

Lecture 2.3 – Reading and Writing Files

Specific Learning Objectives:

3.1 – Load, clean, and organize data in R.

#OtherPeoplesData

- Up until this point, we've used the base R data sets to explore how R works, but eventually, you'll need to start using R with real data sets.
 - Reading in and cleaning up data is the source of frustration for a lot of people on Twitter, who use the #OtherPeoplesData tag to discuss the “creative” ways in which people record their data that make it difficult to use!

Dr Rachel Hale, a vaccinated worm blob! @SeafloorScienc... · Sep 14 ...

This is the worst example of #OtherPeoplesData I've ever seen

Jonathan Carroll @carroll_

Today's fresh #OtherPeop...

It turns out that if someone an asterisk it may entirely b surprise Excel still thinks it'

```
readxl::read_excel("NoAsterisk.xlsx")
# A tibble: 6 x 2
  Date       Value
  <dtm>     <dbl>
1 2020-11-21 00:00:00 321
2 2020-11-22 00:00:00 322
3 2020-11-23 00:00:00 323
4 2020-11-24 00:00:00 324
5 2020-11-25 00:00:00 325
6 2020-11-26 00:00:00 326
```

```
> readxl::read_excel("Asterisk.xlsx")
# A tibble: 6 x 2
  Date       Value
  <chr>      <dbl>
1 44156        321
2 44157        322
3 23/11/2020*   323
4 44159        324
5 44160        325
6 44161        326
```

OpenElections @openelix · Apr 17, 2017

City of Detroit produced a lookup tables for its absentee precincts in 2016. It's in Excel. But wait for it: the values are CLIP ART.

Mar 29, 2017 ...

pler is BLPW. Not Bkpw, or bcpw, sdata

9

18

thought that typing "<1" rather #OtherPeoplesData Many numeric his!

8

Data Reading and Cleaning Skills

- Reading in and cleaning data sets are good skills to have, especially when it comes to reading other people's data!
 - We'll work on an easy set that needs some cleaning, but not a lot.
 - We'll work on examples from several different file types (although I will encourage everyone to save raw data in non-proprietary data formats like CSV).
 - We'll practice saving data in a way that will make sense later on!

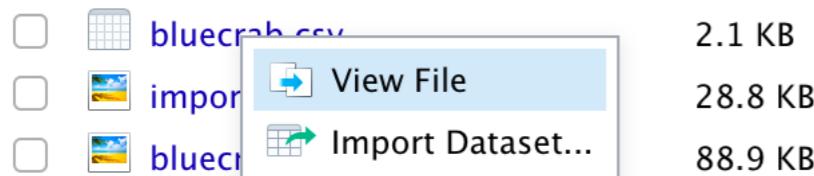
Importing Data

Be sure to download the `data.zip` file in Lecture Notes and unzip it!

- **Importing data:** loading data from storage into memory for immediate use.
 - Remember that most data in storage is not immediately available for use by the computer. It must read this from storage into memory!
 - Importing data into R not only reads in, but will often format the data into what it guesses are appropriate data classes and organizations (this also depends on the function that you use).
- Start by importing `bluecrab.csv` into R. (This is data from a chapter of my dissertation.)
 - Run this chunk:

```
```{r, include=TRUE}
bluecrab.data <- read.csv("bluecrab.csv")
View(bluecrab.data)
...```

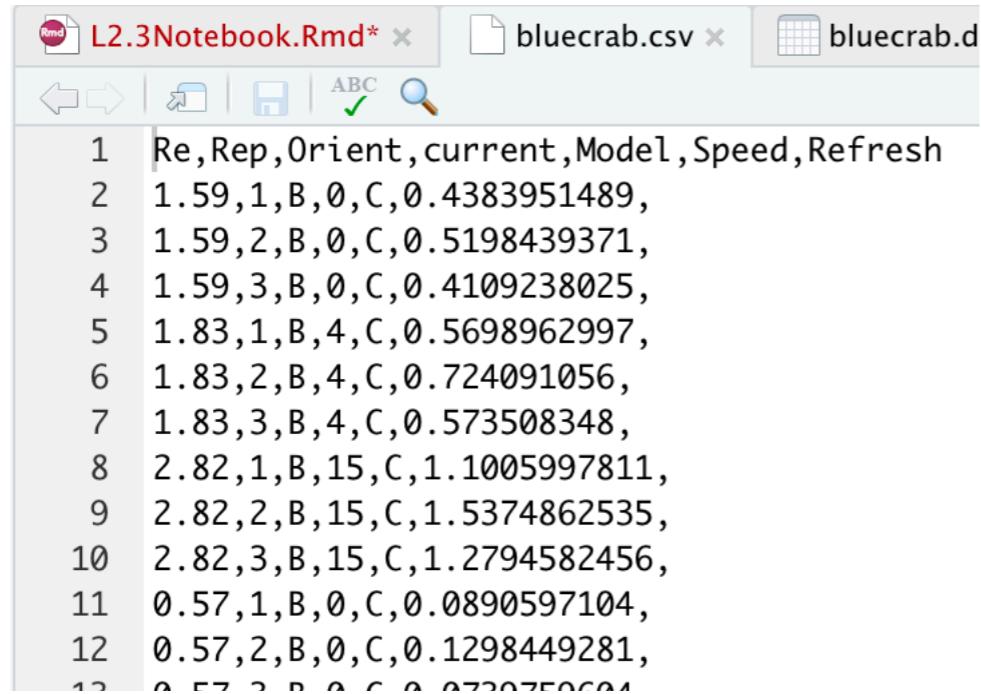
```
  - View the file by clicking it:



	Re	Rep	Orient	current	Model	Speed	Refresh
1	1.59	1	B	0	C	0.43839515	NA
2	1.59	2	B	0	C	0.51984394	NA
3	1.59	3	B	0	C	0.41092380	NA
4	1.83	1	B	4	C	0.56989630	NA
5	1.83	2	B	4	C	0.72409106	NA
6	1.83	3	B	4	C	0.57350835	NA
7	2.82	1	B	15	C	1.10059978	NA
8	2.82	2	B	15	C	1.53748625	NA
9	2.82	3	B	15	C	1.27945825	NA

# Common File Types

- **Text files:** data files consisting of plain text with values separated by (delineated with) a specific character.



L2.3Notebook.Rmd\*    bluecrab.csv    bluecrab.d

1 Re,Rep,Orient,current,Model,Speed,Refresh  
2 1.59,1,B,0,C,0.4383951489,  
3 1.59,2,B,0,C,0.5198439371,  
4 1.59,3,B,0,C,0.4109238025,  
5 1.83,1,B,4,C,0.5698962997,  
6 1.83,2,B,4,C,0.724091056,  
7 1.83,3,B,4,C,0.573508348,  
8 2.82,1,B,15,C,1.1005997811,  
9 2.82,2,B,15,C,1.5374862535,  
10 2.82,3,B,15,C,1.2794582456,  
11 0.57,1,B,0,C,0.0890597104,  
12 0.57,2,B,0,C,0.1298449281,

- **Comma-separate Values (CSV):** values are separated by commas.
- **Tab-separate Values (TSV):** values are separated by tabs.
- Other separators: decimal, semi-colon, space, just about anything.

- **Spreadsheet files:** data files of spreadsheets meant to be opened by specific applications (Excel, Numbers, Google Sheets)

## Mac Numbers

PK?j0S?\*y-??Index/Document.iwa??\?e?  
" ??  
\*Γ7Υ7??7Γ7??7??7??7Γ7  
??7?7Γ7??7B?  
?  
?P`x???J  
1678.10 gregorianlatn"JanuaryFebru  
\March"April"May"Junehly"August"  
September"Octo  
Nov  
Dec

## Microsoft Excel

?A>\?C?f?g?j\_[\_??K?^P??kPDIr??..?jw??d???  
?A??)?)Q?  
??RSLX"??Z??2>?R\$!???  
JU??>??#??02??]`?X?xb??L??+?? /?= {????=????\_????\_????\*Kn%SS?S\_????\_?????7??'??L??-??  
?:????/?}????}??O?????i????c?&a?????0B@?v????^[ ??u????X??<?k?\$\$??F??c?vw:q?  
??8??C??C??=2?L??????Cr %R.?Cb??e e ?????m?èk?=|d#a[ ?0~?????h.Q?R9D15??  
d?2r?G&?/\$Dz?)??c,?K?:?A?  
??]??6?Kr?w?3?=v .P<s?L??~&!E?w?I|??;D=CP?1\_????f"?j??'??e?????t??e?<?3q?:  
?jSt??{?>sX?a3??W`??J?+?U??`e??k?)r+e??m?goq?x(?h?D?J]8?Tz??M?5???)??0??IYg?  
z?|]p+~o??`\_?=?|j ???Qk?<a?  
??\D??Z1?FV?????w?'?w?ws?[???:?(e?D?? ,kzh??p??s?yUs^?????f?^>?e?k??Z????Aj?  
?|??&??O??3!??ZBw}?b0??Q?'?j?"??5?,?#-q?&'23?  
ZCe????L?Tx3&?c??u+?9?N?x??Ng?  
????  
??x)\?C?J??Z=???~TwY?(??aLfuQ\_B^g^X?tX?P&ZFq?  
0mx??E?A?A?f??c?????  
IFz??Pb/3 ?tS?qyju??E?-#?t??0?0??,;?Yz??20?b?3?kE" '??&S??;n??\*#4k????x#?[  
4??m+W>Z@+?qt;x2#?i?Q?N??S?p????\$?%?????7?XX/+?????r??1?????w?`?h9???.?#:  
??CP\*?n?N-r?k.??yJ??})?0-??2MY?N?E?Q?3,?;?  
O?6?muF8=?'3?Zu@,?J?????fw?<zP8?R?PBo#?(  
??Ü`?9?~-?~)1?J?-?a?T?R????vV???\u00a0\*`?Q\????\a?Evi??????.??.?L\_?????\\?|q? ????qY

# Importing Data: `read.table()`

- `read.table()` is a function in base R (utils package) which imports data from text files as a data frame.
  - many options for dealing with different types of data:
    - header (logical)**: will assign headers of columns as column labels
    - sep (character)**: will recognize given character as the separator. Ex: “,” for comma-separated values.
    - skip (numeric/integer)**: will skip the number of rows specified at the start of the file.
    - nrows (numeric/integer)**: maximum number of rows to read in.
    - row.names (character/integer)**: either a vector that gives row names or a single number that specifies which column of the data should be the row names.
    - colClasses (character)**: a vector of characters that specify what data types each column should be imported as.
    - stringsAsFactors (logical)**: specifies whether or not you want character strings to be read in as factors (overwritten by colClasses, applies to all columns).

# Importing Data: `read.table()`

- `read.table()` is a function in base R (utils package) which imports data from text files as a data frame.
  - `read.csv()` is a special case of `read.table()` where `sep=","`
  - `read.delim()` is a special case of `read.table()` where `sep=". "`
- Going back to our original import...

```
```{r, include=TRUE}
bluecrab.data <- read.csv("bluecrab.csv")
View(bluecrab.data)
```
```

The screenshot shows the RStudio interface. On the left, the code editor contains R code for importing a CSV file and viewing its contents. On the right, the data browser shows the imported data as a data frame named 'bluecrab.data'. A large black arrow points from the code editor towards the data browser.

|    | Re   | Rep | Orient | current | Model | Speed        | Refresh |
|----|------|-----|--------|---------|-------|--------------|---------|
| 1  | 1.59 | 1   | B      | 0       | C     | 0.4383951489 | NA      |
| 2  | 1.59 | 2   | B      | 0       | C     | 0.5198439371 | NA      |
| 3  | 1.59 | 3   | B      | 0       | C     | 0.4109238025 | NA      |
| 4  | 1.83 | 1   | B      | 4       | C     | 0.5698962997 | NA      |
| 5  | 1.83 | 2   | B      | 4       | C     | 0.724091056  | NA      |
| 6  | 1.83 | 3   | B      | 4       | C     | 0.573508348  | NA      |
| 7  | 2.82 | 1   | B      | 15      | C     | 1.1005997811 | NA      |
| 8  | 2.82 | 2   | B      | 15      | C     | 1.5374862535 | NA      |
| 9  | 2.82 | 3   | B      | 15      | C     | 1.2794582456 | NA      |
| 10 | 0.57 | 1   | B      | 0       | C     | 0.0890597104 | NA      |
| 11 | 0.57 | 2   | B      | 0       | C     | 0.1298449281 | NA      |
| 12 | 0.57 | 3   | B      | 0       | C     | 0.2720750001 | NA      |

# Importing Data: Not-so-perfect

- Now let's look at a not-so-perfect data set and see what we can't do to import it.
  - First try of importing babycrab1.csv

**character class???**

|   | H..oregonensis | X         | X.1              | X.2 | X.3           | X.4         | X.5               | X.6          |
|---|----------------|-----------|------------------|-----|---------------|-------------|-------------------|--------------|
| 1 | number         | size (mm) | log size         | sex | Hair diameter | Hair length | Cuticle thickness | Hair-bearing |
| 2 | 1              | 27        | 3.29583686600433 | m   | 8.1E-06       | 648         | 7.68E-07          | 11           |
| 3 | 2              | 8.5       | 2.14006616349627 | f   | 6.73E-06      | 465         | 7.19E-07          | 9            |
| 4 | 3              | 11        | 2.39789527279837 | f   |               |             |                   |              |
| 5 | 4              | 5         | 1.6094379124341  | f   | 5.68E-06      | 374         |                   | 8            |
| 6 | 5              | 11.5      | 2.4423470353692  | m   | 7.44E-06      | 484         | 8.74E-07          | 9            |

- look at the file babycrab1.csv

**look at this garbage**

```
1 H. oregonensis,,,...,,
2 number,size (mm),log size,sex,Hair diameter,Hair length,Cuticle thickness,Hair-bearing
segments,length of 3rd segment,ratio HL/3L,array width,HL/AW,Cuticle to diameter,,
3 1,27,3.29583686600433,m,8.1E-06,648,7.68E-07,11,240,2.7,202,3.20792079207921,9.48E-02,,
4 2,8.5,2.14006616349627,f,6.73E-06,465,7.19E-07,9,147,3.16326530612245,150,3.1,1.07E-01,,
5 3,11,2.39789527279837,f,,,...,,
6 4,5,1.6094379124341,f,5.68E-06,374,,8,129,2.89922480620155,,,,,
7 5,11.5,2.4423470353692,m,7.44E-06,484,8.74E-07,9,153,3.16339869281046,124,3.90322580645161
,1.17E-01,,
8 6,28,3.3322045101752,m,8.4868E-06,660,1.01E-06,10,272,2.42647058823529,217,3.04147465437788
,1.19E-01,,,
```

# Importing Data: Not-so-perfect

- How can we improve this import?

**First row looks pretty useless.**

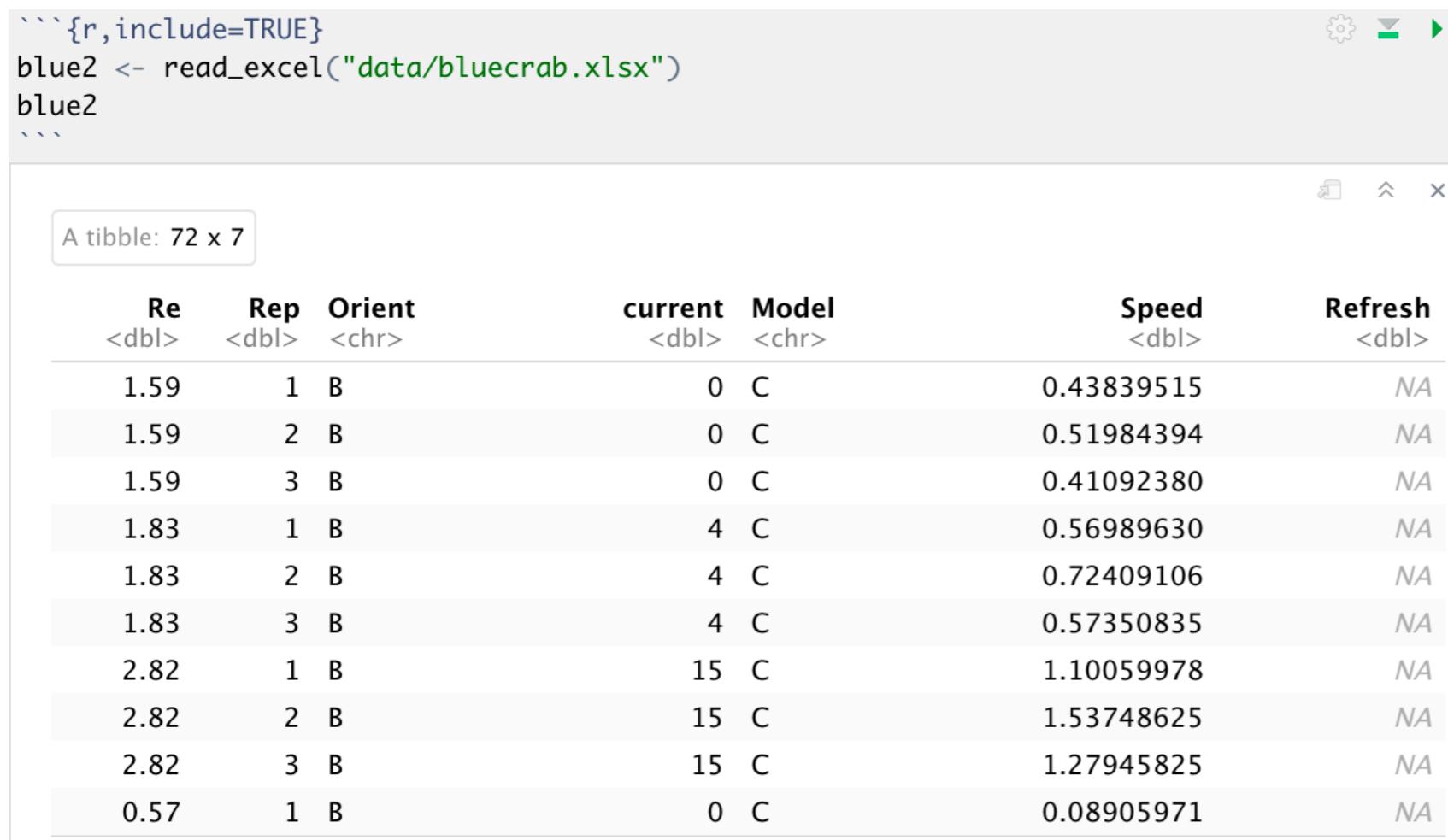
```
L2.3Notebook.Rmd* x babycrab.data x babycrab1.csv x
L2.3Notebook.Rmd* x babycrab.data x babycrab1.csv x
Show whitespace
1 H. oregonensis,,,,,,,,,,,
2 number,size (mm),log size,sex,Hair diameter,Hair length,Cuticle thickness,Hair-bearing
 segments,length of 3rd segment,ratio HL/3L,array width,HL/AW,Cuticle to diameter,,,,
3 1,27,3.29583686600433,m,8.1E-06,648,7.68E-07,11,240,2.7,202,3.20792079207921,9.48E-02,,
4 2,8.5,2.14006616349627,f,6.73E-06,465,7.19E-07,9,147,3.16326530612245,150,3.1,1.07E-01,,
5 3,11,2.39789527279837,f,,,,,,,,,,,
6 4,5,1.6094379124341,f,5.68E-06,374,,8,129,2.89922480620155,,,,,,,
7 5,11.5,2.4423470353692,m,7.44E-06,484,8.74E-07,9,153,3.16339869281046,124,3.90322580645161
 ,1.17E-01,,,,
8 6,28,3.3322045101752,m,8.4868E-06,660,1.01E-06,10,272,2.42647058823529,217,3.04147465437788
 ,1.19E-01,,,
```

**second row looks like it might  
be the column names...**

**Let's try to use skip = 1**

# Importing Data: Non-text File Formats

- Tidyverse package tools:
  - `readxl` package: assists in reading xls and xlsx files
- Read Excel file using `read_excel()`:



The screenshot shows an RStudio session with the following code:

```
```{r, include=TRUE}
blue2 <- read_excel("data/bluecrab.xlsx")
blue2
````
```

The output is a tibble with 72 rows and 7 columns:

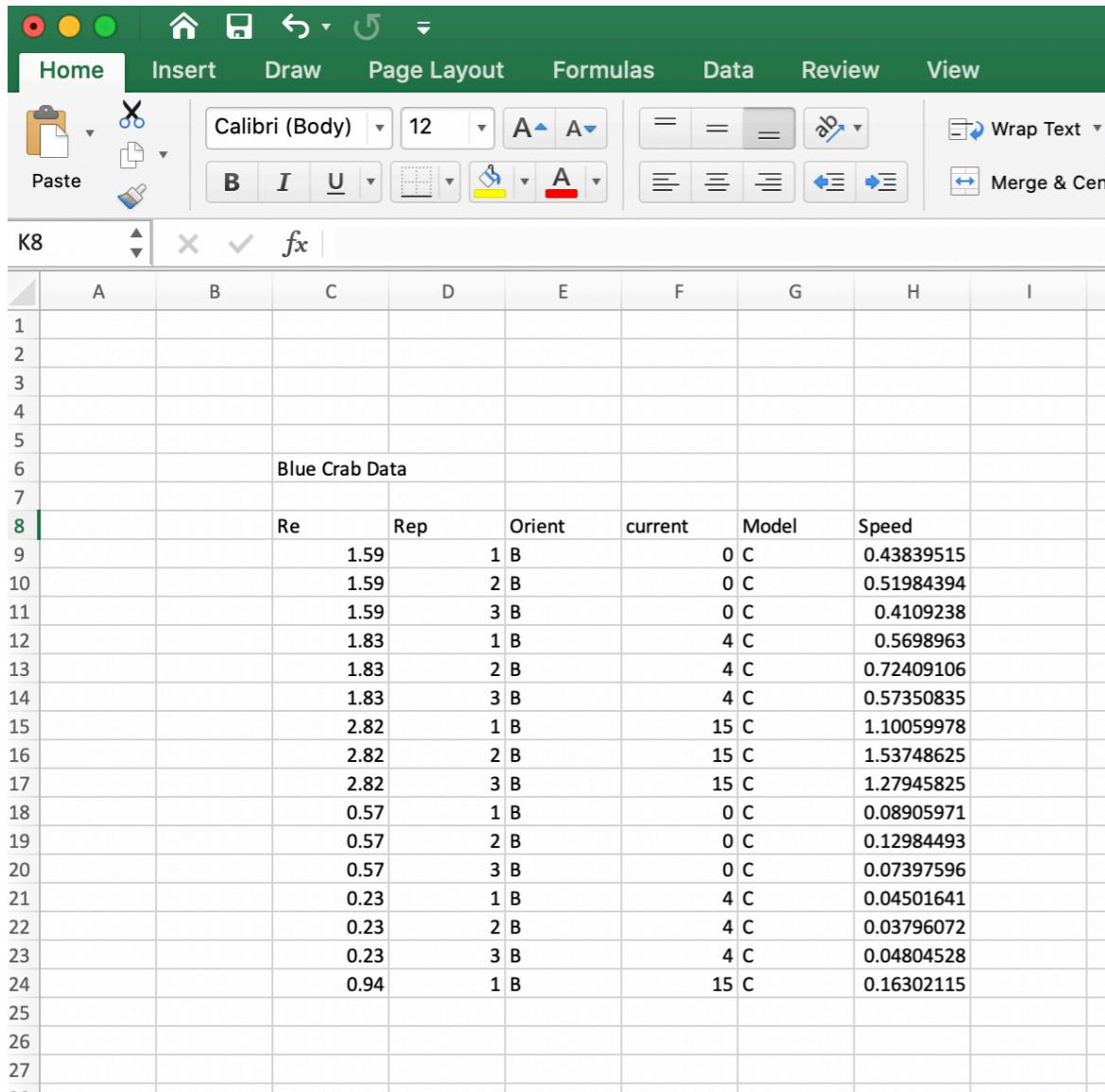
| Re    | Rep   | Orient | current | Model | Speed      | Refresh |
|-------|-------|--------|---------|-------|------------|---------|
| <dbl> | <dbl> | <chr>  | <dbl>   | <chr> | <dbl>      | <dbl>   |
| 1.59  | 1     | B      | 0       | C     | 0.43839515 | NA      |
| 1.59  | 2     | B      | 0       | C     | 0.51984394 | NA      |
| 1.59  | 3     | B      | 0       | C     | 0.41092380 | NA      |
| 1.83  | 1     | B      | 4       | C     | 0.56989630 | NA      |
| 1.83  | 2     | B      | 4       | C     | 0.72409106 | NA      |
| 1.83  | 3     | B      | 4       | C     | 0.57350835 | NA      |
| 2.82  | 1     | B      | 15      | C     | 1.10059978 | NA      |
| 2.82  | 2     | B      | 15      | C     | 1.53748625 | NA      |
| 2.82  | 3     | B      | 15      | C     | 1.27945825 | NA      |
| 0.57  | 1     | B      | 0       | C     | 0.08905971 | NA      |

- Works OK for simple spreadsheets, but will trip with complicated sheets and other “objects” like graphs
- Consider exporting your spreadsheet as a text file (e.g. CSV) for use in R!

# Using Spreadsheets Safely

- I probably can't convince you to not use spreadsheets, so you may as well use them SAFELY (so you can't upload the data to R later).
  - Align your data in the upper left, leaving no extra columns on either side.

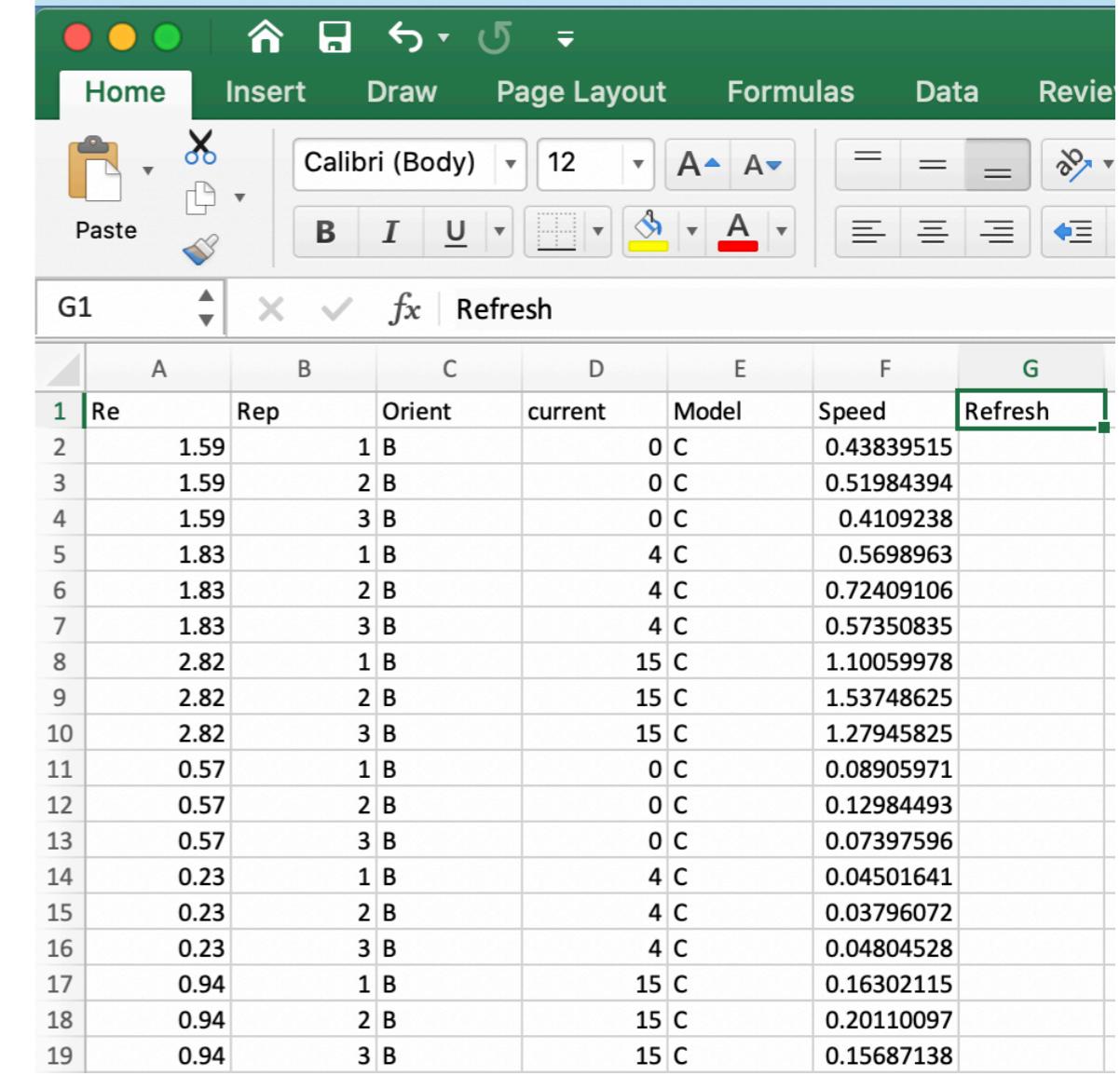
**DON'T** 🤫



A screenshot of a spreadsheet application showing a table titled "Blue Crab Data". The table has columns labeled "Re", "Rep", "Orient", "current", "Model", and "Speed". The data is aligned to the left, with empty columns A through I present in the row headers. The formula bar shows "fx".

|                | A  | B    | C      | D       | E     | F          | G | H | I |
|----------------|----|------|--------|---------|-------|------------|---|---|---|
| 1              |    |      |        |         |       |            |   |   |   |
| 2              |    |      |        |         |       |            |   |   |   |
| Blue Crab Data |    |      |        |         |       |            |   |   |   |
| 8              | Re | Rep  | Orient | current | Model | Speed      |   |   |   |
| 9              |    | 1.59 | 1 B    |         | 0 C   | 0.43839515 |   |   |   |
| 10             |    | 1.59 | 2 B    |         | 0 C   | 0.51984394 |   |   |   |
| 11             |    | 1.59 | 3 B    |         | 0 C   | 0.4109238  |   |   |   |
| 12             |    | 1.83 | 1 B    |         | 4 C   | 0.5698963  |   |   |   |
| 13             |    | 1.83 | 2 B    |         | 4 C   | 0.72409106 |   |   |   |
| 14             |    | 1.83 | 3 B    |         | 4 C   | 0.57350835 |   |   |   |
| 15             |    | 2.82 | 1 B    |         | 15 C  | 1.10059978 |   |   |   |
| 16             |    | 2.82 | 2 B    |         | 15 C  | 1.53748625 |   |   |   |
| 17             |    | 2.82 | 3 B    |         | 15 C  | 1.27945825 |   |   |   |
| 18             |    | 0.57 | 1 B    |         | 0 C   | 0.08905971 |   |   |   |
| 19             |    | 0.57 | 2 B    |         | 0 C   | 0.12984493 |   |   |   |
| 20             |    | 0.57 | 3 B    |         | 0 C   | 0.07397596 |   |   |   |
| 21             |    | 0.23 | 1 B    |         | 4 C   | 0.04501641 |   |   |   |
| 22             |    | 0.23 | 2 B    |         | 4 C   | 0.03796072 |   |   |   |
| 23             |    | 0.23 | 3 B    |         | 4 C   | 0.04804528 |   |   |   |
| 24             |    | 0.94 | 1 B    |         | 15 C  | 0.16302115 |   |   |   |
| 25             |    |      |        |         |       |            |   |   |   |
| 26             |    |      |        |         |       |            |   |   |   |
| 27             |    |      |        |         |       |            |   |   |   |

**DO** 🤗



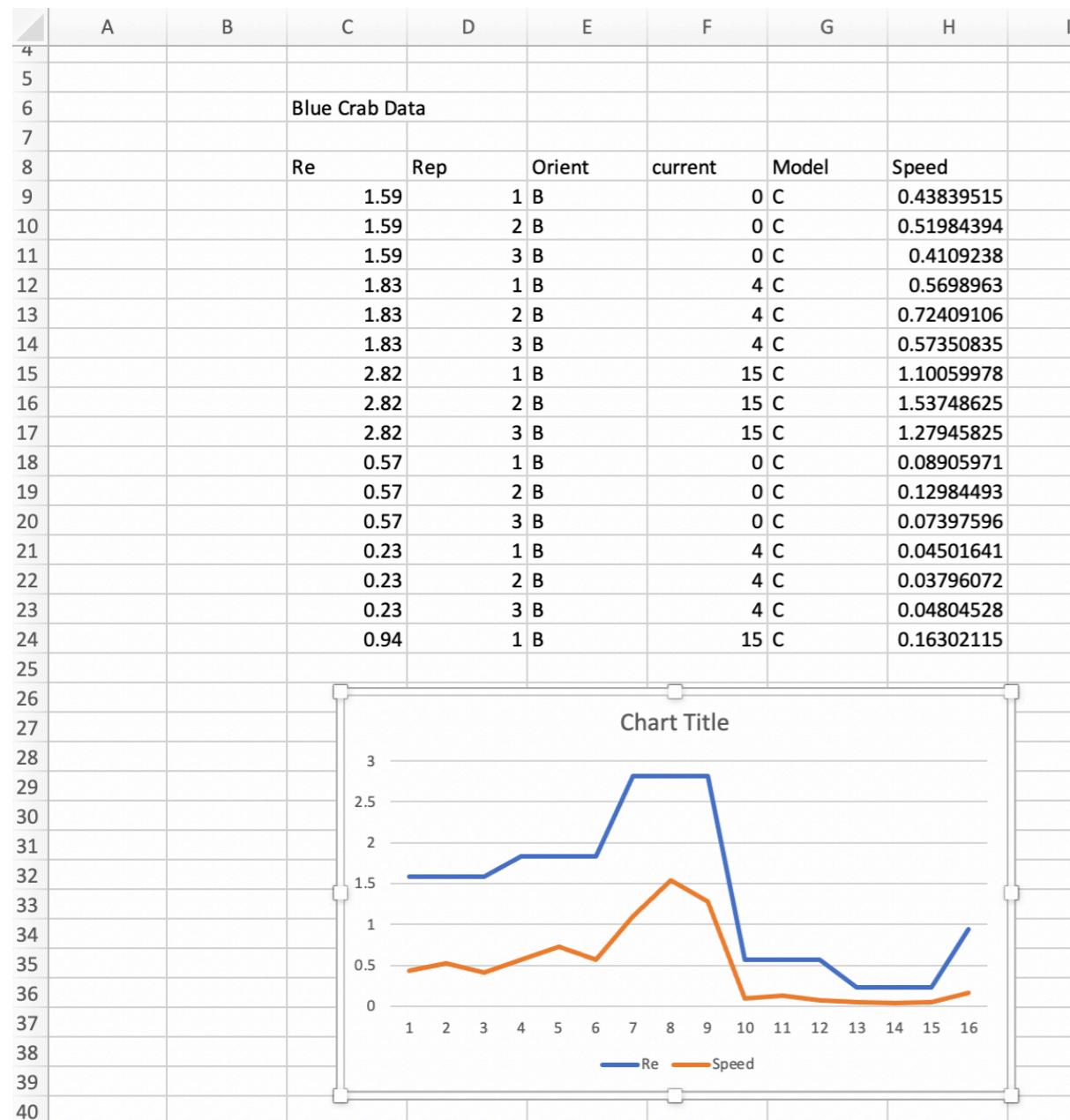
A screenshot of a spreadsheet application showing the same "Blue Crab Data" table. The data is aligned to the left, but there are no empty columns A through I present in the row headers. The formula bar shows "fx Refresh".

|    | A  | B    | C      | D       | E     | F          | G       |
|----|----|------|--------|---------|-------|------------|---------|
| 1  | Re | Rep  | Orient | current | Model | Speed      | Refresh |
| 2  |    | 1.59 | 1 B    |         | 0 C   | 0.43839515 |         |
| 3  |    | 1.59 | 2 B    |         | 0 C   | 0.51984394 |         |
| 4  |    | 1.59 | 3 B    |         | 0 C   | 0.4109238  |         |
| 5  |    | 1.83 | 1 B    |         | 4 C   | 0.5698963  |         |
| 6  |    | 1.83 | 2 B    |         | 4 C   | 0.72409106 |         |
| 7  |    | 1.83 | 3 B    |         | 4 C   | 0.57350835 |         |
| 8  |    | 2.82 | 1 B    |         | 15 C  | 1.10059978 |         |
| 9  |    | 2.82 | 2 B    |         | 15 C  | 1.53748625 |         |
| 10 |    | 2.82 | 3 B    |         | 15 C  | 1.27945825 |         |
| 11 |    | 0.57 | 1 B    |         | 0 C   | 0.08905971 |         |
| 12 |    | 0.57 | 2 B    |         | 0 C   | 0.12984493 |         |
| 13 |    | 0.57 | 3 B    |         | 0 C   | 0.07397596 |         |
| 14 |    | 0.23 | 1 B    |         | 4 C   | 0.04501641 |         |
| 15 |    | 0.23 | 2 B    |         | 4 C   | 0.03796072 |         |
| 16 |    | 0.23 | 3 B    |         | 4 C   | 0.04804528 |         |
| 17 |    | 0.94 | 1 B    |         | 15 C  | 0.16302115 |         |
| 18 |    | 0.94 | 2 B    |         | 15 C  | 0.20110097 |         |
| 19 |    | 0.94 | 3 B    |         | 15 C  | 0.15687138 |         |

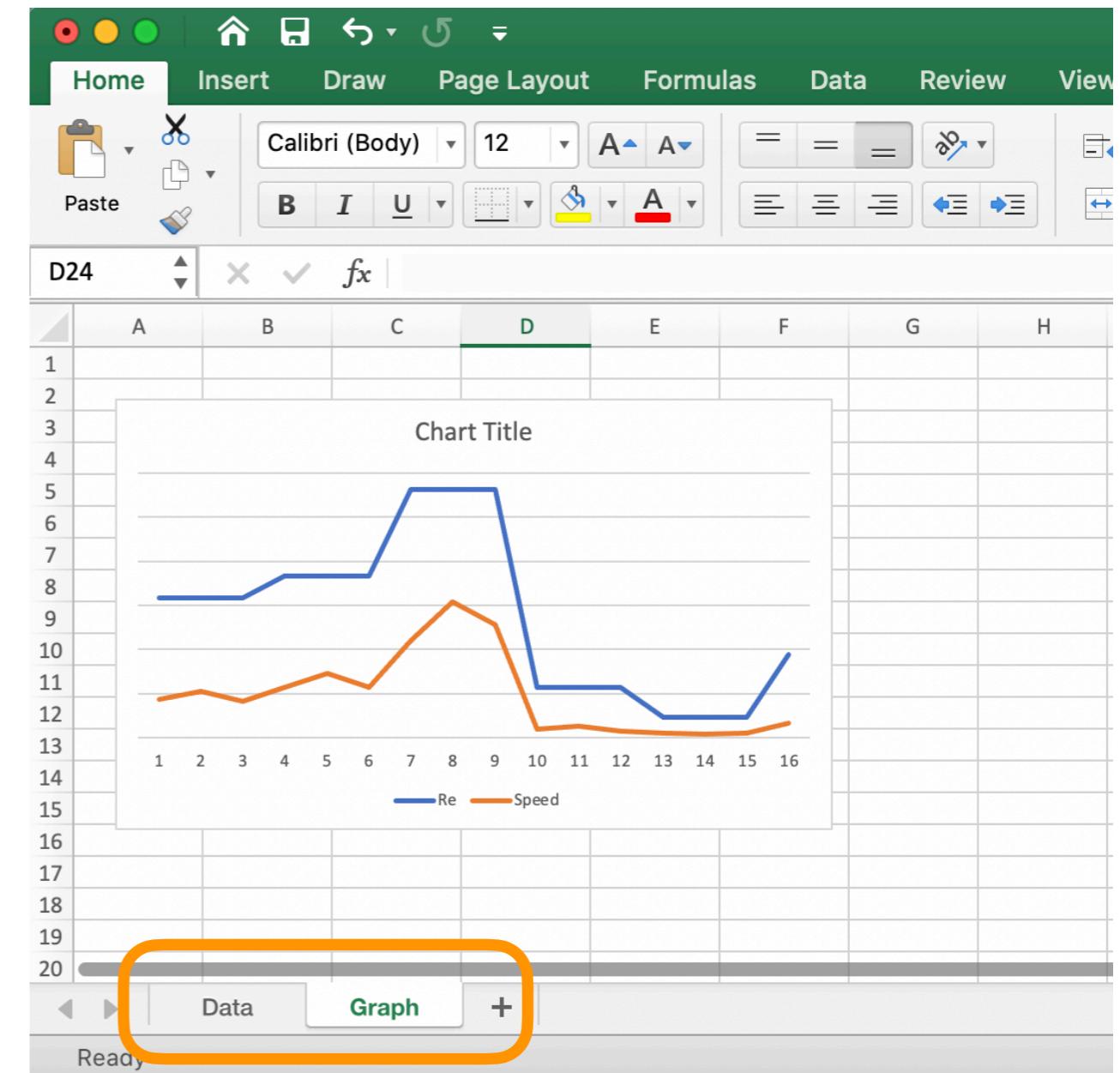
# Using Spreadsheets Safely

- Put objects like charts and graphs on a separate sheet from the data.

# DON'T



**DO**



# Using Spreadsheets Safely

- Only put one data set per sheet.

# DON'T



**DO**



# Using Spreadsheets Safely

- Use plain text and avoid special characters and inserting things like pictures, clipart, and video.
- If you need to write notes, make a “comments” column and restrict text to that column. Don’t write comments in random cells!
- If a column should be numeric, make sure ALL data is numeric in that column.
- Any calculations should (ideally) go in another sheet.



# Check Your Understanding

**Import the data set in `tomatohaul2021.csv` as `tomato.data`. The first column should be row names, the dates should be characters, and there is a header.**

# Cleaning Data

- After importing, data often needs to be cleaned up.
  - During this process, data are put into the appropriate form for whatever calculations/analyses are needed.
  - Often quite frustrating because of inconsistencies in how the data were entered or coded, weird format issues, extra info in file types, etc.
  - Quite a few tools exist to help clean data.
- Tidyverse package tools:
  - **tidyverse** package
  - **janitor** package

# Cleaning Data: Standardizing Names

- A useful thing to do is to standardize names so they are easier to reference and use. Both row and column names could have issues.
- `clean_names()` in the `janitor` package is an easy way to do this.

```
```{r, include=TRUE}
# Note: data frame with new column names is not stored
clean_names(babycrab.data)
```

Description: df [25 x 16]

number	size_mm	log_size	sex	hair_diameter	hair_length
number	size..mm.	log.size	sex	Hair.diameter	Hair.length
1	12.700	2.205827	m	8.1000e-06	648
4	5.00	1.6094	old column names	300e-06	374
5	11.50	2.442347	m	7.4400e-06	484
6	28.00	3.332205	m	8.4868e-06	660
7	25.00	3.218876	f	8.0100e-06	543
8	15.00	2.708050	m	6.9950e-06	505
9	22.00	3.091042	f	8.8100e-06	552
10	4.90	1.589235	m	5.6900e-06	347


```

# Cleaning Data: Removing Empty Rows and Columns

- Removing completely empty rows and/or columns is a common problem.
- `remove_empty()` in the `janitor` package is an easy way to do this. Use `which` argument to specify if you want to remove rows, columns, or both.

```
```{r,include=TRUE}
# Note: new data frame is not stored
remove_empty(babycrab.data, which=c("rows","cols"))
````
```

Description: df [25 × 13]

Column number went from 16  
to 13, removing 3 empty  
columns

|    | number<br><int> | size..mm.<br><dbl> | log.size<br><dbl> | sex<br><fctr> | Hair.diameter<br><dbl> | Hair.length<br><int> | ▶ |
|----|-----------------|--------------------|-------------------|---------------|------------------------|----------------------|---|
| 1  | 1               | 27.00              | 3.295837          | m             | 8.1000e-06             | 648                  |   |
| 2  | 2               | 8.50               | 2.140066          | f             | 6.7300e-06             | 465                  |   |
| 3  | 3               | 11.00              | 2.397895          | f             | NA                     | NA                   |   |
| 4  | 4               | 5.00               | 1.609438          | f             | 5.6800e-06             | 374                  |   |
| 5  | 5               | 11.50              | 2.442347          | m             | 7.4400e-06             | 484                  |   |
| 6  | 6               | 28.00              | 3.332205          | m             | 8.4868e-06             | 660                  |   |
| 7  | 7               | 25.00              | 3.218876          | f             | 8.0100e-06             | 543                  |   |
| 8  | 8               | 15.00              | 2.708050          | m             | 6.9950e-06             | 505                  |   |
| 9  | 9               | 22.00              | 3.091042          | f             | 8.8100e-06             | 552                  |   |
| 10 | 10              | 4.90               | 1.589235          | m             | 5.6900e-06             | 347                  |   |

# Cleaning Data: Pipe Operator in Tidy

- Tidyverse has a helpful pipe command `%>%` which will allow you to run several functions at once! The syntax is the:

```
object.name %>%
 fun1() %>%
 fun2() %>%
```

- Example:

**Assigns the output to  
`babycrab.clean`**

**Note: you don't have to include  
`babycrab.data` as an argument in  
each function!**

```
```{r,include=TRUE}  
babycrab.clean <- babycrab.data %>%  
  clean_names() %>%  
  remove_empty(which=c("cols"))  
...``
```

**Cleans names
then removes empty columns**

Check Your Understanding

Use the pipe operator to create `babycrab.clean` object that takes `babycrab.data` and cleans it with each of the cleaning steps at once but in this order:

- 1. cleans column names**
- 2. removes empty rows and columns**
- 3. removes all rows with NA values**

Writing Data to a File

- If you wish to save a data set to a file, this is easy to do with `write.table()`

Example:

```
```{r, include=TRUE}
write.table(babycrab.clean, file = "data/babycrab-clean.csv", sep = ",")
```

object you  
want to write  
to file

file name  
(put in folder data/)

separator  
is a  
comma

- `write.table()` options:

`write.csv()` will  
set `sep=","!`

**sep (character)**: type of separator for columns

**append (logical)**: do you want to add to the end of an existing file?

**row.names (logical/character)**: T/F indicating whether row names should be printed, or a vector of row names the same length as the number of rows.

**col.names (logical/character)**: similar to `row.names` but for columns

# Check Your Understanding

**Pick a data frame from `data()` and save it as a csv file in the `data/` folder.**

# Action Items

- 1. Complete Assignment 2.2 using R Markdown.**
  
- 2. Read Davies Chapter 8 and Chang Chapters 1-2 for next time.**