	1-2 hrs	1949 Lackland AFB, TX. L
10/10/1955 17:00	chester (uk/england)	NA	gb	circle	20	20 seconds	Green/Orange circular disc ove
10/10/1956 21:00	edna	tx	us	circle	20	1/2 hour	My older brother and twin siste
10/10/1960 20:00	kaneohe	hi	us	light	900	15 minutes	AS a Marine 1st Lt. flying an FJ-
10/10/1961 19:00	bristol	tn	us	sphere	300	5 minutes	My father is now 89 my brothe
10/10/1965 21:00	penarth (uk/wales)	NA	gb	circle	180	about 3 mins	penarth uk circle 3mins staye
10/10/1965 23:45	norwalk	ct	us	disk	1200	20 minutes	A bright orange color changing
10/10/1966 20:00	pell city	al	us	disk	180	3 minutes	Strobe Lighted disk shape obje

Previewing first 50 entries.

Import Options:

Name: ufo_data_complete ☒ First Row as Names Delimiter: Comma Escape: None

Skip: 0 ☒ Trim Spaces Quotes: Default Comment: Default

☒ Open Data Viewer Locale: Configure... NA: Default

Code Preview:

```
library(readr)
ufo_data_complete <- read_csv(
  "ufo_data_complete.csv")
View(ufo_data_complete)
```


Import Cancel

? Reading rectangular data using readr

Data Input

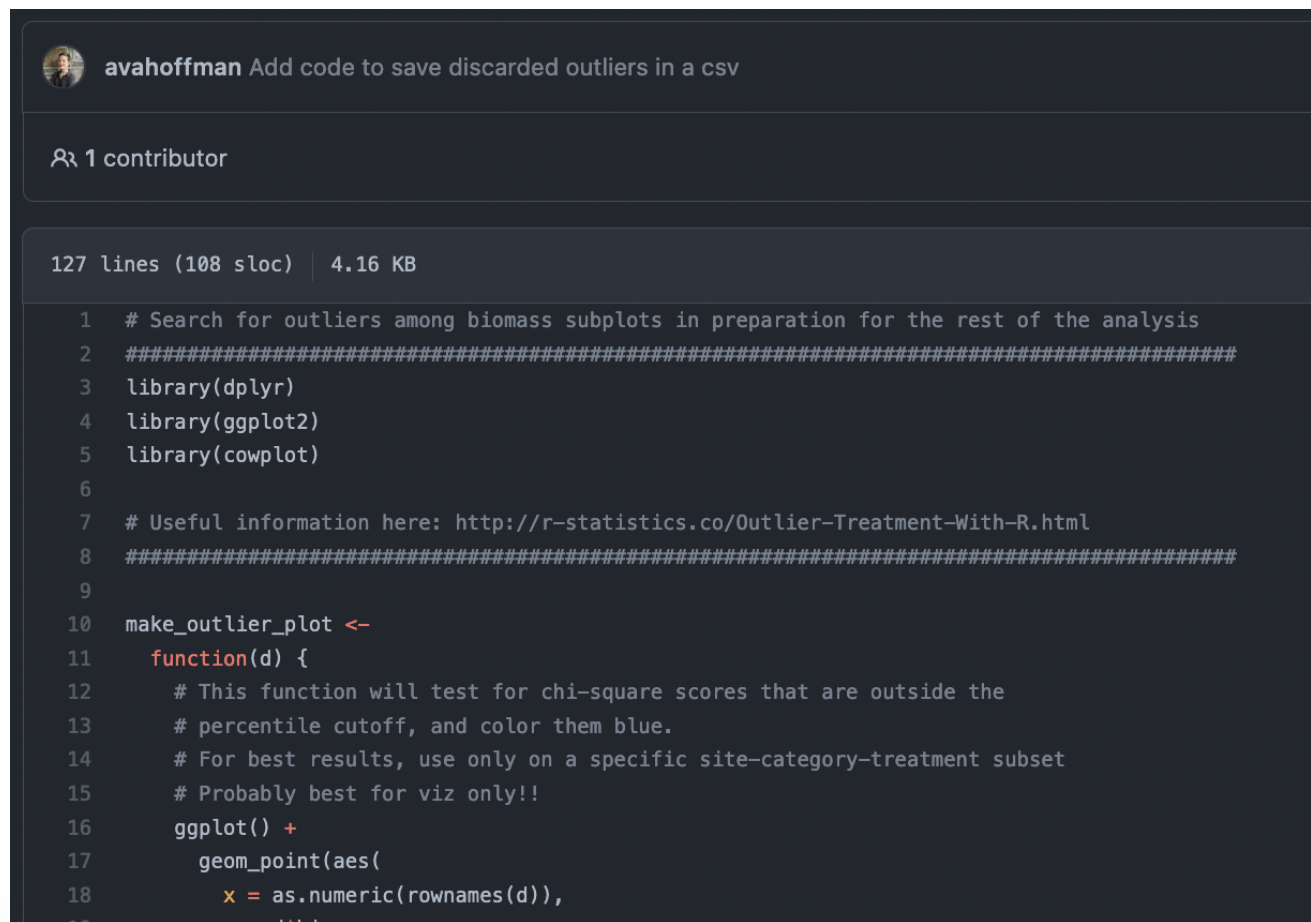
```
Console Terminal x Jobs x
R 4.0.3 · /cloud/project/
> library(readr)
> ufo_data_complete <- read_csv("ufo_data_complete.csv")

— Column specification —
cols(
  datetime = col_character(),
  city = col_character(),
  state = col_character(),
  country = col_character(),
  shape = col_character(),
  `duration (seconds)` = col_double(),
  `duration (hours/min)` = col_character(),
  comments = col_character(),
  `date posted` = col_character()
```



Commenting in Scripts

Commenting in code is super important. You should be able to go back to your code years after writing it and figure out exactly what the script is doing. Commenting helps you do this.



The screenshot shows a GitHub repository interface. At the top, the repository name is 'avahoffman' and the description is 'Add code to save discarded outliers in a csv'. Below this, it says '1 contributor'. The file statistics show '127 lines (108 sloc)' and '4.16 KB'. The code is displayed in a dark-themed editor with line numbers on the left. The code is an R script with many comments explaining its purpose and usage. The comments include instructions on how to use the function, where to find more information, and specific details about the data and plotting. The code itself defines a function 'make_outlier_plot' that takes a data frame 'd' as input and returns a ggplot object. The function uses 'dplyr' and 'ggplot2' libraries. It filters the data to a specific subset and then plots it using 'geom_point'.

```
1 # Search for outliers among biomass subplots in preparation for the rest of the analysis
2 #####
3 library(dplyr)
4 library(ggplot2)
5 library(cowplot)
6
7 # Useful information here: http://r-statistics.co/Outlier-Treatment-With-R.html
8 #####
9
10 make_outlier_plot <-
11   function(d) {
12     # This function will test for chi-square scores that are outside the
13     # percentile cutoff, and color them blue.
14     # For best results, use only on a specific site-category-treatment subset
15     # Probably best for viz only!!
16     ggplot() +
17       geom_point(aes(
18         x = as.numeric(rownames(d)),
19         y = d$biomass
```

Commenting in Scripts

Add a comment header to your script from today: # is the comment symbol

```
#####  
# Title: Demo R Script  
# Date: 7/13/2020  
# Purpose: Demonstrate comments in R  
#####  
  
# nothing to its right is evaluated  
x = 2 # but the left will be  
  
# this # is still a comment  
### you can use many #s as you want  
##### 5 #s or more make a collapsible section  
  
# sometimes if you have a really long comment,  
# you can take it to another line
```

R variables

- You can create variables from within the R environment and from files on your computer
- R uses "=" or "<-" to assign values to a variable name
- Variable names are case-sensitive, i.e. X and x are different

```
x = 2 # Same as: x <- 2  
x
```

```
[1] 2
```

```
x * 4
```

```
[1] 8
```

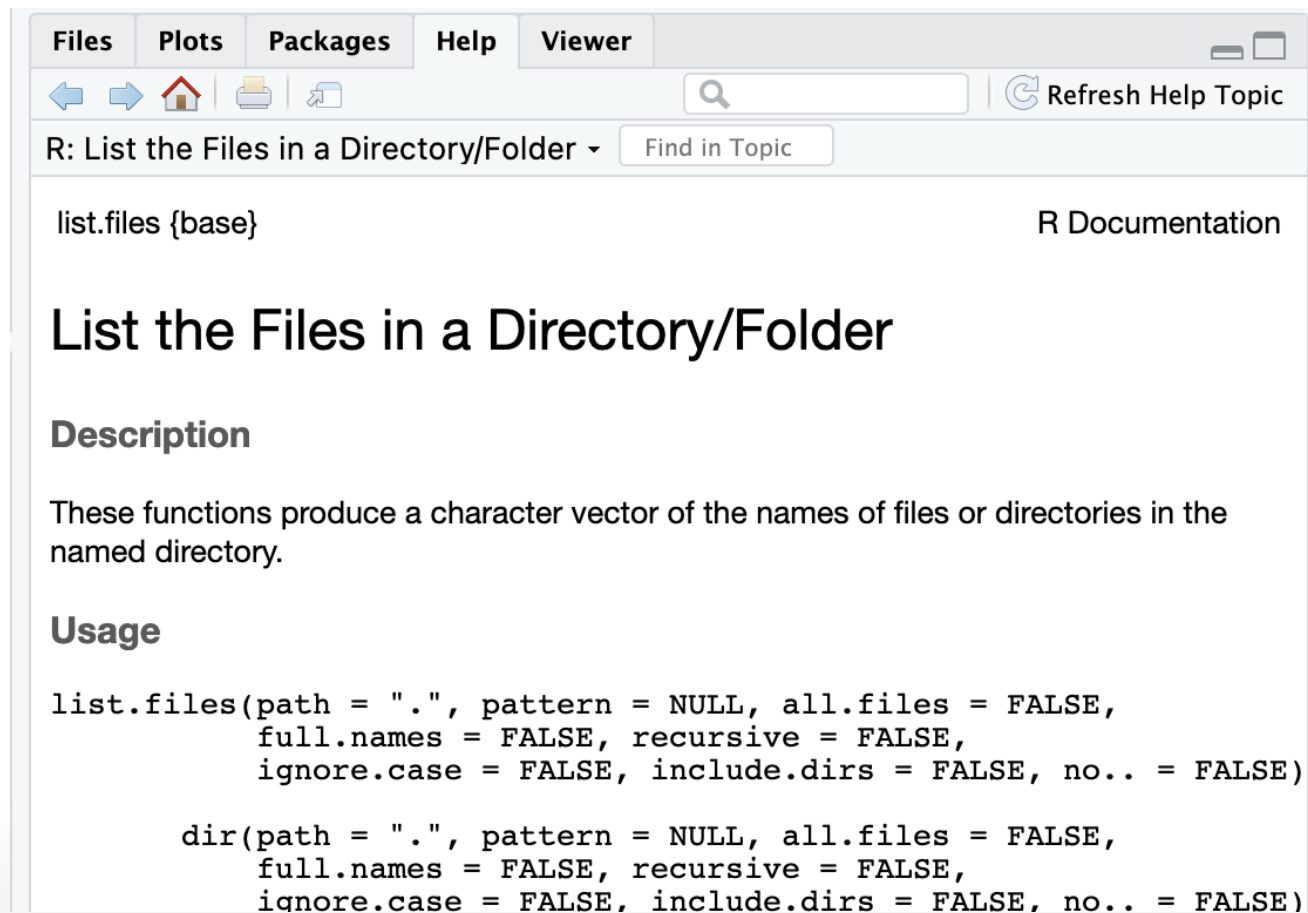
```
x + 2
```

```
[1] 4
```

Help

For any function, you can write `?FUNCTION_NAME`, or `help("FUNCTION_NAME")` to look at the help file:

```
?dir  
help("dir")
```



The screenshot shows the R Help Viewer window. The title bar includes tabs for Files, Plots, Packages, Help, and Viewer. The main content area displays the help page for the `list.files` function. The page title is "R: List the Files in a Directory/Folder". The description states: "These functions produce a character vector of the names of files or directories in the named directory." The usage section shows the following code:

```
list.files(path = ".", pattern = NULL, all.files = FALSE,  
           full.names = FALSE, recursive = FALSE,  
           ignore.case = FALSE, include.dirs = FALSE, no.. = FALSE)  
  
dir(path = ".", pattern = NULL, all.files = FALSE,  
    full.names = FALSE, recursive = FALSE,  
    ignore.case = FALSE, include.dirs = FALSE, no.. = FALSE)
```

Data Input

Initially-harder-but-gets-way-easier method: Utilizing functions in the `readr` package called `read_delim()` and `read_csv()` with code.

Data Input

So what is going on “behind the scenes”?

`read_delim()`: Read a delimited file into a data frame.

```
function (file, delim, quote = "\"", escape_backslash = FALSE,
  escape_double = TRUE, col_names = TRUE, col_types = NULL,
  locale = default_locale(), na = c("", "NA"), quoted_na = TRUE,
  comment = "", trim_ws = FALSE, skip = 0, n_max = Inf, guess_max = min(1000
    n_max), progress = show_progress(), skip_empty_rows = TRUE)
NULL
```

```
# example:
read_delim(file = "file.txt", delim = "\t")
read_delim("file.txt", "\t")
```

Data Input

- The filename is the path to your file, in quotes
- The function will look in your “working directory” if no absolute file path is given
- Note that the filename can also be a path to a file on a website (e.g. 'www.someurl.com/table1.txt')

Data Input

There is another convenient function for reading in CSV files, where the delimiter is assumed to be a comma:

```
ufo = read_csv("../data/ufo/ufo_data_complete.csv")
```

— Column specification —

```
cols(  
  datetime = col_character(),  
  city = col_character(),  
  state = col_character(),  
  country = col_character(),  
  shape = col_character(),  
  `duration (seconds)` = col_double(),  
  `duration (hours/min)` = col_character(),  
  comments = col_character(),  
  `date posted` = col_character(),  
  latitude = col_character(),  
  longitude = col_double()  
)
```

Warning: 199 parsing failures.

row	col	expected	actual	file
877	--	11 columns	12 columns	'../data/ufo/ufo_data_complete.csv'
1712	--	11 columns	12 columns	'../data/ufo/ufo_data_complete.csv'
1814	--	11 columns	12 columns	'../data/ufo/ufo_data_complete.csv'
2857	--	11 columns	12 columns	'../data/ufo/ufo_data_complete.csv'
3733	--	11 columns	12 columns	'../data/ufo/ufo_data_complete.csv'

Data Input

The `read_delim()` and related functions return a “tibble” is a `data.frame` with special printing, which is the primary data format for most data cleaning and analyses.

```
class(ufo)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Data Input

ufo

```
# A tibble: 88,875 x 11
  datetime city state country shape `duration (seco...`duration (hour... commen
  <chr>      <chr> <chr> <chr>    <chr>      <dbl> <chr>          <chr>
1 10/10/1... san ... tx     us     cyli...    2700 45 minutes    This e
2 10/10/1... lack... tx     <NA>    light    7200 1-2 hrs      1949 I
3 10/10/1... ches... <NA>    gb     circ...     20 20 seconds    Green/
4 10/10/1... edna  tx     us     circ...     20 1/2 hour      My old
5 10/10/1... kane... hi     us     light    900 15 minutes    AS a M
6 10/10/1... bris... tn     us     sphe...    300 5 minutes     My fat
7 10/10/1... pena... <NA>    gb     circ...    180 about 3 mins    penart
8 10/10/1... norw... ct     us     disk     1200 20 minutes    A brig
9 10/10/1... pell... al     us     disk     180 3 minutes     Strobe
10 10/10/1... live... fl     us     disk     120 several minutes    Saucer
# ... with 88,865 more rows, and 3 more variables: date posted <chr>,
# latitude <chr>, longitude <dbl>
```

Data Input

There are also data importing functions provided in base R (rather than the `readr` package), like `read.delim` and `read.csv`.

These functions have slightly different syntax for reading in data, like `header` and `as.is`.

However, while many online resources use the base R tools, recent versions of RStudio switched to use these new `readr` data import tools, so we will use them in the class for slides. They are also up to two times faster for reading in large datasets, and have a progress bar which is nice.

Data Input - Excel

Many data analysts collaborate with researchers who use Excel to enter and curate their data. Often times, this is the input data for an analysis. You therefore have two options for getting this data into R:

- Saving the Excel sheet as a .csv file, and using `read_csv()`
- Using an add-on package, like `readxl`, the `read_excel` function

For single worksheet .xlsx files, better to keep the XLSX file as the raw data as Excel exporting CSV can make slight changes (but I often have to strip off additional summary data from the columns).

Data Input - Other Software

- `haven` package (<https://cran.r-project.org/web/packages/haven/index.html>) reads in SAS, SPSS, Stata formats
- `sas7bdat` reads `.sas7bdat` files
- `sf` & `rgdal` read in spatial data
- `pdftools` reads `.pdf`, but things can get tricky quickly

Common new user mistakes we have seen

1. **Working directory problems: trying to read files that R “can’t find”**
 - RStudio can help, and so do RStudio Projects
2. Lack of comments in code
3. Typos (R is **case sensitive**, x and x are different)
 - RStudio helps with “tab completion”
4. Data type problems (is that a string or a number?)
5. Open ended quotes, parentheses, and brackets

Working Directories

- R “looks” for files on your computer (or cloud) relative to the “working” directory
- Best practice is to double-click an R file to start a new RStudio session, instead of launching RStudio from Desktop or somewhere else.
 - it sets the working directory to match that file
 - RStudio Cloud can simplify things
- If you do set a working directory, do it at the beginning of your script.
- Example of getting and setting the working directory:

```
## get the working directory  
getwd()  
setwd("~/Lectures")
```