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

R variables

A few reminders: * You can create variables from within the R environment and from files on your computer * Use “<-” to assign values to a variable name * Variable names are case-sensitive, i.e. X and x are different

```
x <- 2
x
```

```
[1] 2
```

```
x * 4
```

```
[1] 8
```

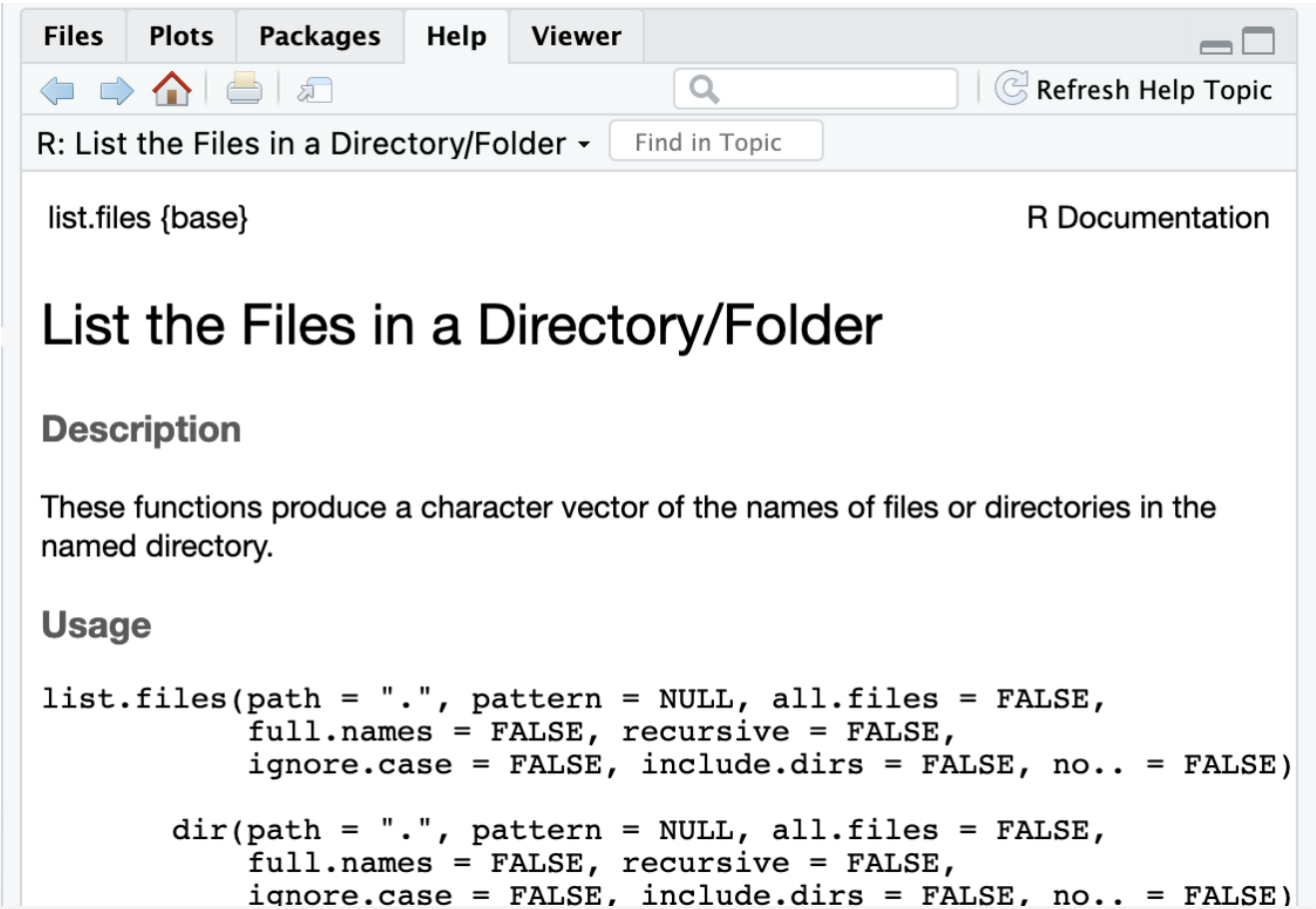
```
x + 2
```

```
[1] 4
```

Help

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

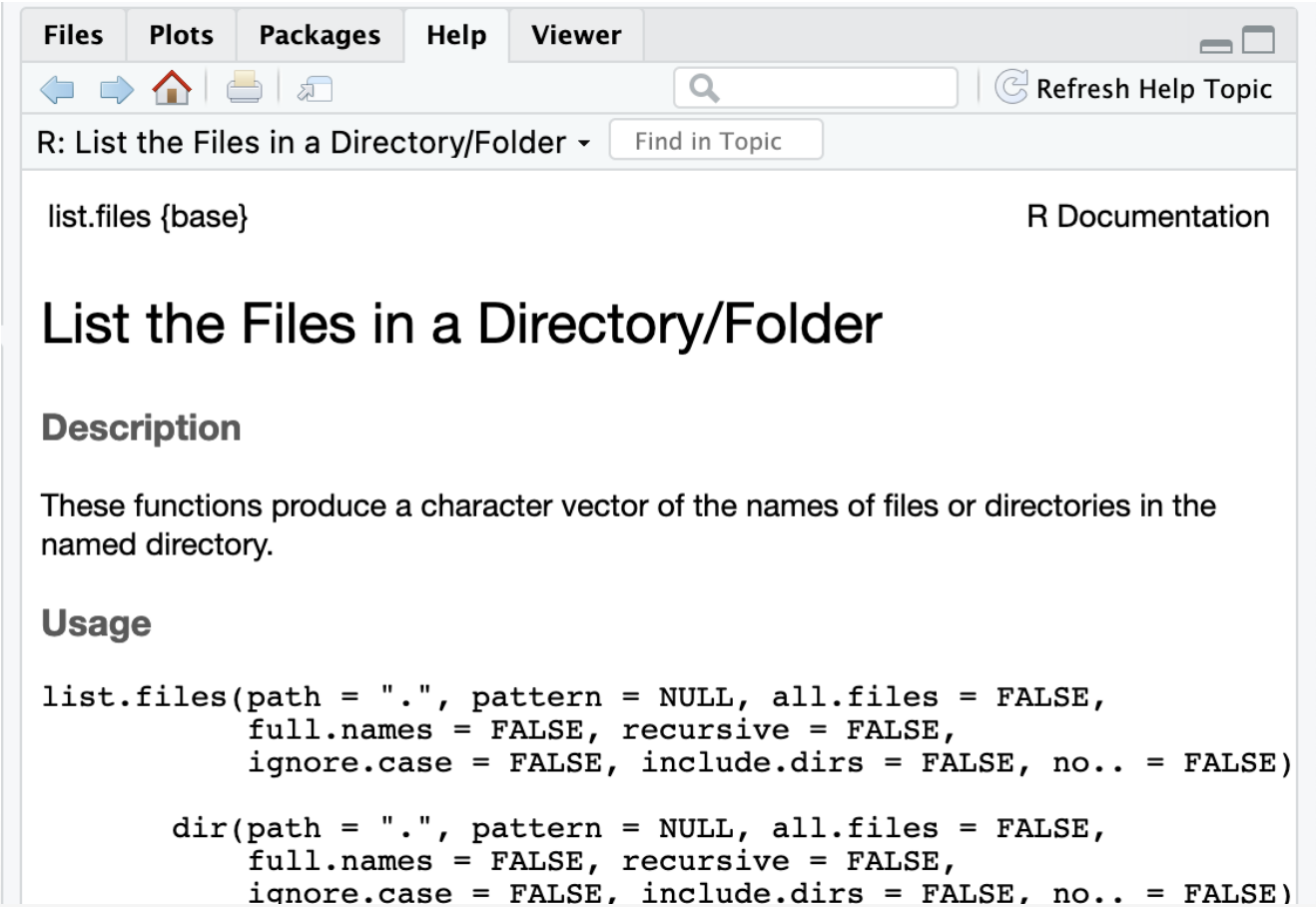
```
?dir
help("dir")
```



Packages

Not all packages are available on AnVIL's RStudio.

```
install.packages("naniar")
AnVIL::install("naniar") # Alternative
library(naniar)
```



Data Input

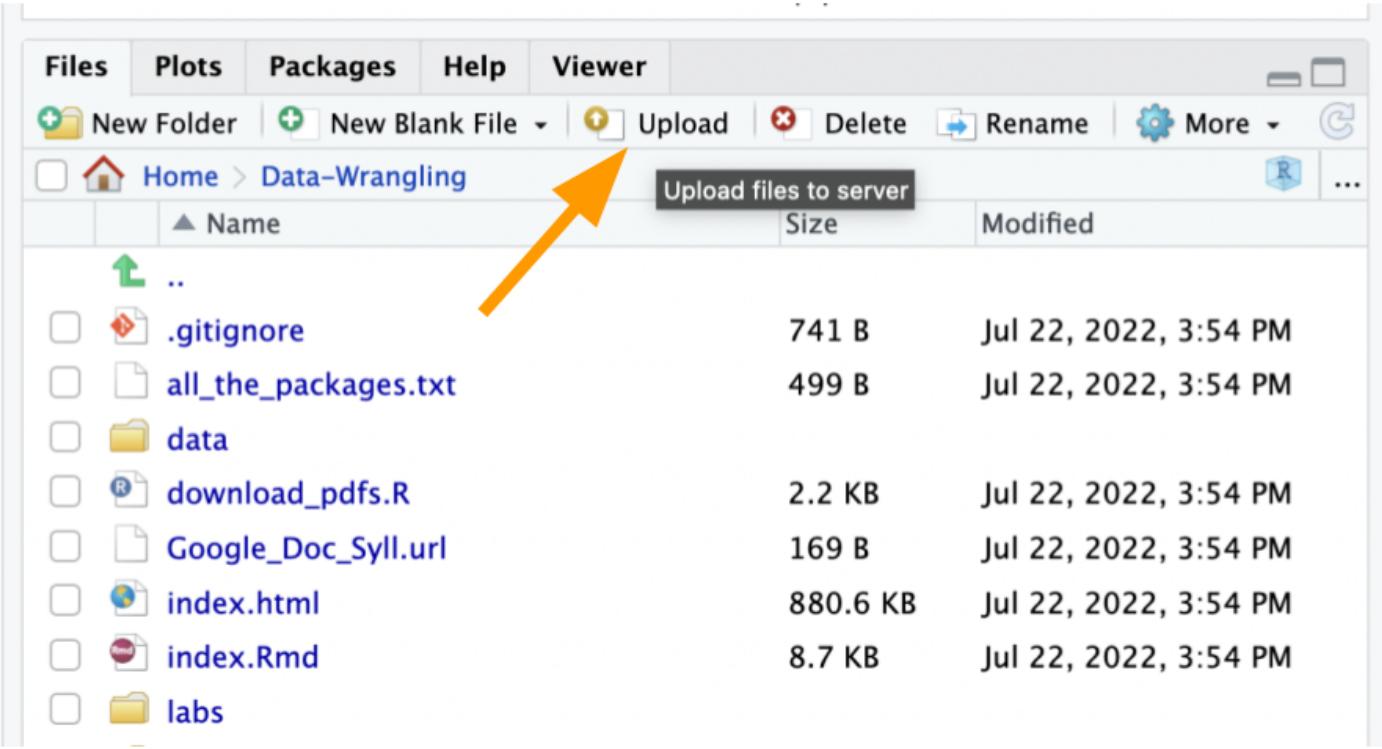
The Easy way: import text datasets using the “File –> Import Dataset –> From Text (readr)” command. Selecting this will bring up a new screen that lets you specify the formatting of your text file.

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

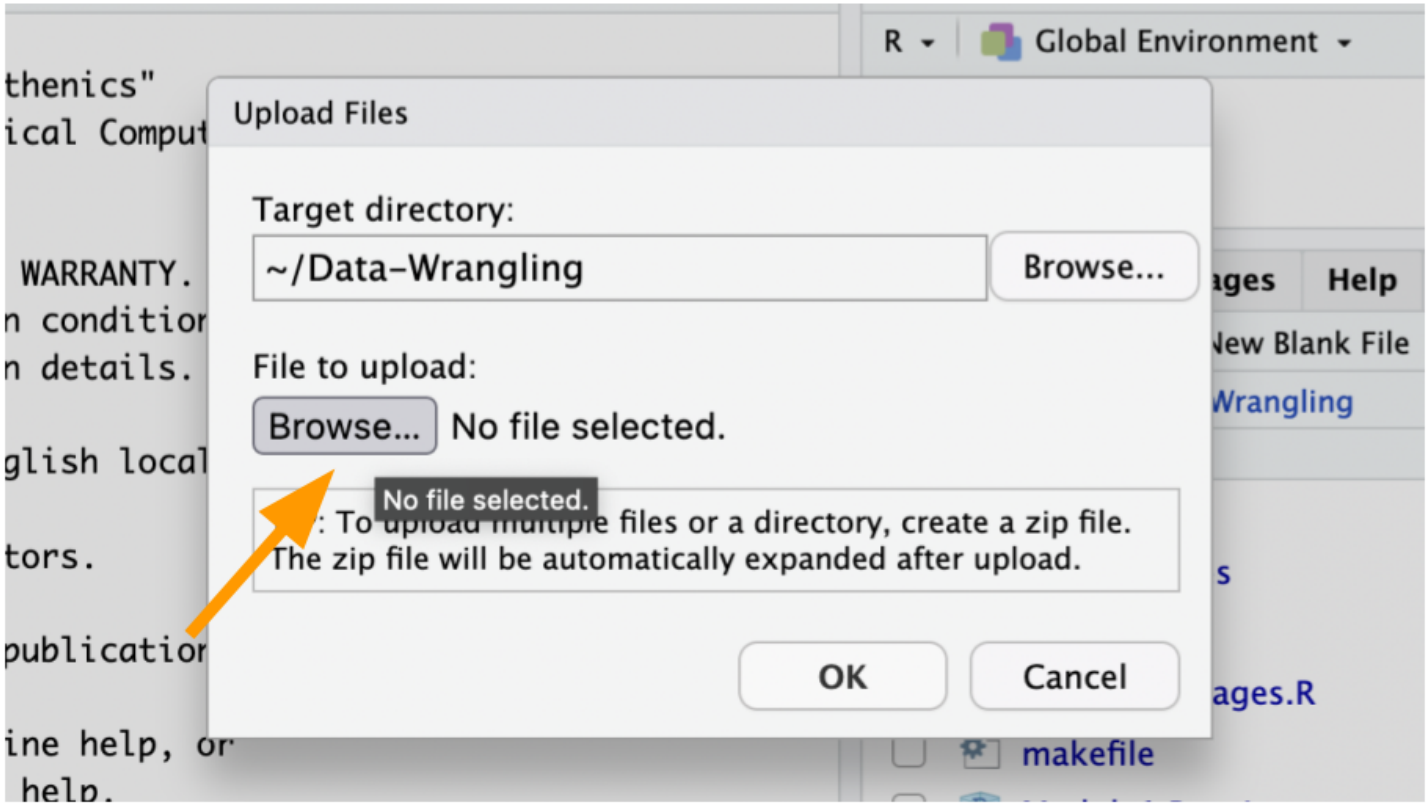
Approach #1

- Download data onto your machine from http://sisbid.github.io/Data-Wrangling/data/ufo/ufo_data_complete.csv.gz (right-click this link)
- Upload the data to AnVIL
- Read into RStudio's environment

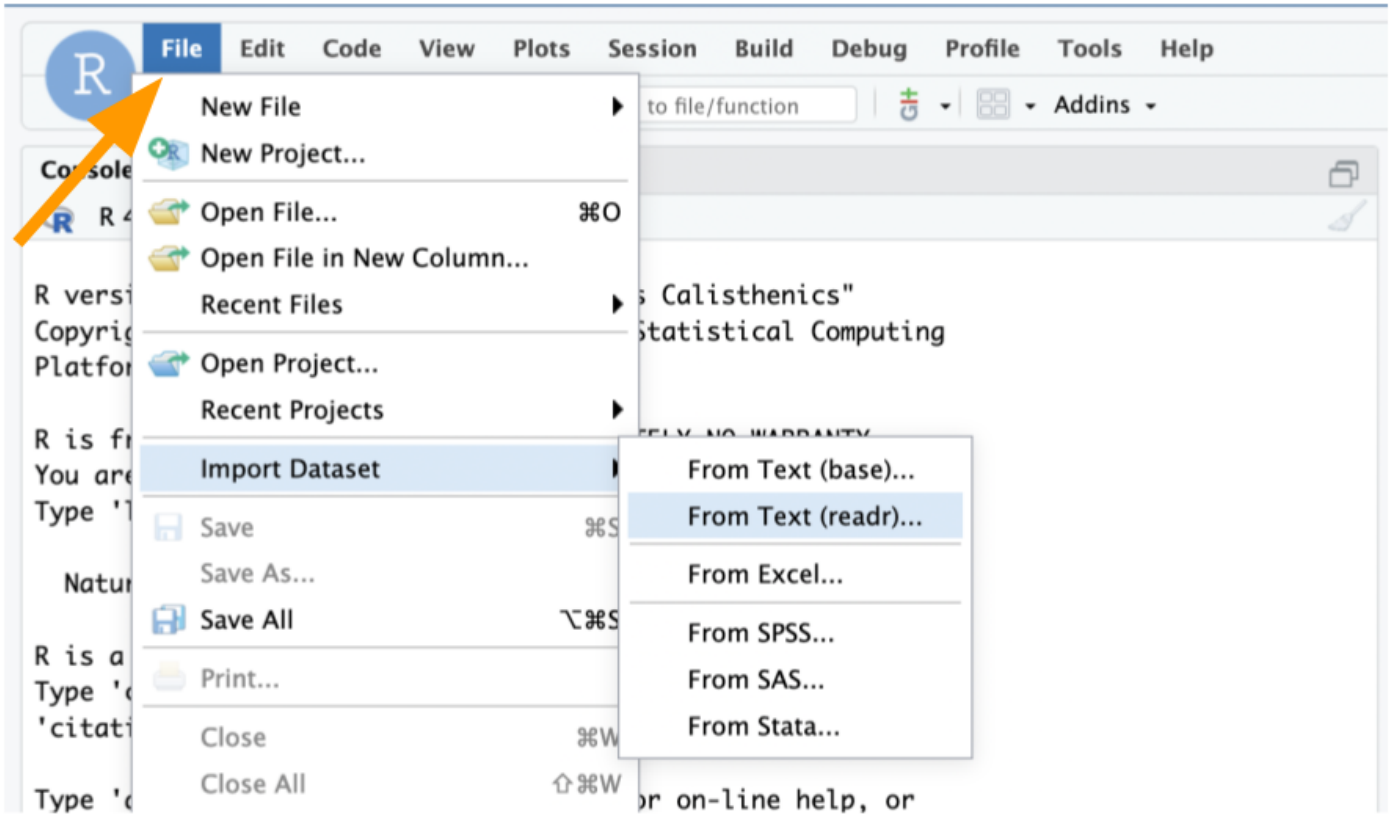
Upload the data to AnVIL



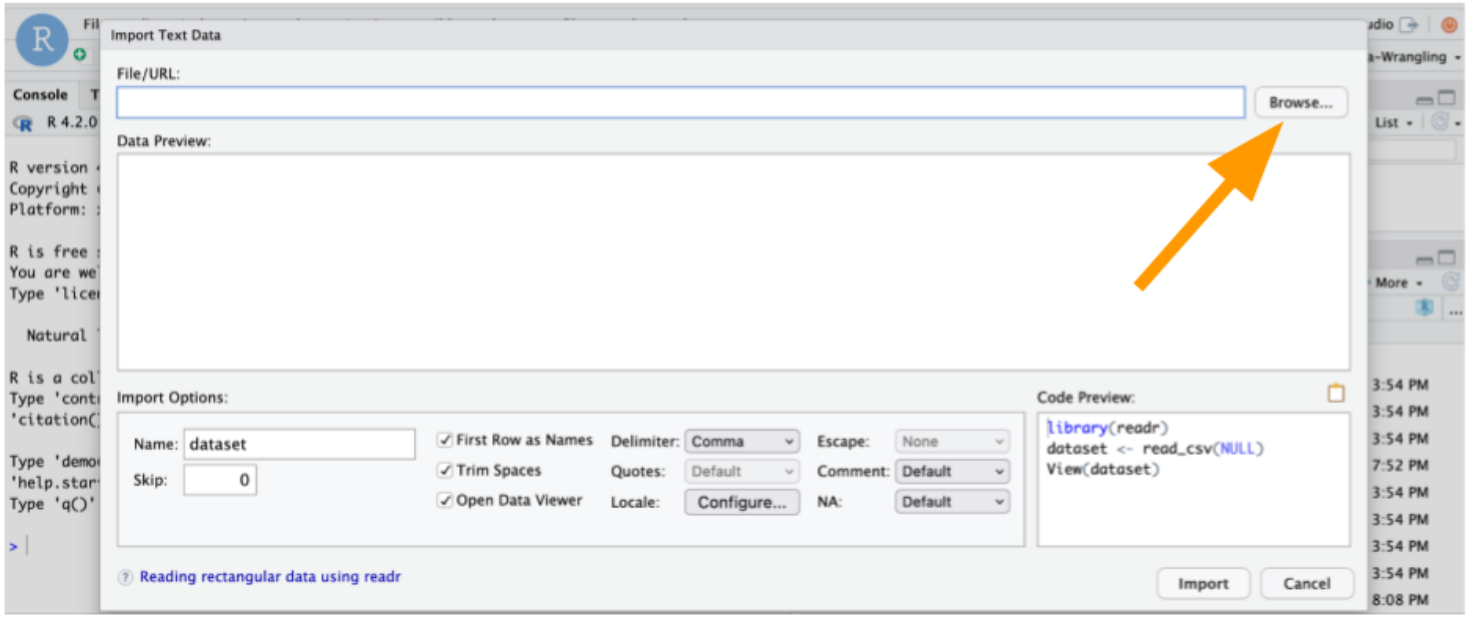
Upload the data to AnVIL



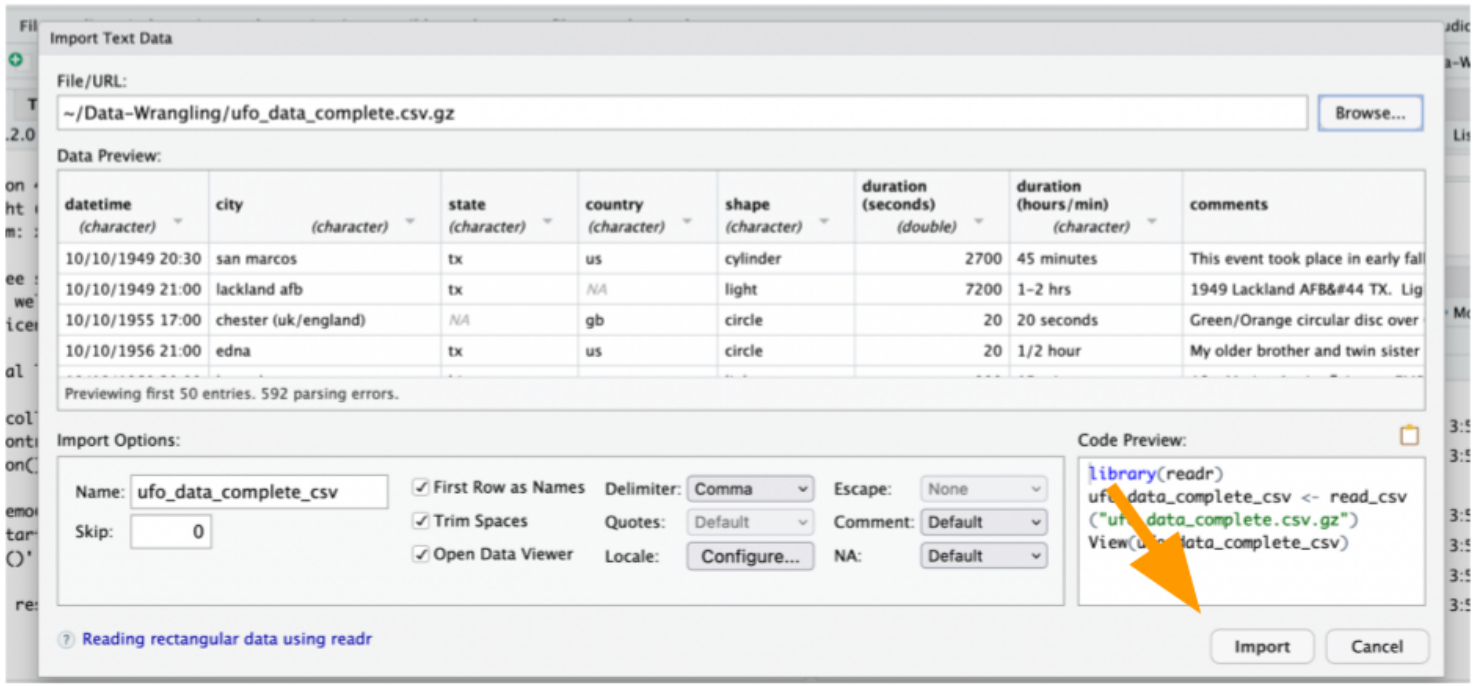
Read into RStudio's environment



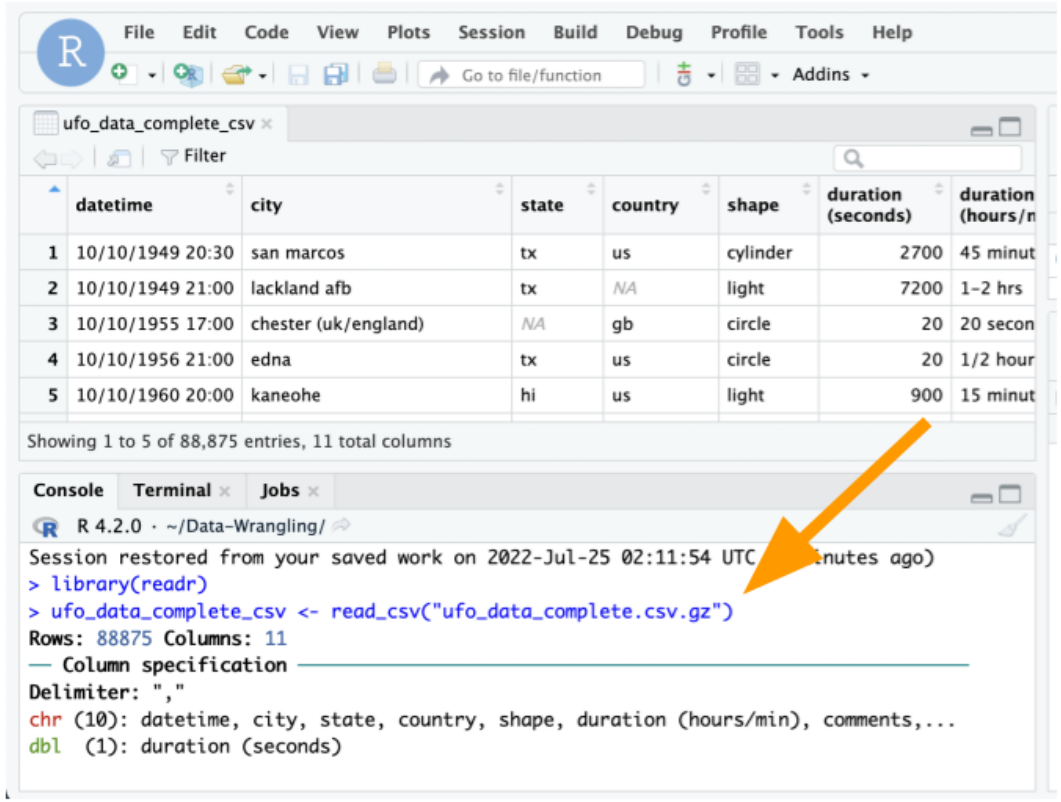
Read into RStudio's environment



Read into RStudio's environment



Check out the commands!



Approach #2

- Read into RStudio's environment directly

```
ufo_data_complete_csv <-
  read_csv("http://sisbid.github.io/Data-Wrangling/data/ufo/ufo_data_complete.csv.gz")
```

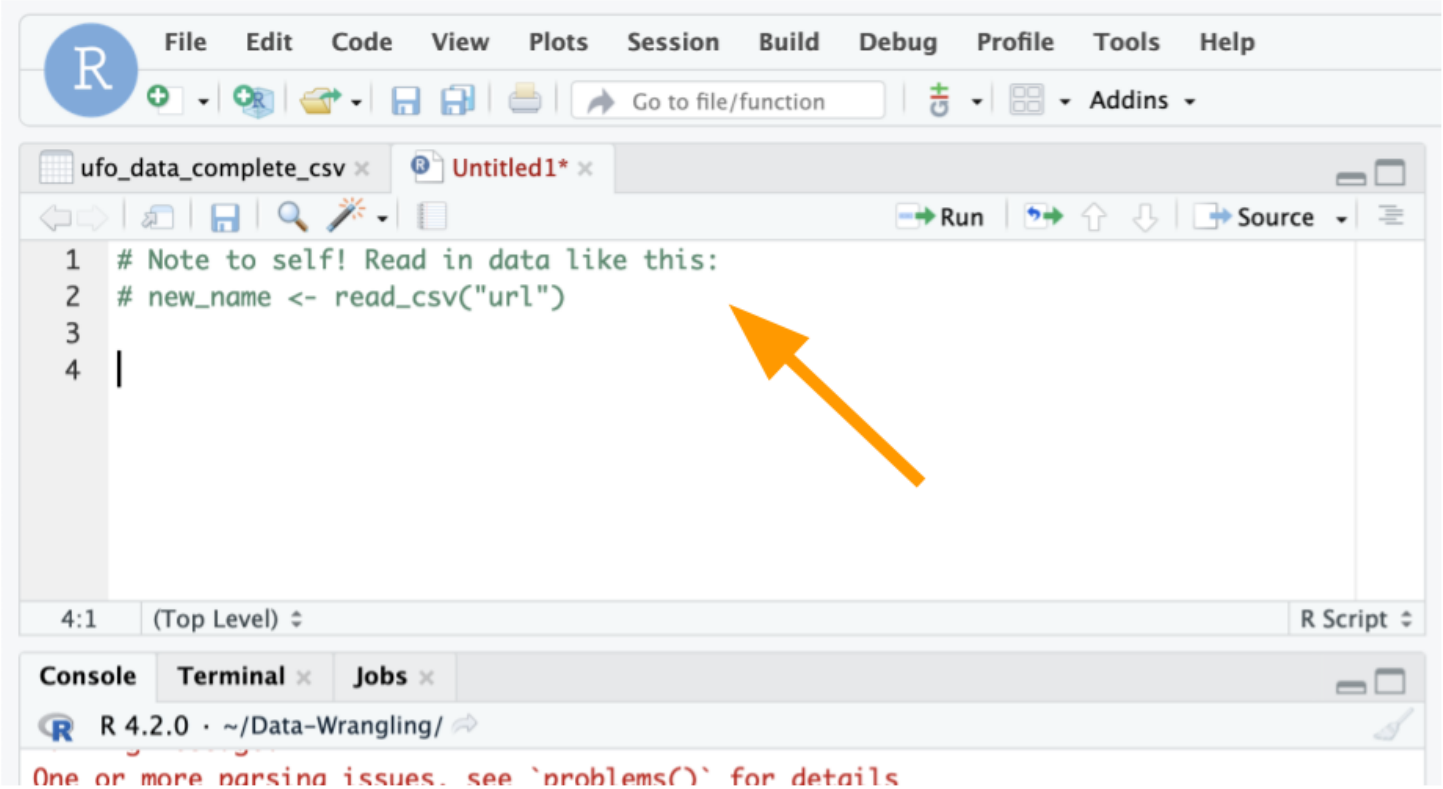
```
Warning: One or more parsing issues, see `problems()` for details

Rows: 88875 Columns: 11
── Column specification ────────────────────────────────────────────────────────────────────────────────────────────────────
Delimiter: ","
chr (10): datetime, city, state, country, shape, duration (hours/min), comments, date posted, latitude, longitude
dbl  (1): duration (seconds)

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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. Also handy for notes!



Commenting in Scripts

avahoffman Add code to save discarded outliers in a csv

1 contributor

127 lines (108 sloc) | 4.16 KB

```r
# Search for outliers among biomass subplots in preparation for the rest of the analysis
##########################################################################################
library(dplyr)
library(ggplot2)
library(cowplot)

# Useful information here: http://r-statistics.co/Outlier-Treatment-With-R.html
##########################################################################################

make_outlier_plot <-
  function(d) {
    # This function will test for chi-square scores that are outside the
    # percentile cutoff, and color them blue.
    # For best results, use only on a specific site-category-treatment subset
    # Probably best for viz only!!
    ggplot() +
      geom_point(aes(
        x = as.numeric(rownames(d)),
```

Data Input: `readr`

`read_delim()` and `read_csv()` from the `readr` package

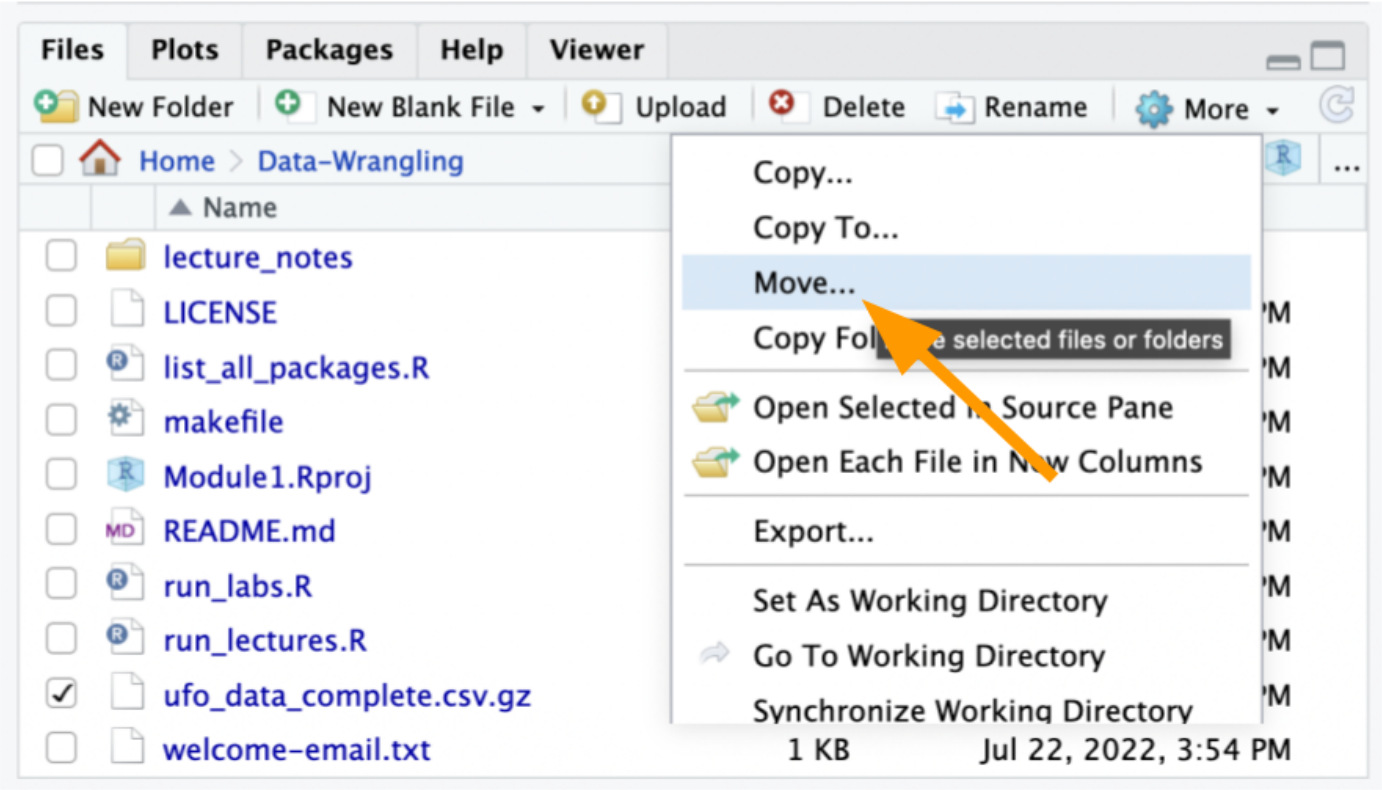
```
# example:
read_delim(file = "file.txt", delim = "\t")
read_csv("file.csv")
```

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

What if your file is in the “Home” directory?



Data Input

Backtrack using the relative path with `../` like:

```
ufo <- read_csv("../ufo_data_complete.csv.gz")
```

Data Input

Or, read in from a subfolder:

```
ufo <- read_csv("data/ufo/ufo_data_complete.csv")
```

```
Warning: One or more parsing issues, see `problems()` for details

Rows: 88875 Columns: 11
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (10): datetime, city, state, country, shape, duration (hours/min), comment
dbl  (1): duration (seconds)

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this messa
```

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 × 11
   datetime         city  state country shape `duration (sec…` `duration (hou…
   <chr>            <chr> <chr> <chr>   <chr>             <dbl> <chr>
 1 10/10/1949 20:30 san … tx    us      cyli…              2700 45 minutes
 2 10/10/1949 21:00 lack… tx    <NA>    light              7200 1-2 hrs
 3 10/10/1955 17:00 ches… <NA>  gb      circ…                20 20 seconds
 4 10/10/1956 21:00 edna  tx    us      circ…                20 1/2 hour
 5 10/10/1960 20:00 kane… hi    us      light               900 15 minutes
 6 10/10/1961 19:00 bris… tn    us      sphe…               300 5 minutes
 7 10/10/1965 21:00 pena… <NA>  gb      circ…               180 about 3 mins
 8 10/10/1965 23:45 norw… ct    us      disk               1200 20 minutes
 9 10/10/1966 20:00 pell… al    us      disk               180 3  minutes
10 10/10/1966 21:00 live… fl    us      disk               120 several minutes
# … with 88,865 more rows
```

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: `readr` highlights

- Modern, improved tools from `readr` R package: `read_delim()`, `read_csv()`
  - needs a file path to be provided
  - parses the file into rows/columns, determines column type
  - returns a data frame
- Some functions to look at a data frame:
  - `head()` shows first few rows
  - `tail()` shows the last few rows
  - `View()` shows the data as a spreadsheet
  - `spec()` gives specification of column types

Data input: other file types

- From `readr` package:
  - `read_delim()`: general delimited files
  - `read_csv()`: comma separated (CSV) files
  - `read_tsv()`: tab separated files
  - others
- For reading Excel files, you can do one of:
  - use `read_excel()` function from `readxl` package
  - use other packages: `xlsx`, `openxlsx`

Data input: other file types

- `haven` package has functions to read SAS, SPSS, Stata formats

```
library(haven)

# SAS
read_sas(file = "mtcars.sas7bdat")

# SPSS
read_sav(file = "mtcars.sav")

# Stata
read_dta(file = "mtcars.dta")
```