

Reading data files

Introduction

- First thing we need to do is to read in data, so that we can use our software to analyze.
- Consider these:
 - ▶ Spreadsheet data saved as .csv file.
 - ▶ “Delimited” data such as values separated by spaces.
 - ▶ Actual Excel spreadsheets.

Packages for this section

```
library(tidyverse)
```

A spreadsheet

The screenshot shows the LibreOffice Calc application window titled "test1.xlsx - LibreOffice Calc". The menu bar includes File, Edit, View, Insert, Format, Sheet, Data, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and formatting. The status bar at the bottom shows "Sheet 1 of 1", "Default", "Sum=0", and "200%".

The spreadsheet data is as follows:

	A	B	C	D	E
1	id	x	y	group	
2	p1	10	21	upper	
3	p2	11	20	lower	
4	p3	13	25	upper	
5	p4	15	27	lower	
6	p5	16	30	upper	
7	p6	17	31	lower	
8					
9					
10					
11					

Save as .csv

- .csv or “comma-separated values” is a way of turning spreadsheet values into plain text.
- Easy to read into R
- but does not preserve formulas. (This is a reason for doing all your calculations in your statistical software, and only having data in your spreadsheet.)
- File, Save As Text CSV (or similar).
- used name test1.csv.

The .csv file

```
id,x,y,group  
p1,10,21,upper  
p2,11,20,lower  
p3,13,25,upper  
p4,15,27,lower  
p5,16,30,upper  
p6,17,31,lower
```

To read this in:

- Fire up R Studio at r.datatools.utoronto.ca
- Upload this .csv file. (Bottom right, next to New Folder, Upload.)
Click Choose File, find the file, click Open. Click OK. See the file appear bottom right.

Make a new Quarto document

- File, New File, Quarto Document
- ...and get rid of the template document (leaving the first four lines).
- Make a code chunk and in it put this. Run it.

```
library(tidyverse)
```

Reading in the file

- Use `read_csv` with the name of the file, in quotes. Save the read-in file in something, here called `mydata`. Make a new code chunk for this:

```
mydata <- read_csv("test1.csv")  
mydata
```

```
# A tibble: 6 x 4  
  id      x      y group  
  <chr> <dbl> <dbl> <chr>  
1 p1      10     21 upper  
2 p2      11     20 lower  
3 p3      13     25 upper  
4 p4      15     27 lower  
5 p5      16     30 upper  
6 p6      17     31 lower
```


More on the above

- `read_csv` guesses what kind of thing is in each column. Here it correctly guesses that:
 - ▶ `id` and `group` are text (categorical variables). `id` is actually “identifier variable”: identifies individuals.
 - ▶ `x` and `y` are “double”: numbers that might have a decimal point in them.

R Studio on your own computer

- Put the .csv file in the same folder as your project. Then read it in as above like `read_csv("test1.csv")`.
- Or, use

```
# f <- file.choose()  
f
```

which brings up a file selector (as if you were going to find a file to load or save it). Find your .csv file, the address of which will be saved in `f`, and then:

```
mydata <- read_csv(f)
```

- When you have selected the file, comment out the `file.choose` line by putting a `#` on the front of it. That will save you having to find the file again by mistake. (Keyboard shortcut: go to the line, type control-shift-C or Mac equivalent with Cmd.)

Looking at what we read in

- Again, type the name of the thing to display it:

```
mydata
```

```
# A tibble: 6 x 4
  id      x      y group
<chr> <dbl> <dbl> <chr>
1 p1      10     21 upper
2 p2      11     20 lower
3 p3      13     25 upper
4 p4      15     27 lower
5 p5      16     30 upper
6 p6      17     31 lower
```

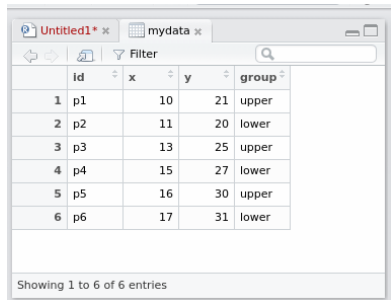
- This is a “tibble” or data frame, the standard way of storing a data set in R.
- Tibbles print as much as will display on the screen. If there are more rows or columns, it will say so.
- You might see navigation keys to display more rows or columns (if there are more).

View-ing your data frame

- Another way to examine your data frame is to View it, like this:

```
View(mydata)
```

- ...or find your data frame in the Global Environment top right and click it.
- This pops up a “data frame viewer” top left:



The screenshot shows a window titled 'Untitled1 * x' with a tab for 'mydata x'. Below the tab is a toolbar with navigation icons and a 'Filter' input field. The main area displays a table with the following data:

	id	x	y	group
1	p1	10	21	upper
2	p2	11	20	lower
3	p3	13	25	upper
4	p4	15	27	lower
5	p5	16	30	upper
6	p6	17	31	lower

Showing 1 to 6 of 6 entries

This View

- Read-only: cannot edit data
- Can display data satisfying conditions: click on Filter, then:
 - ▶ for a categorical variable, type name of category you want
 - ▶ for a quantitative variable, use slider to describe values you want.
- Can sort a column into ascending or descending order (click little arrows next to column name).
- Clicking the symbol with arrow on it left of Filter “pops out” View into separate (bigger) window.

Summarizing what we read in

- It is always a good idea to look at your data after you have read it in, to make sure you have believable numbers (and the right number of individuals and variables).
- Quick check for errors: these often show up as values too high or too low, so the min and/or max will be unreasonable.
- Five-number summary:

```
summary(mydata)
```

id	x	y	group
Length:6	Min. :10.00	Min. :20.00	Length:6
Class :character	1st Qu.:11.50	1st Qu.:22.00	Class :character
Mode :character	Median :14.00	Median :26.00	Mode :character
	Mean :13.67	Mean :25.67	
	3rd Qu.:15.75	3rd Qu.:29.25	
	Max. :17.00	Max. :31.00	

- Quantitative, five-number summary plus mean.
- Categorical, how many rows.

Reading from a URL

- Any data file on the Web can be read directly.
- **Example data link:**
- Use URL instead of filename.
- I like to save the URL in a variable first (because URLs tend to be long), and then put that variable in the `read_` function:

```
my_url <- "http://ritsokiguess.site/datafiles/global.csv"
my_url
```

```
[1] "http://ritsokiguess.site/datafiles/global.csv"
```

```
global <- read_csv(my_url)
```

The data

```
global
```

```
# A tibble: 10 x 3
  warehouse size cost
  <chr>      <dbl> <dbl>
1 A         225 12.0
2 B         350 14.1
3 A         150  8.93
4 A         200 11.0
5 A         175 10.0
6 A         180 10.1
7 B         325 13.8
8 B         290 13.3
9 B         400 15
10 A        125  7.97
```


Space-delimited files

- Another common format for data is a text file with the values separated by spaces. Top of some other data:

```
cup tempdiff
Starbucks 13
Starbucks 7
Starbucks 7
Starbucks 17.5
Starbucks 10
Starbucks 15.5
Starbucks 6
Starbucks 6
SIGG 12
SIGG 16
SIGG 9
SIGG 23
SIGG 11
SIGG 20.5
SIGG 12.5
SIGG 20.5
SIGG 24.5
CUPPS 6
CUPPS 6
```

Reading the coffee data

- This file was on my computer so I uploaded it to `r.datatools.utoronto.ca` first.
- This time, `read_delim`, and we also have to say what the thing is separating the values:

```
coffee <- read_delim("coffee.txt", " ")
```

```
Rows: 32 Columns: 2
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
chr (1): cup
```

```
dbl (1): tempdiff
```

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.

- Name of the cup, text, and tempdiff, a decimal number.

Looking at the values

```
coffee
```

```
# A tibble: 32 x 2
  cup      tempdiff
<chr>      <dbl>
1 Starbucks    13
2 Starbucks     7
3 Starbucks     7
4 Starbucks   17.5
5 Starbucks    10
6 Starbucks   15.5
7 Starbucks     6
8 Starbucks     6
9 SIGG        12
10 SIGG        16
# i 22 more rows
```

These were four brands of travel mug (in cup), and for each, how much the temperature of the coffee in the mug decreased over 30 minutes.

Reading from the Web; the soap data

- Use the URL in place of the filename.
- Save the URL in a variable first:

```
my_url <- "http://ritsokiguess.site/datafiles/soap.txt"  
soap <- read_delim(my_url, " ")
```

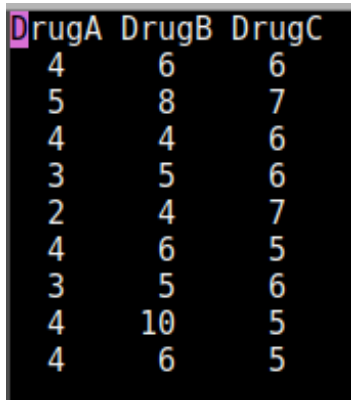
The soap data (some)

```
soap
```

```
# A tibble: 27 x 4
  case scrap speed line
  <dbl> <dbl> <dbl> <chr>
1     1    218   100 a
2     2    248   125 a
3     3    360   220 a
4     4    351   205 a
5     5    470   300 a
6     6    394   255 a
7     7    332   225 a
8     8    321   175 a
9     9    410   270 a
10    10    260   170 a
# i 17 more rows
```

Data aligned in columns

- Sometimes you see data aligned in columns, thus:



DrugA	DrugB	DrugC
4	6	6
5	8	7
4	4	6
3	5	6
2	4	7
4	6	5
3	5	6
4	10	5
4	6	5

- `read_delim` will not work: values separated by more than one space.
- The number of spaces between values is not constant, because there is one fewer space before the 10.
- `read_table` works for this.

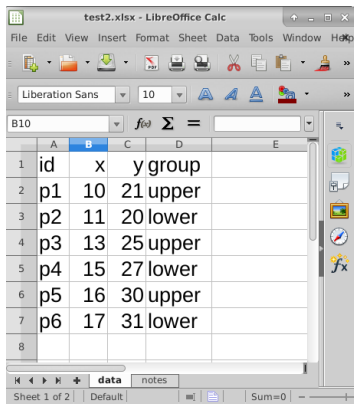
Reading in column-aligned data

```
drugs <- read_table("migraine.txt")  
drugs
```

```
# A tibble: 9 x 3  
  DrugA DrugB DrugC  
  <dbl> <dbl> <dbl>  
1     4     6     6  
2     5     8     7  
3     4     4     6  
4     3     5     6  
5     2     4     7  
6     4     6     5  
7     3     5     6  
8     4    10     5  
9     4     6     5
```

Reading an Excel sheet directly

- Here is **my spreadsheet** from before, but tarted up a bit:



The screenshot shows the LibreOffice Calc application window titled 'test2.xlsx - LibreOffice Calc'. The interface includes a menu bar (File, Edit, View, Insert, Format, Sheet, Data, Tools, Window, Help), a toolbar with various icons, and a status bar at the bottom. The spreadsheet is displayed with columns A through E and rows 1 through 8. The 'data' sheet is active, showing the following data:

	A	B	C	D	E
1	id	x	y	group	
2	p1	10	21	upper	
3	p2	11	20	lower	
4	p3	13	25	upper	
5	p4	15	27	lower	
6	p5	16	30	upper	
7	p6	17	31	lower	
8					

The status bar at the bottom indicates 'Sheet 1 of 2', 'Default', and 'Sum=0'. The 'notes' sheet is also visible in the sheet tab bar.

- Now a workbook with a second sheet called “notes”.

Reading it in

- Read into R, saying that we only want the sheet “data”. Upload spreadsheet first.
- Excel spreadsheets must be “local”: cannot read one in from a URL.

```
library(readxl) # install first (once) with install.packages
mydata2 <- read_excel("test2.xlsx", sheet = "data")
mydata2
```

```
# A tibble: 6 x 4
```

	id	x	y	group
	<chr>	<dbl>	<dbl>	<chr>
1	p1	10	21	upper
2	p2	11	20	lower
3	p3	13	25	upper
4	p4	15	27	lower
5	p5	16	30	upper
6	p6	17	31	lower