Data I/O + Structure

Data Wrangling in R

Data Input

Outline

- Part 0: A little bit of set up!
- Part 1: reading in manually (point and click)
- Part 2: reading in directly & working directories
- Part 3: checking data & multiple file formats

Data Input: `readr`

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

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

```r
# comma delimited:
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’)

Example

https://sisbid.github.io/Data-Wrangling/data/ufo/ufo_data_complete.csv

```
# From URL
ufo <- read_csv(
  "https://sisbid.github.io/Data-Wrangling/data/ufo/ufo_data_complete.csv"
)

# From your 'data-wrangling' directory
ufo <- read_csv("ufo_data_complete.csv")
```

(`Warning message: One or more parsing issues, call 'problems()'` – more on this later)

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" 
```

Check to make sure you see the new object in the Environment pane.

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 here. They are also up to two times faster for reading in large datasets, and have a progress bar which is nice.

Data Input: **readr**

`read_table()` from the `readr` package, allows any number of whitespace characters between columns, and the lines can be of different lengths.

```
# example for whitespace delimited :
read_table(file = "file.txt")
```

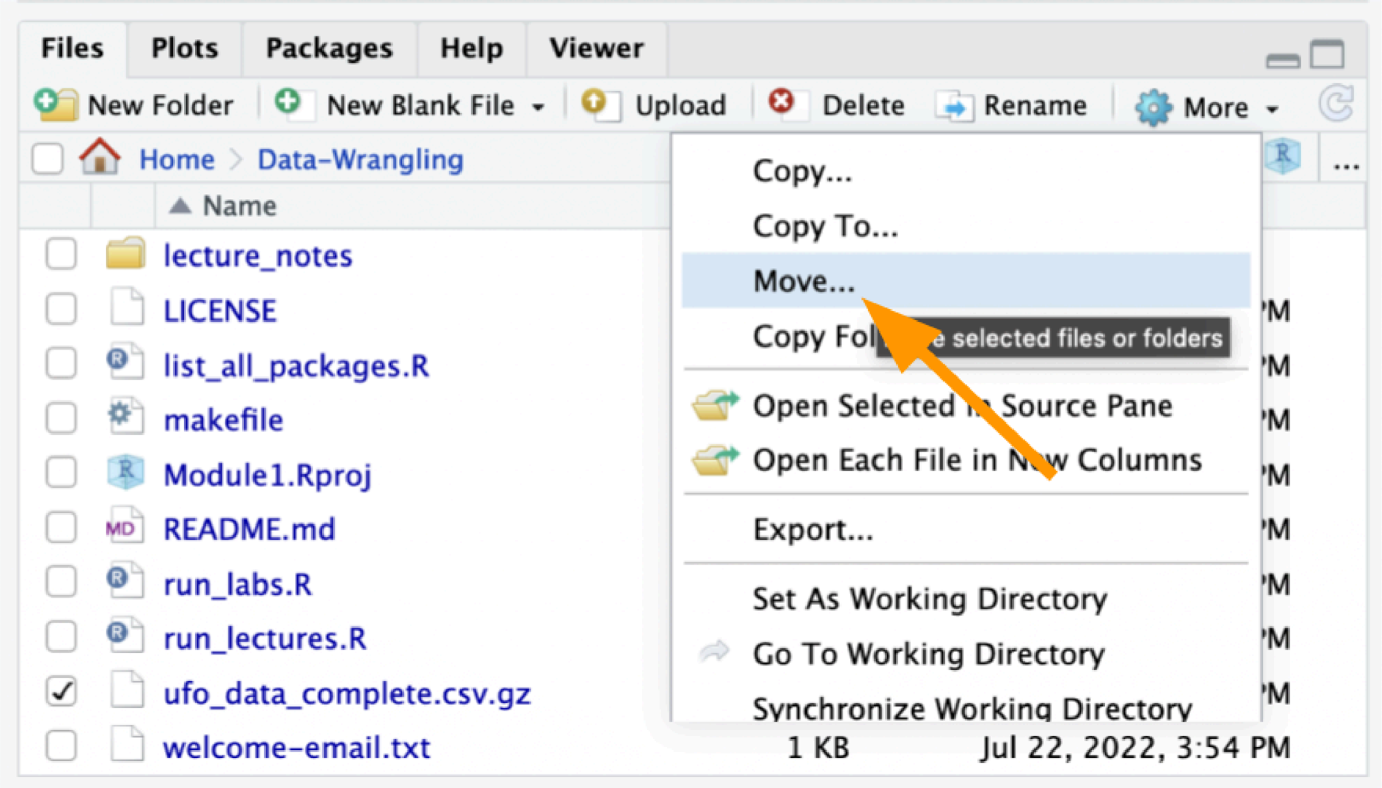
Clean the data while you read it in!

The argument `trim_ws` removes trailing and leading spaces around your data.

```
# example:
read_csv(file = "file.txt", trim_ws = TRUE)
```

Data Input - working directories

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, call `problems()` on your data frame for
  dat <- vroom(...)
  problems(dat)

Rows: 88875 Columns: 11
── Column specification ──────────────────────────────────────────────
Delimiter: ","
chr (10): datetime, city, state, country, shape, duration (hours/min), comme.
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
```

Check the data + other formats

Check the data out

- 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
  - `str()` gives the column types and specs
  - `glimpse()` similar to `str` (dplyr package)

What did I just read in?

- `nrow()` displays the number of rows of a data frame
- `ncol()` displays the number of columns
- `dim()` displays a vector of length 2: # rows, # columns

```
nrow(ufo)
```

```
[1] 88875
```

```
ncol(ufo)
```

```
[1] 11
```

```
dim(ufo)
```

```
[1] 88875    11
```

All Column Names

- `colnames()` displays the column names

```
colnames(ufo)
```

```
 [1] "datetime"             "city"                 "state"
 [4] "country"              "shape"                "duration (seconds)"
 [7] "duration (hours/min)" "comments"             "date posted"
[10] "latitude"             "longitude"
```

Column names and classes using **`glimpse()`**

```
glimpse(ufo)
```

```
Rows: 88,875
Columns: 11
$ datetime                 <chr> "10/10/1949 20:30", "10/10/1949 21:00", "10/10/
$ city                     <chr> "san marcos", "lackland afb", "chester (uk/engl
$ state                    <chr> "tx", "tx", NA, "tx", "hi", "tn", NA, "ct", "al
$ country                  <chr> "us", NA, "gb", "us", "us", "us", "gb", "us", "
$ shape                    <chr> "cylinder", "light", "circle", "circle", "light
$ `duration (seconds)`     <dbl> 2700, 7200, 20, 20, 900, 300, 180, 1200, 180, 1
$ `duration (hours/min)`   <chr> "45 minutes", "1-2 hrs", "20 seconds", "1/2 hou
$ comments                 <chr> "This event took place in early fall around 194
$ `date posted`            <chr> "4/27/2004", "12/16/2005", "1/21/2008", "1/17/2
$ latitude                 <chr> "29.8830556", "29.38421", "53.2", "28.9783333",
$ longitude                <chr> "-97.9411111", "-98.581082", "-2.916667", "-96
```

Data Input

- Sometimes you get weird messages when reading in data.
- The problems()` function shows you any issues with the data read-in.

```
head(problems(ufo))
```

```
# A tibble: 6 × 5
    row   col expected   actual     file
  <int> <int> <chr>      <chr>      <chr>
1   878    12 11 columns 12 columns /Users/avahoffman/Dropbox/JHSPH/Data-Wrang
2  1713    12 11 columns 12 columns /Users/avahoffman/Dropbox/JHSPH/Data-Wrang
3  1815    12 11 columns 12 columns /Users/avahoffman/Dropbox/JHSPH/Data-Wrang
4  2858    12 11 columns 12 columns /Users/avahoffman/Dropbox/JHSPH/Data-Wrang
5  3734    12 11 columns 12 columns /Users/avahoffman/Dropbox/JHSPH/Data-Wrang
6  4756    12 11 columns 12 columns /Users/avahoffman/Dropbox/JHSPH/Data-Wrang
```

```
dim(problems(ufo))
```

```
[1] 199   5
```

Data input: other file types

- For reading Excel files, you can do one of:
  - use `read_excel()` function from `readxl` package
  - use other packages: `xlsx`, `openxlsx`
- `haven` package has functions to read SAS, SPSS, Stata formats

Selecting Excel sheets

Use the `sheet` argument to indicate which sheet to pull from. It can refer to the sheet’s index or name.

```
# example:
read_excel(path = "file.xlsx", sheet = 2)
read_excel(path = "file.xlsx", sheet = "data")
```

After hours of cleaning… output!

Data Output

While its nice to be able to read in a variety of data formats, it's equally important to be able to output data somewhere.

`write_delim()`: Write a data frame to a delimited file `write_csv()`: Write a data frame to a comma-delimited file

This is about twice as fast as `write.csv()`, and never writes row names.

Data Output

For example, we can write back out just the first 100 lines of the `ufo` dataset:

```r
first_100 <- ufo[1:100,]
write_delim(first_100, file = "ufo_first100.csv", delim = ",")
write_csv(first_100, file = "ufo_first100.csv")
```

More ways to save: `write_rds`

If you want to save **one** object, you can use `readr::write_rds` to save to a compressed `rds` file:

```
write_rds(ufo, file = "ufo_dataset.rds", compress = "xz")
```

Read it back in:

```
ufo_new <- read_rds(file = "ufo_dataset.rds")
```

Tip: the `compress = "xz"` argument saves on file size!

More ways to save: **save**

The `save` command can save a set of `R` objects into an “R data file”, with the extension `.rda` or `.RData`.

```r
x = 5
save(ufo, x, file = "ufo_data.rda")
```

The opposite of `save` is `load`.

```r
load(file = "ufo_data.rda")
```

Summary & Lab

- Use `read_delim()`, `read_csv()`, `read_table()` for common data types
- These have helpful `trim_ws` and `na` arguments!
- `read_excel()` has the `sheet` argument for reading from different sheets of the Excel file
- Many functions like `str()`, `View()`, and `glimpse()` can help you understand your data better
- Save your data with `write_delim()` and `write_csv()`

https://sisbid.github.io/Data-Wrangling/02_Data_IO/lab/data-io-lab-part2.Rmd