Lecture 07 – Grammar of Graphics

Learning Objectives:

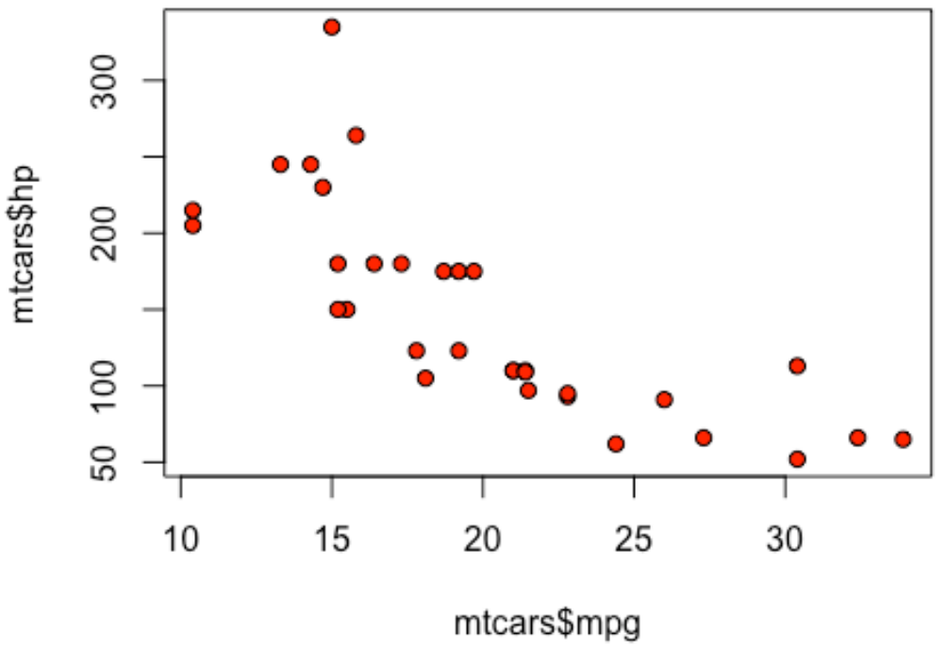
3.3 Learn the basics of `ggplot2`.

The Plotting Systems of R

- Base graphics and **`grid`**

```
plot(x = mtcars$mpg, y = mtcars$hp,
     col = "black", bg = "red",
     pch = 21)
```

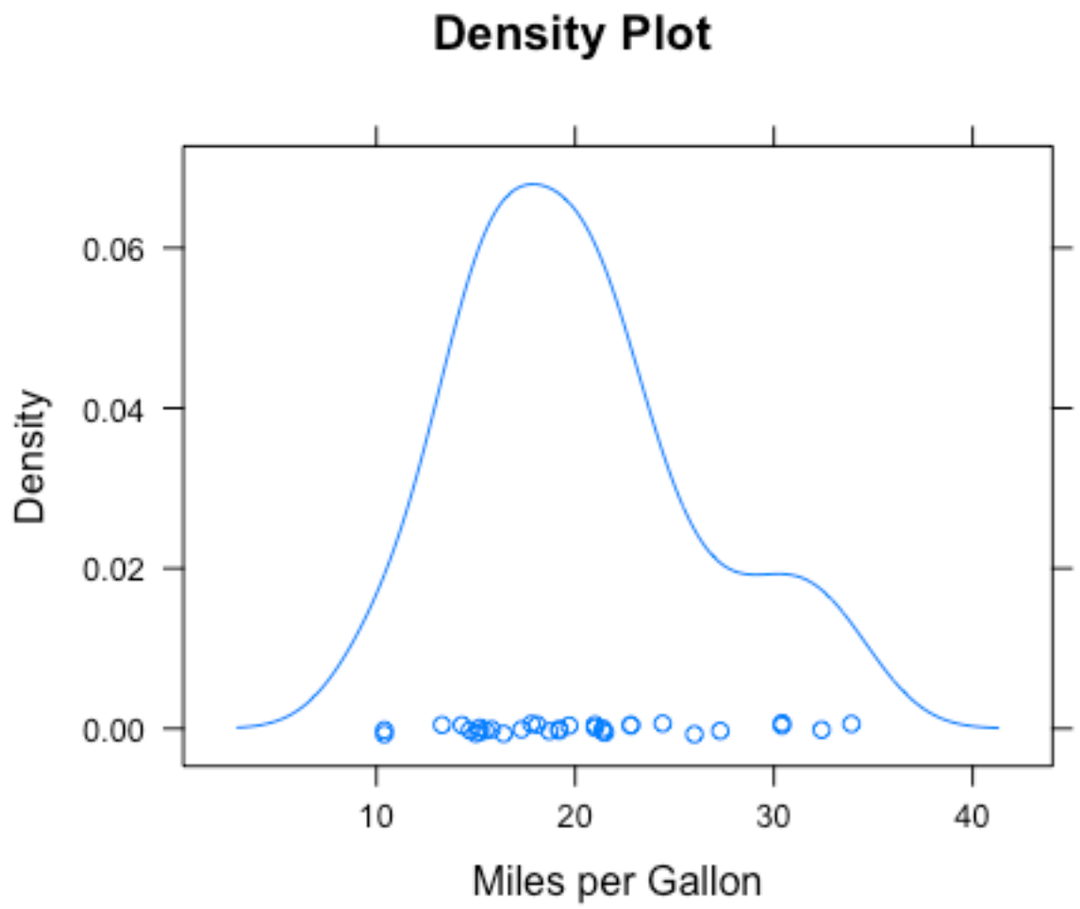
Benefits:
- simple and quick
- handles a variety of data types
- not many background calculations



- The Trellis system in **`lattice`**

```
densityplot(~mpg,
            main="Density Plot",
            xlab="Miles per Gallon")
```

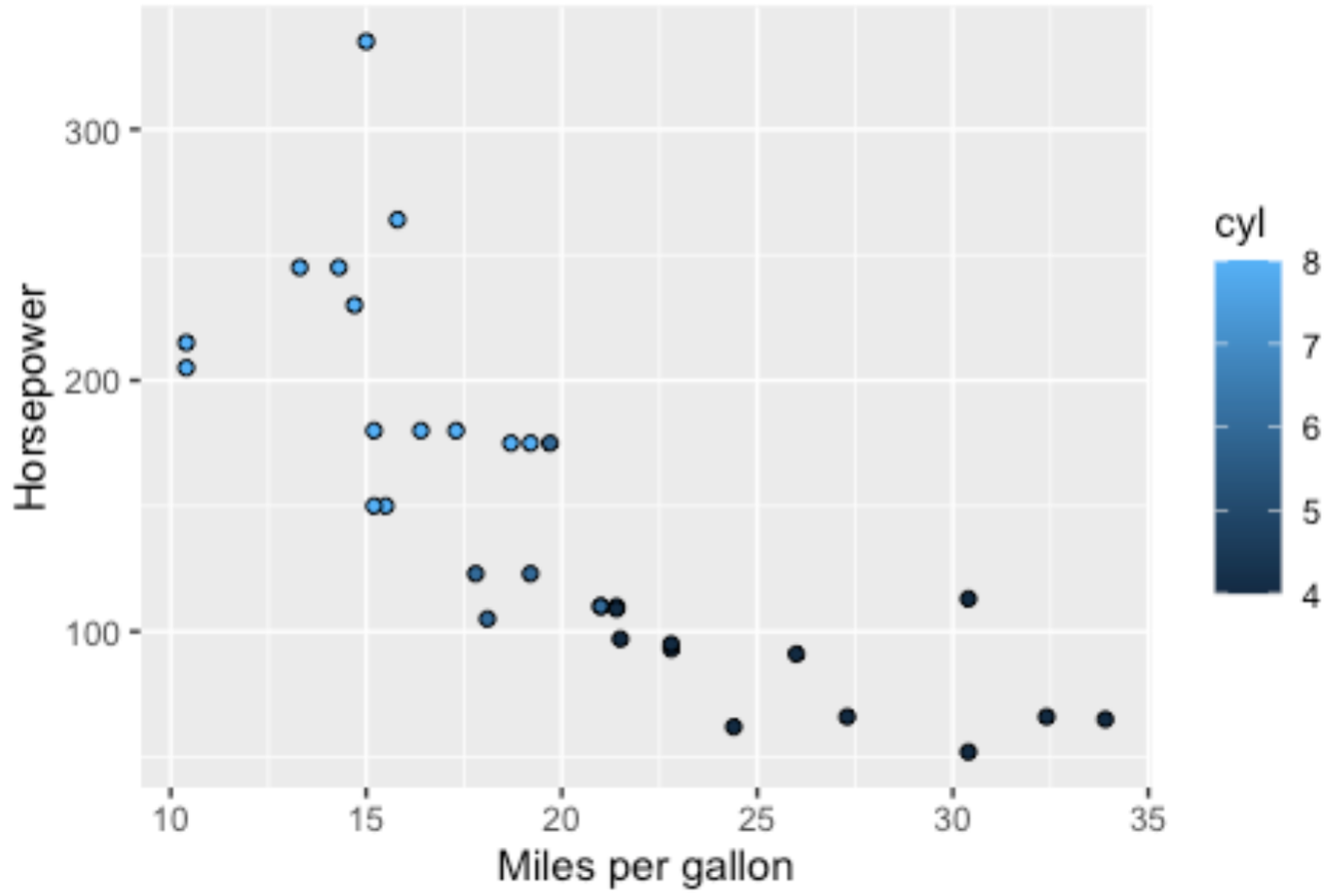
Benefits:
- quick
- many specialized stats plots
- not too many background calculations



The Plotting Systems of R

- Grammar of Graphics `ggplot2`

```
ggplot(mtcars, aes(x=mpg,y=hp,fill=cyl)) +
  geom_point(pch=21, size=2) +
  xlab("Miles per gallon") + ylab("Horsepower")
```



Benefits:

- **beautiful visualizations**
- **many specialized plots**
- **consistent syntax**
- **have full control over aesthetics**

Drawbacks:

- **lots of background calculations**
- **slow**
- **only uses data frames in long format**

Grammar of Graphics: terminology

Data

	mpg	cyl	disp	hp
Mazda RX4	21.0	6	160.0	110
Mazda RX4 Wag	21.0	6	160.0	110
Datsun 710	22.8	4	108.0	93
Hornet 4 Drive	21.4	6	258.0	110
Hornet Sportabout	18.7	8	360.0	175
Valiant	18.1	6	225.0	105
Duster 360	14.3	8	360.0	245
Merc 240D	24.4	4	146.7	62
Merc 230	22.8	4	140.8	95

Mapping

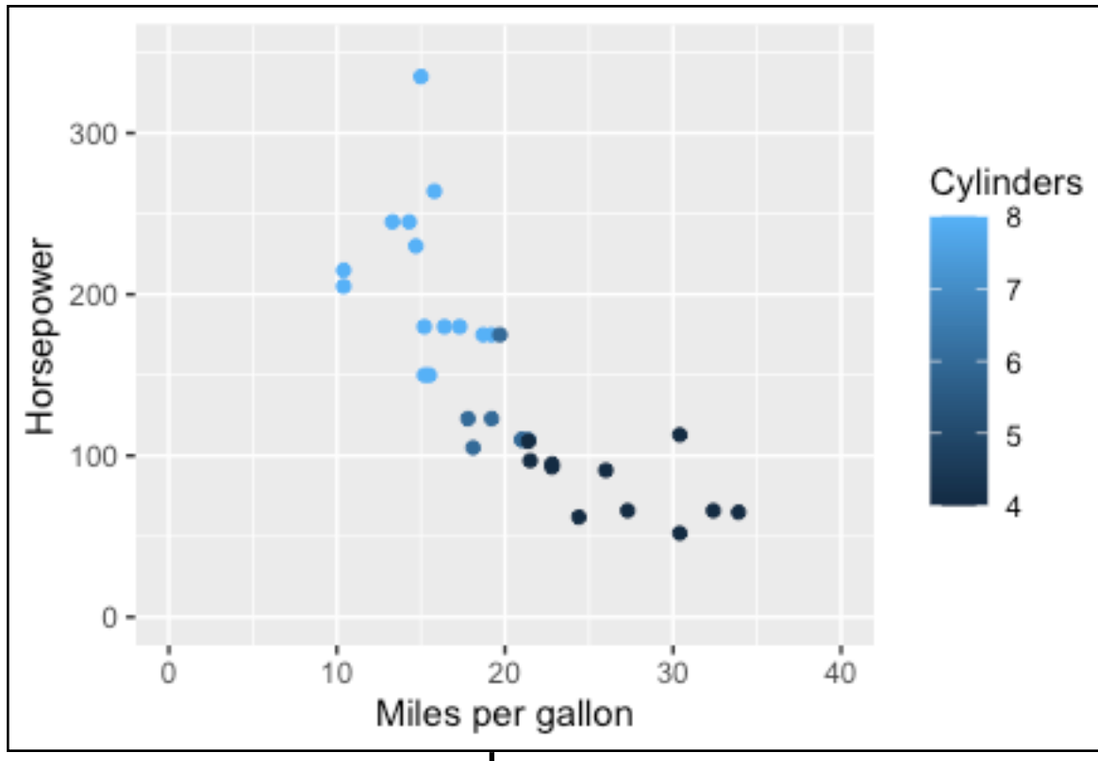
Aesthetic attributes

x

y

color/fill

Geometric object




```
ggplot( mtcars,        aes( x = mpg,     )      + geom_point()
                            y = hp,
                            fill = cyl)
```

Other stuff:

Scales

```
+ xlim(0, 40)
```

Guides

```
+ labs(color="Cylinders")
```

Check Your Understanding

Use the `Orange` data set to create a scatterplot with `ggplot2` of `circumference` versus tree `age`.

Data: wide versus long format

- To reap many of the benefits of the **`ggplot`** package, you may need to reshape your data set.

Wide: Columns represent different measurements

Seed	Year 3	Year 5	Year 10	Year 15	Year 20	Year 25
301	4.51	10.89	28.72	41.75	52.70	60.92
303	4.55	10.92	29.07	42.83	53.88	63.39
305	4.79	11.37	30.21	44.40	52.82	64.10
307	4.81	11.20	28.66	41.66	53.31	63.05

Long: Each row is a unique observation

Seed	Age	Height
301	3	4.51
301	5	10.89
301	10	28.72
301	15	41.74
301	20	52.70
301	25	60.92
303	3	4.55
303	5	10.92
303	10	29.07
303	15	42.38
303	20	53.88
303	25	63.39

Data: wide versus long format

Dataset: `Loblolly` **Package:** `tidyr`

Wide: Columns represent different measurements

Seed	Year 3	Year 5	Year 10	Year 15	Year 20	Year 25
301	4.51	10.89	28.72	41.75	52.70	60.92
303	4.55	10.92	29.07	42.83	53.88	63.39
305	4.79	11.37	30.21	44.40	52.82	64.10
307	4.81	11.20	28.66	41.66	53.31	63.05

```
Loblolly.wide <- pivot_wider(Loblolly,
                             names_from = age,
                             values_from = height)
```

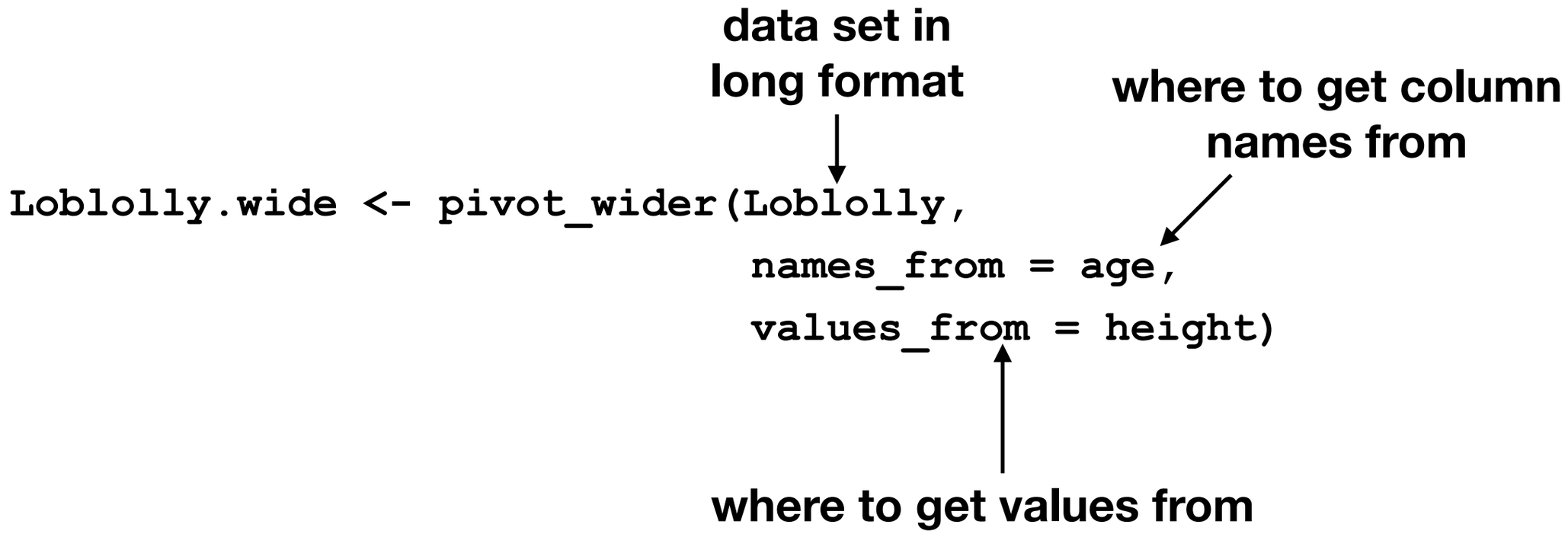
```
Loblolly.long <-
pivot_longer(Loblolly.wide, id.vars =
                      "Seed")
```

Long: Each row is a unique observation

Seed	Age	Height
301	3	4.51
301	5	10.89
301	10	28.72
301	15	41.74
301	20	52.70
301	25	60.92
303	3	4.55
303	5	10.92
303	10	29.07
303	15	42.38
303	20	53.88
303	25	63.39

Data: wide versus long format

- The **`pivot_wider()`** function: long to wide

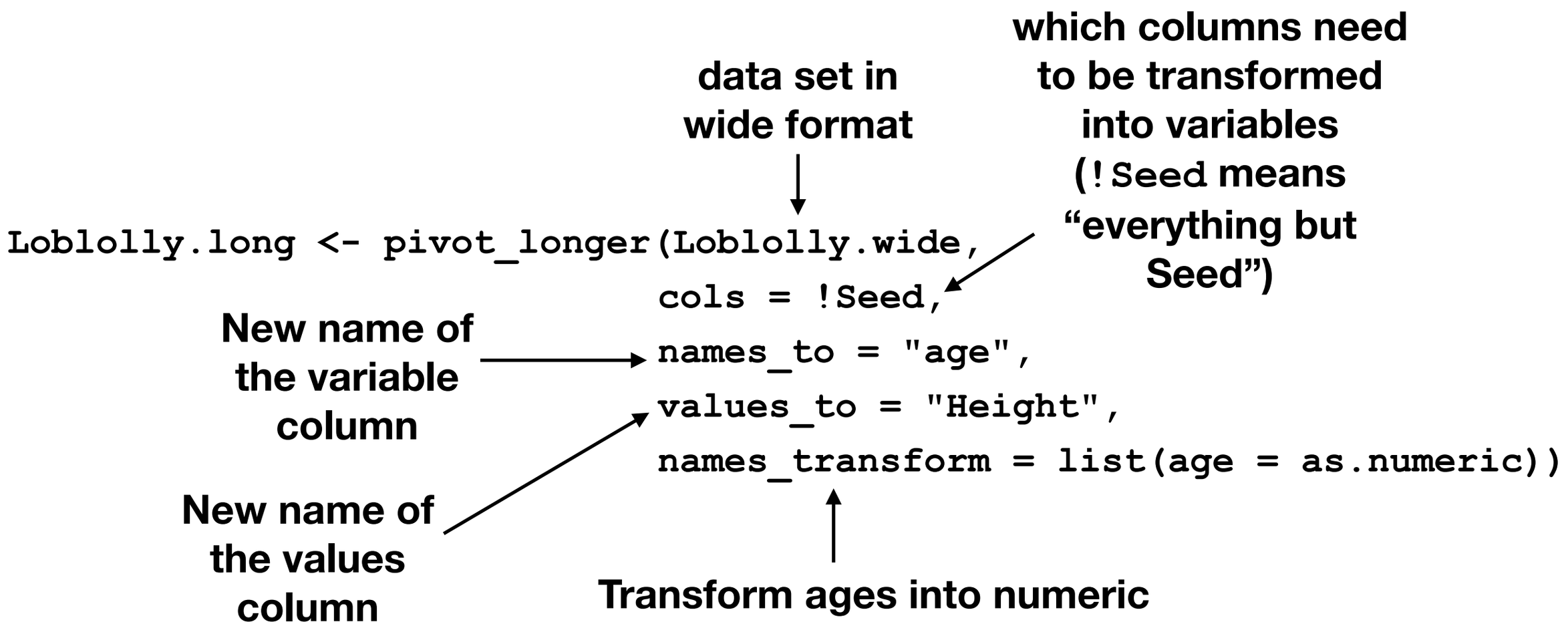


- The new columns are named based on **`names_from`**

- The cell values are filled from **`values_from`**

Data: wide versus long format

- The **`pivot_longer()`** function: wide to long



- Pick the columns that should be data values (**`cols`**). The column names will become variables in long format. The variable column will be named by **`names_to`**.

- Pick the name of that new column where the values will be stored (**`values_to`**).

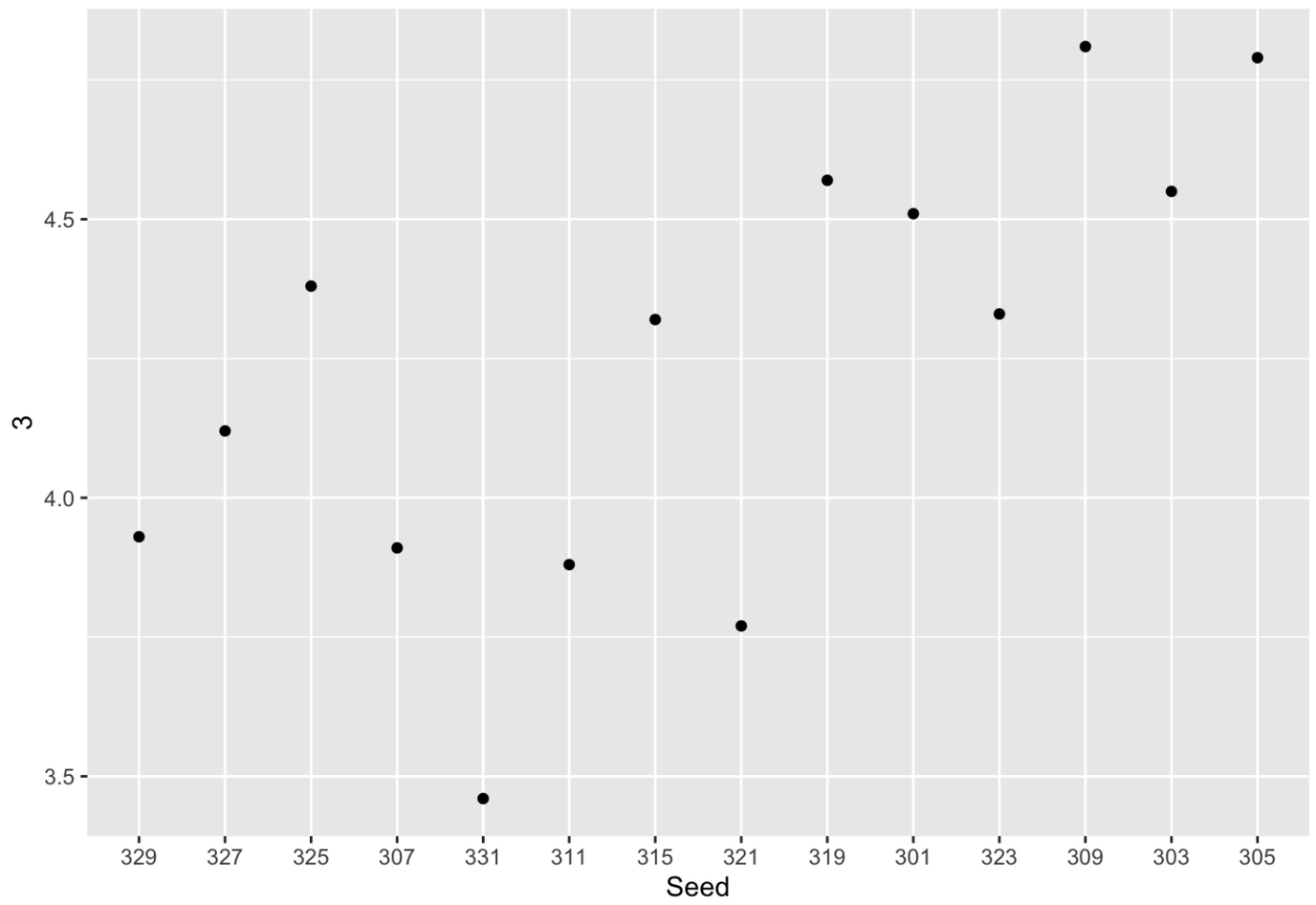
Plotting: wide format

- Plotting in the wide format limits your options. You are forced to choose which columns to map, which limits you to the ways in which you can plot!

column options for `Loblolly.wide`:

	Seed	3	5	10	15	20	25
1	329	3.93	9.34	26.08	37.79	48.31	56.43
2	327	4.12	9.92	26.54	37.82	48.43	56.81

```
ggplot(Loblolly.wide,
          aes(x = Seed,
              y = `3`))+
    geom_point()
```



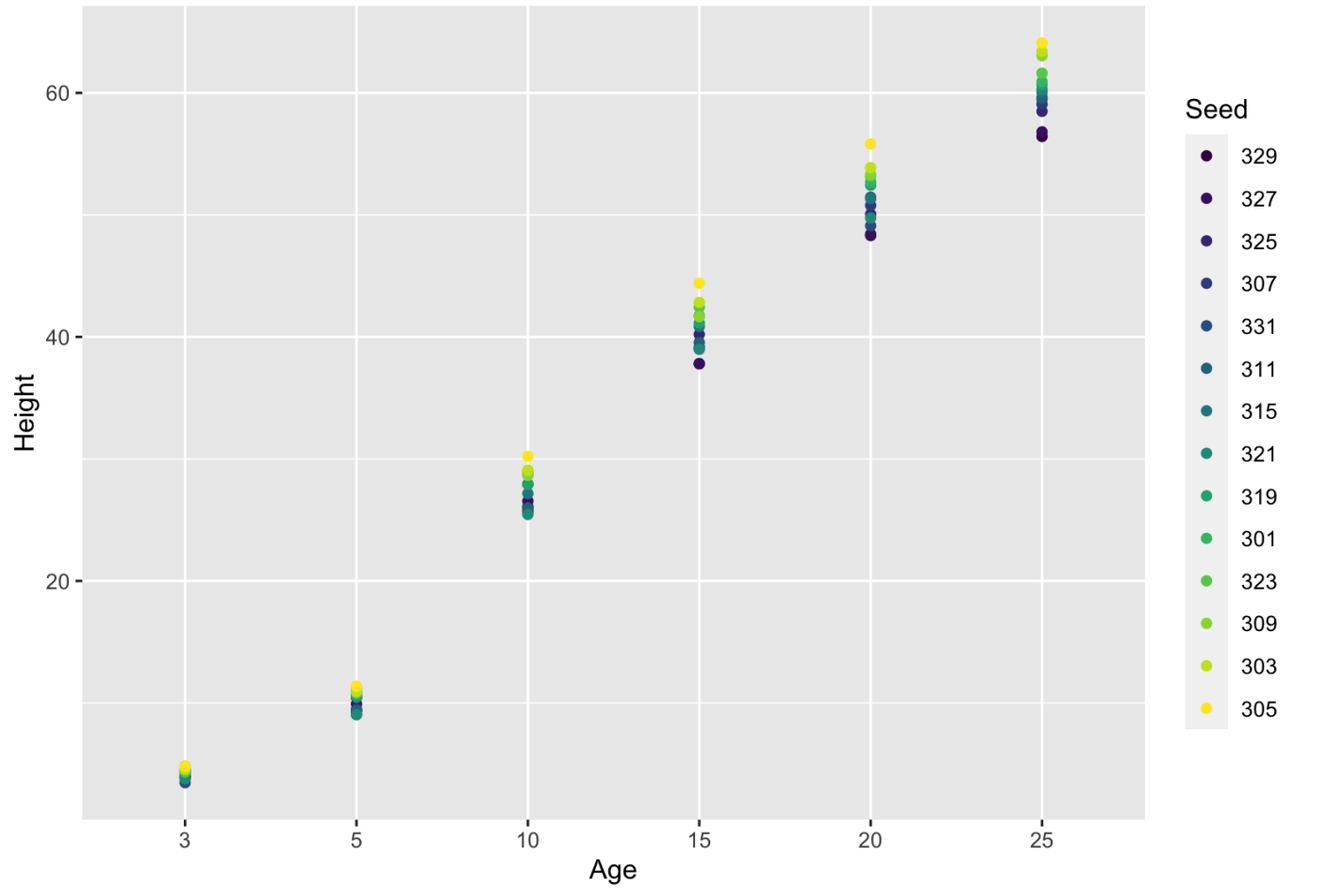
Plotting: long format

- The long format doesn't suffer from this same issue. You have appropriate options for plotting and mapping aesthetics.

column options for `Loblolly.long`:

	Seed	Age	Height
1	329	3	3.93
2	327	3	4.12

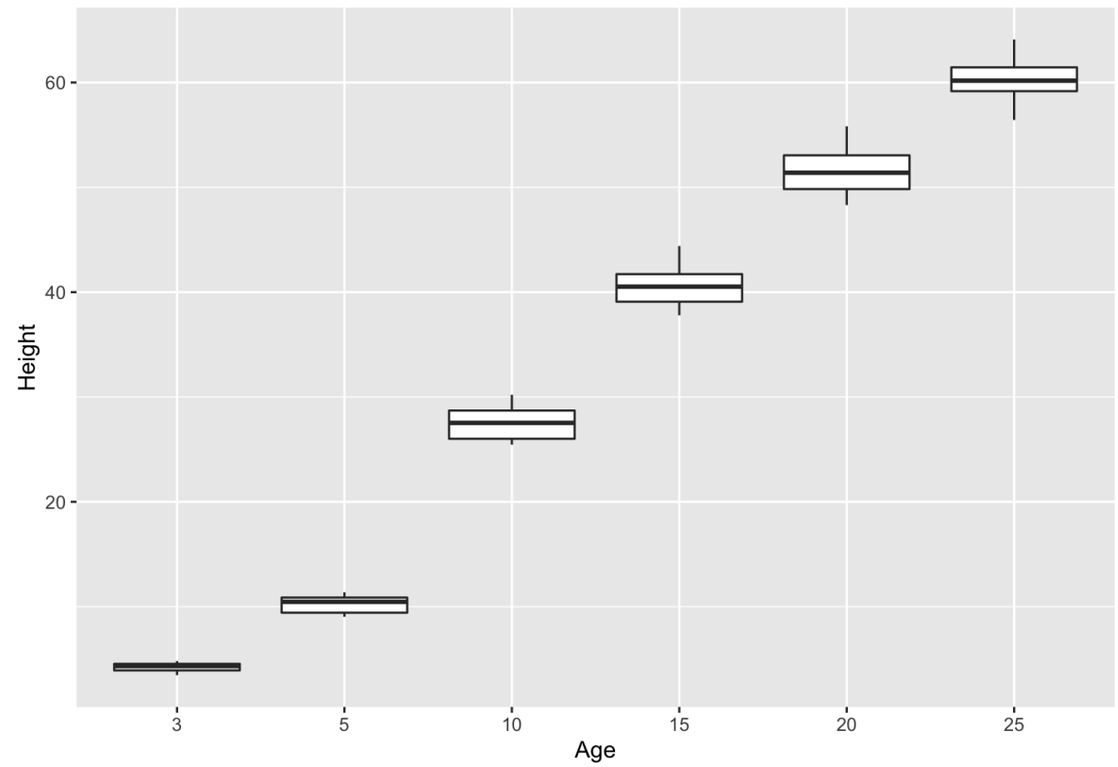
```
ggplot(Loblolly.long,
       aes(x = age,
           y = Height,
           color = Seed)) +
  geom_point(pch = 19)
```



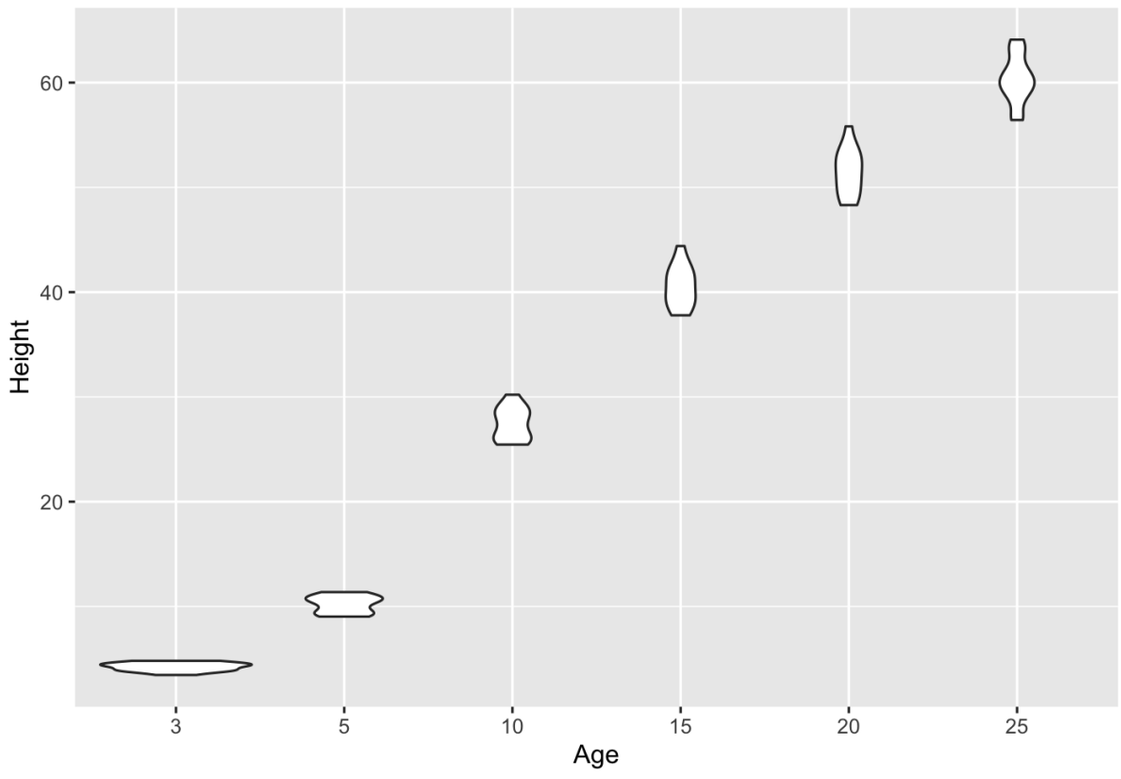
Plotting: long format

- You can push many calculations for plot types into **`ggplot`** so as not to worry about doing them yourself!

```
ggplot(Loblolly.long,
       aes(x = factor(age),
           y = Height)) +
    geom_boxplot()
```




```
ggplot(Loblolly.long,
       aes(x = factor(age),
           y = Height)) +
    geom_violin()
```



Check Your Understanding

1. **Reshape the `fish_encounters` data set (in the `tidyr` package) into a wide format.**

2. **Reshape the `relig_income` data set (in the `tidyr` package) into a long format.**

Changing Basic Plot Attributes

- Changing basic attributes:

Base plot: `p <- ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point()`

- Plot labels and titles:

X-axis label: `p + xlab("Miles per Gallon")`
Y-axis label: `p + ylab("Horsepower")`
Plot title: `p + ggtitle("Horsepower vs MPG from MtCARS")`

- Axis scales:

Change x limits: `p + xlims(min, max)`
Change y limits: `p + ylims(min, max)`
Change color limits: `p + lims(color = c(newlimits))`

- Formatting of plot area and text:

There are lots of arguments here!

Most items: `p + theme()`
Minimal theme: `p + theme_minimal()`

Changing Aesthetic Attributes

If independent of mapping, you will change them in the geom!

- Changing aesthetic attributes:

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point()
```

- Point size, shape, color, and fill:

(2 is 2x size of default points)

Point size: `... + geom_point(size = 2)`

Point shape: `... + geom_point(shape = 19)`

or

`... + geom_point(pch = 19)`

Point color: `... + geom_point(color = "red")`

Point fill:
```
... + geom_point(pch = 21,
                 color = "black", fill = "red")
```

Check Your Understanding

Add to the plot in your last check your understanding by:

1. **Adding axis labels and a title.**
2. **Changing the color, shape, and size of the points.**
3. **Rotating the x-axis labels by 45 degrees.**

Grouping and Aesthetic Values

- It's easy to change group characteristics using **`aes()`**, include the specific characteristic you want (**`color`**, **`fill`**, **`shape`**) or use **`group`** and define those later.

Assign colors based on `cyl`:

```
ggplot(mtcars, aes(x = mpg, y = hp,
                   color = factor(cyl)) +
       geom_point()
```

Assign fill based on `cyl`:

```
ggplot(mtcars, aes(x = mpg, y = hp,
                   fill = factor(cyl)) +
       geom_point(pch = 21)
```

Assign shape based on `cyl`:

```
ggplot(mtcars, aes(x = mpg, y = hp,
                   shape = factor(cyl)) +
       geom_point()
```

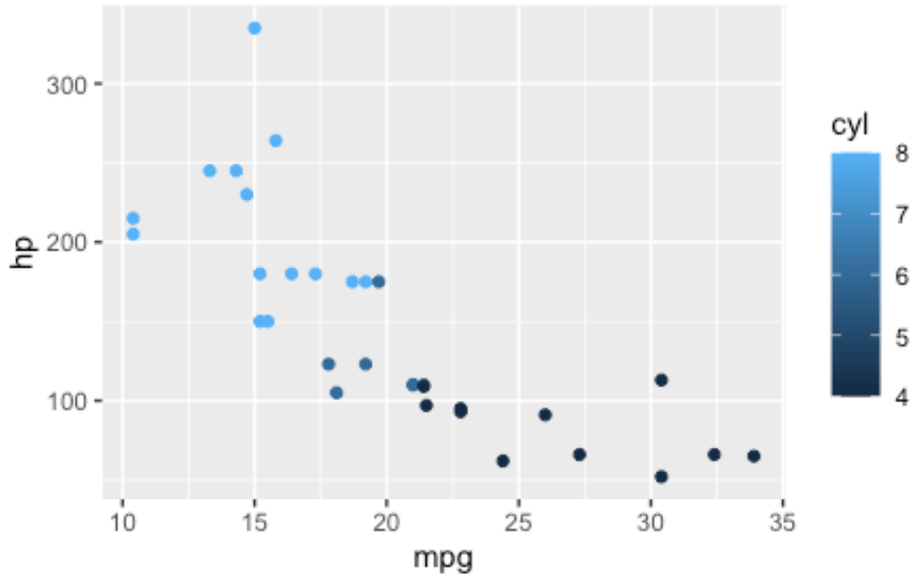
Note: using `factor(cyl)` will help out these plots!

Mapping aesthetics based on data

- Mapping aesthetics that change with data must be done with **`aes()`**

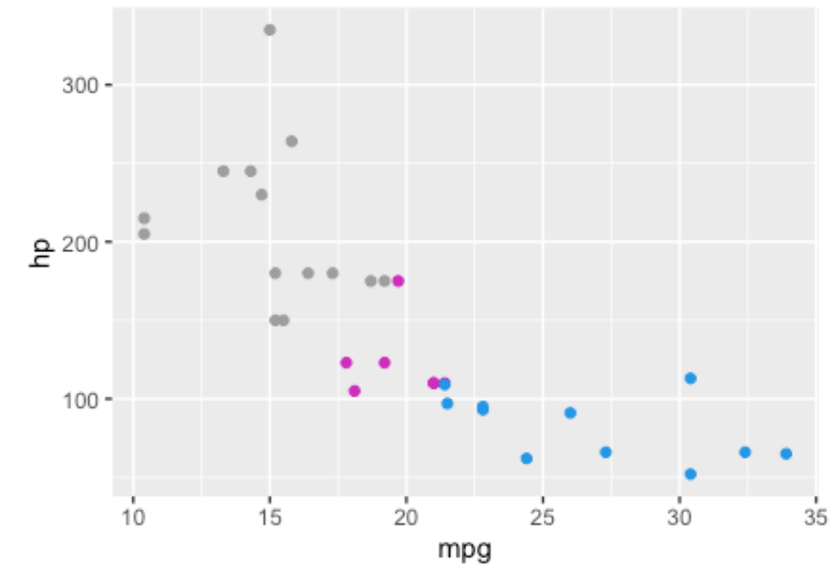
```
ggplot(mtcars,
       aes(x = mpg, y = hp,
           color = cyl)) +
  geom_point()
```

Maps correctly




```
ggplot(mtcars,
       aes(x = mpg, y = hp)) +
  geom_point(color = cyl)
```

Maps incorrectly



- Changing non-mapped aesthetics can be done anywhere

```
ggplot(mtcars,
       aes(x = mpg, y = hp)) +
  geom_point(aes(color = cyl))
```

Maps correctly

Adding data from other sets to existing plots

- Adding data manually from other sets is very clunky and should be avoided whenever possible.

```
p2 <- ggplot(Loblolly.wide, aes(x = Seed, y = `3`)) +
         geom_point()
```

Add another column from `Loblolly.wide` to the graph as red dots:

```
p2 + geom_point(data = Loblolly.wide,
                mapping = aes(x = Seed, y = `5`),
                color = "red")
```

- the data set must be specified in the geom using **`data=`**
- mapping must be specified in the geom using **`mapping=`** and **`aes()`**
- other attributes have to be set independently in the geom
- it is ugly and won't give you a correct legend

Check Your Understanding

Add to the plot in your last check your understanding by:

1. **Coloring the plot points by Tree**
2. **Adding lines in the link data from the individual trees together**

Geometric objects

Scatter plot – `geom_point()`

Line plot – `geom_line()`

Box plot – `geom_boxplot()`

Violin plot – `geom_violin()`

Bar plot – `geom_bar()`

Contour plot – `geom_contour()`

Density plot – `geom_density()`

Plot a map – `geom_map()`

Rectangles – `geom_raster()`
`geom_tile()`

Quantile-quantile plot – `geom_qq_line()`

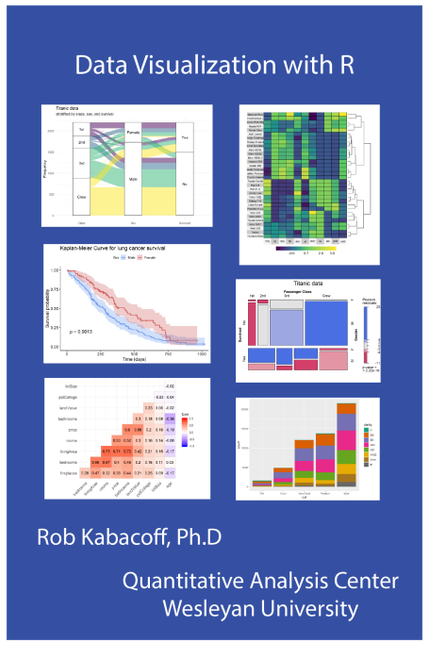
Stacked dot plot – `geom_dotplot()`

Histogram – `geom_histogram()`

Choose the correct plot for your data!

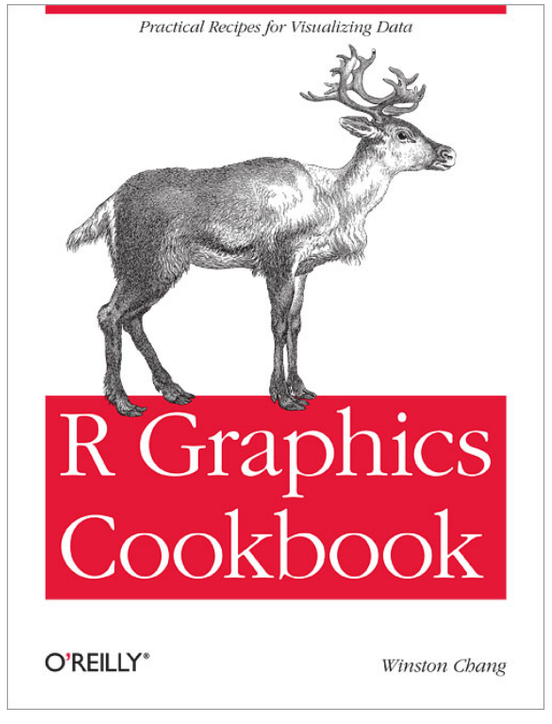
Excellent Resources for ggplot2

Tidyverse Reference Guide – https://ggplot2.tidyverse.org/reference/



Data Visualization with R – https://rkabacoff.github.io/datavis/

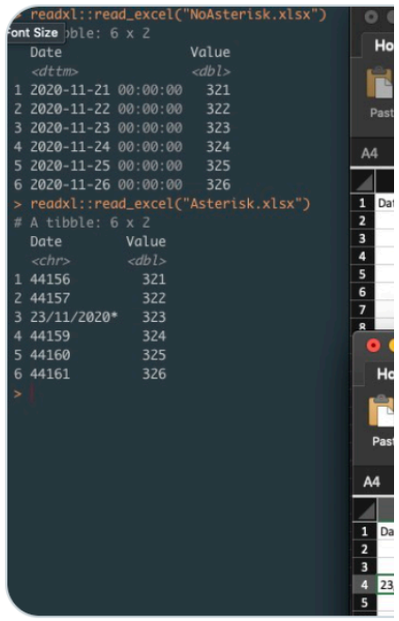
The R Graphics Cookbook – https://r-graphics.org/



#OtherPeoplesData

- 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!

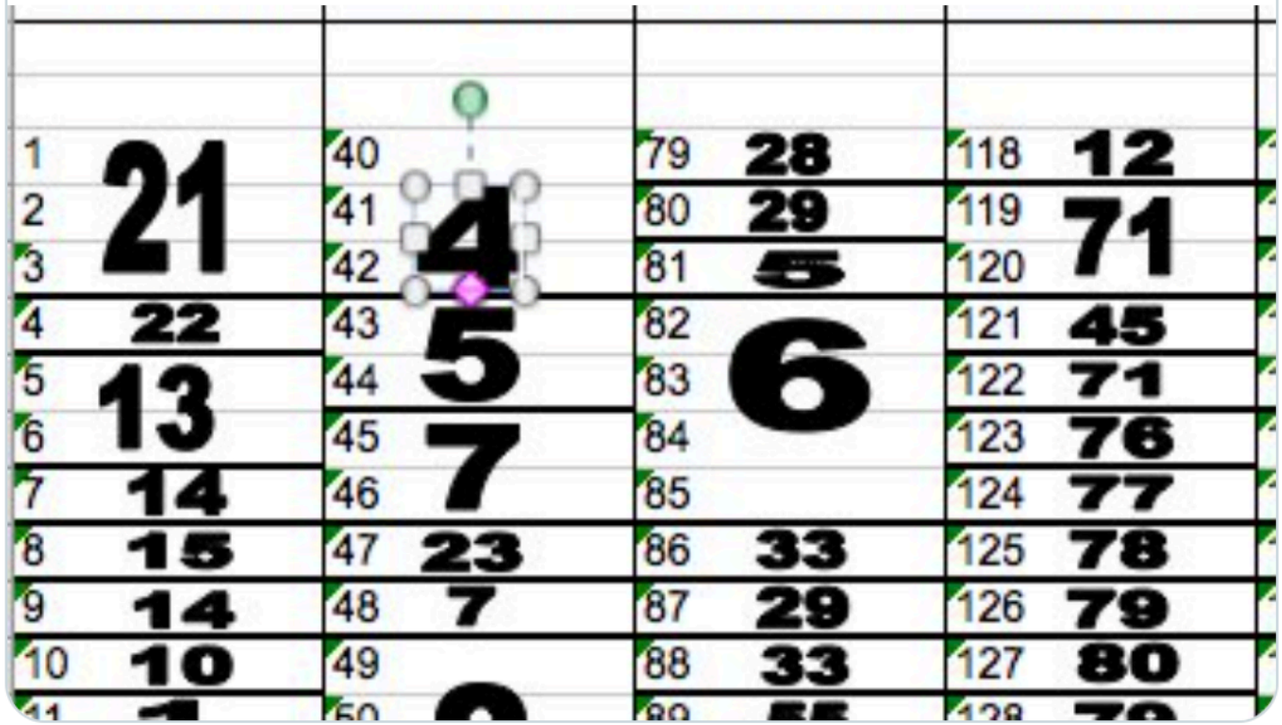
Jonathan Carroll @carroll_
Today's fresh #OtherPeopl
It turns out that if someone
an asterisk it may entirely b
surprise Excel still thinks it'



2 1

Dr Rachel Hale, a vaccinated worm blob! @SeafloorScien... · Sep 14
This is the worst example of #OtherPeoplesData I've ever seen

OpenElections @openelex · 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.



5 2 15

4
eoplesdata as a 276 page set of
13

Mar 29, 2017
bler is BLPW. Not Bkpw, or bcpw,
sdata
9

18
thought that typing "<1" rather
erPeoplesData Many numeric
is!
8

Importing Data

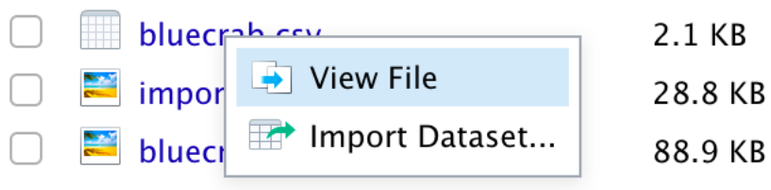
Be sure to download the `data.zip` file in Lecture Notes and unzip it!

- **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:



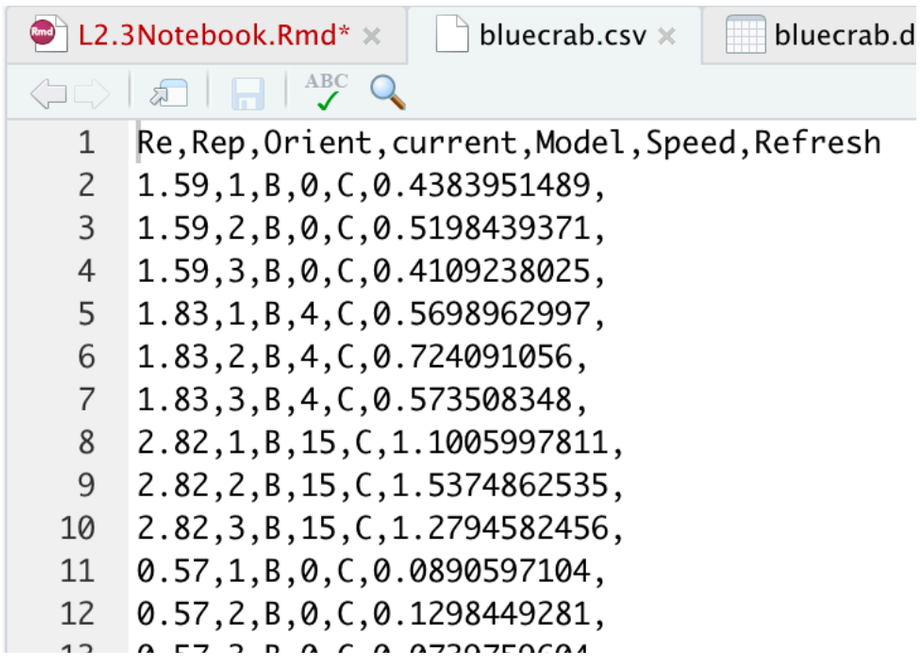
L2.3Notebook.Rmd* | bluecrab.data

Filter

	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

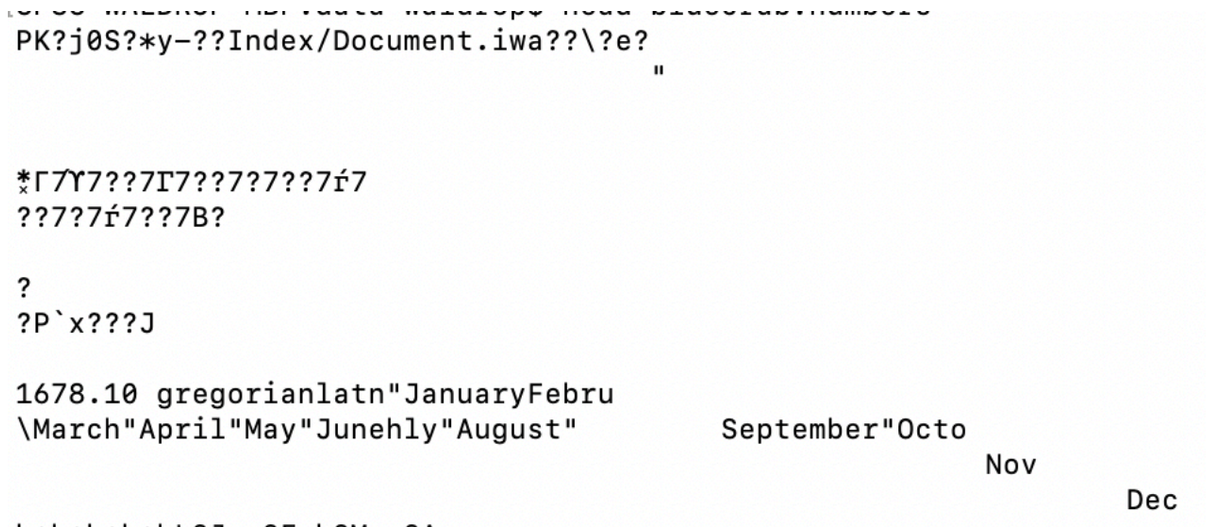
- **Text files**: data files consisting of plain text with values separated by (delineated with) a specific character.



- **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



Microsoft Excel

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

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

```
Re,Rep,Orient,current,Model,Speed,Refresh
1.59,1,B,0,C,0.4383951489,
1.59,2,B,0,C,0.5198439371,
1.59,3,B,0,C,0.4109238025,
1.83,1,B,4,C,0.5698962997,
1.83,2,B,4,C,0.724091056,
1.83,3,B,4,C,0.573508348,
2.82,1,B,15,C,1.1005997811,
2.82,2,B,15,C,1.5374862535,
2.82,3,B,15,C,1.2794582456,
0.57,1,B,0,C,0.0890597104,
0.57,2,B,0,C,0.1298449281,
```

L2.3Notebook.Rmd* | bluecrab.data

Filter

	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

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???

L2.3Notebook.Rmd* | babycrab.data | babycrab1.csv

Filter

	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

L2.3Notebook.Rmd* | babycrab.data | babycrab1.csv

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,,,
```

Importing Data: Not-so-perfect

- How can we improve this import?

First row looks pretty useless.

L2.3Notebook.Rmd* | babycrab.data | babycrab1.csv

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()`**:

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

A tibble: 72 x 7

Re <dbl>	Rep <dbl>	Orient <chr>	current <dbl>	Model <chr>	Speed <dbl>	Refresh <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

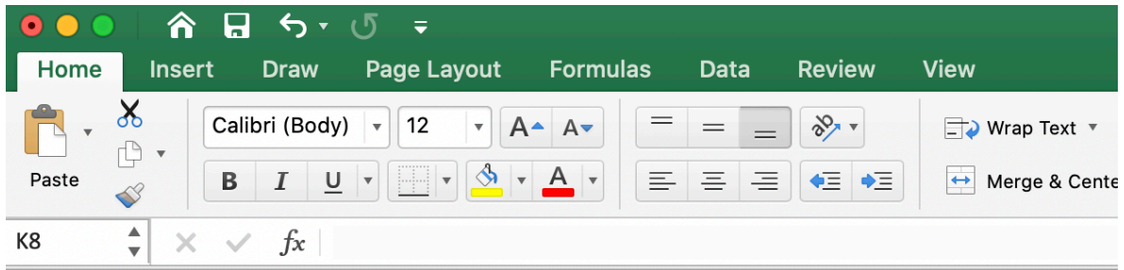
- **Consider exporting your spreadsheet as a text file (e.g. CSV) for use in R!**

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).

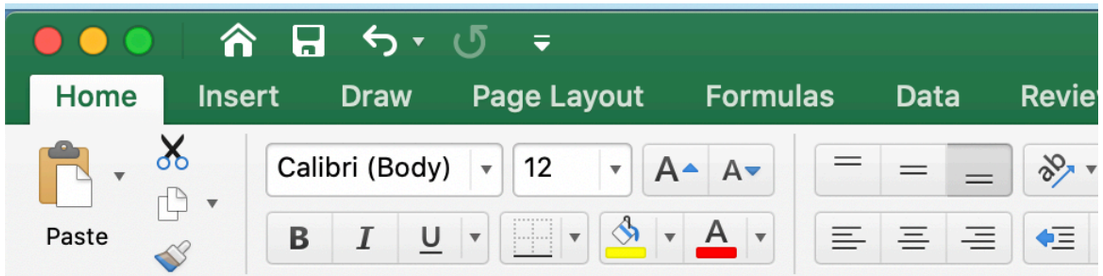
  - Align your data in the upper left, leaving no extra columns on either side.

DON'T 🙅‍♀️



	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6			Blue Crab Data						
7									
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 👍🏻



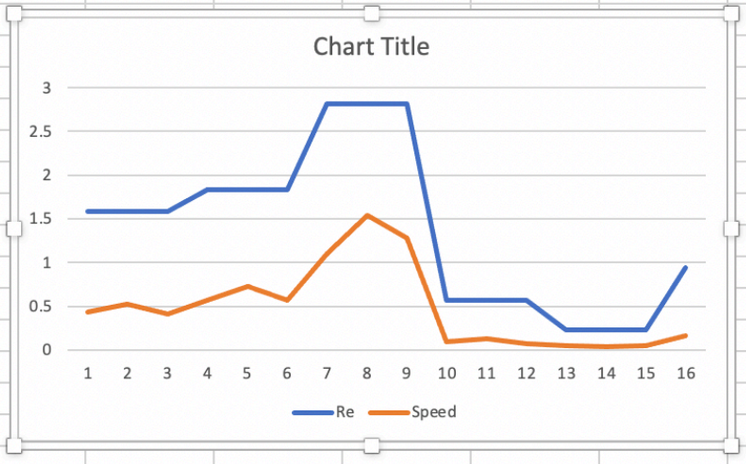
	A	B	C	D	E	F	G
1	Re	Rep	Orient	current	Model	Speed	Refresh
2	1.59	1	B	0	C	0.43839515	
3	1.59	2	B	0	C	0.51984394	
4	1.59	3	B	0	C	0.4109238	
5	1.83	1	B	4	C	0.5698963	
6	1.83	2	B	4	C	0.72409106	
7	1.83	3	B	4	C	0.57350835	
8	2.82	1	B	15	C	1.10059978	
9	2.82	2	B	15	C	1.53748625	
10	2.82	3	B	15	C	1.27945825	
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

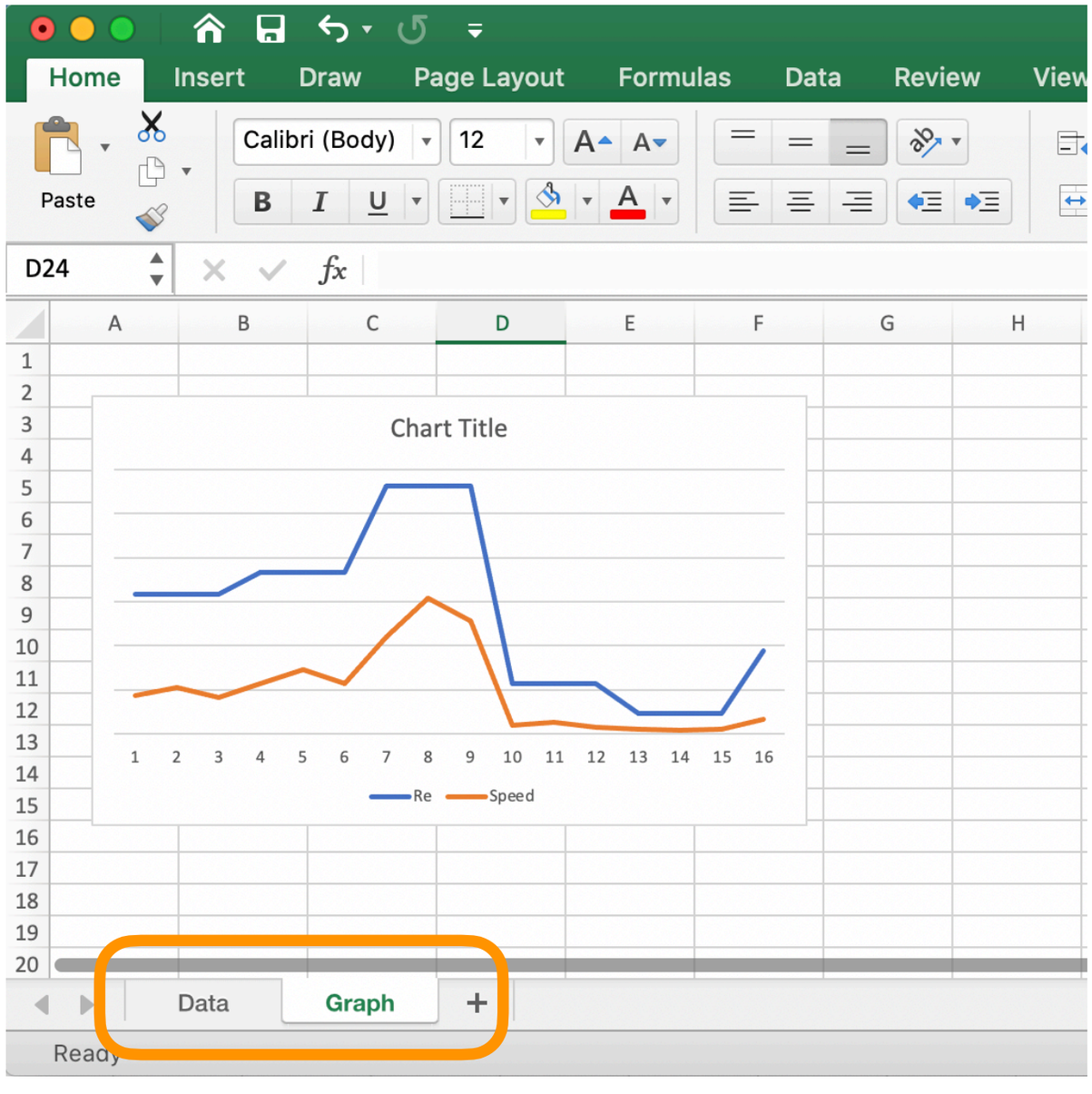
- Put objects like charts and graphs on a separate sheet from the data.

DON'T 🤦🏼‍♀️

	A	B	C	D	E	F	G	H	I
6			Blue Crab Data						
7									
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	



DO 👍🏻



Using Spreadsheets Safely

- Only put one data set per sheet.

DON'T 🙅

J6 | Baby crab data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
6			Blue Crab Data							Baby crab data							
8			Re	Rep	Orient	current	Model	Speed		number	size (mm)	log size	sex	Hair diameter	Hair length	Cuticle thickness	
9			1.59	1	B	0	C	0.43839515		1	27	3.29583687	m	8.10E-06	648	7.68E-07	
10			1.59	2	B	0	C	0.51984394		2	8.5	2.14006616	f	6.73E-06	465	7.19E-07	
11			1.59	3	B	0	C	0.4109238		3	11	2.39789527	f				
12			1.83	1	B	4	C	0.5698963		4	5	1.60943791	f	5.68E-06	374		
13			1.83	2	B	4	C	0.72409106		5	11.5	2.44234704	m	7.44E-06	484	8.74E-07	
14			1.83	3	B	4	C	0.57350835		6	28	3.33220451	m	8.49E-06	660	1.01E-06	
15			2.82	1	B	15	C	1.10059978		7	25	3.21887582	f	8.01E-06	543	1.01E-06	
16			2.82	2	B	15	C	1.53748625		8	15	2.7080502	m	7.00E-06	505	7.04E-07	
17			2.82	3	B	15	C	1.27945825		9	22	3.09104245	f	8.81E-06	552		
18			0.57	1	B	0	C	0.08905971		10	4.9	1.58923521	m	5.69E-06	347	6.39E-07	
19			0.57	2	B	0	C	0.12984493		11	5.85	1.76644166	m	5.16E-06	356		
20			0.57	3	B	0	C	0.07397596		12	7.12	1.96290773	m	5.53E-06	400	6.31E-07	
21			0.23	1	B	4	C	0.04501641		13	16	2.77258872	m	8.38E-06	435	7.91E-07	
22			0.23	2	B	4	C	0.03796072		14	13	2.56494936	f	7.75E-06	468	5.93E-07	
23			0.23	3	B	4	C	0.04804528									
24			0.94	1	B	15	C	0.16302115									

DO 👍

D19

	A	B	C	D	E	F	G	H	I
1	number	size (mm)	log size	sex	Hair diameter	Hair length	Cuticle thickness		
2	1	27	3.29583687	m	8.10E-06	648	7.68E-07		
3	2	8.5	2.14006616	f	6.73E-06	465	7.19E-07		
4	3	11	2.39789527	f					
5	4	5	1.60943791	f	5.68E-06	374			
6	5	11.5	2.44234704	m	7.44E-06	484	8.74E-07		
7	6	28	3.33220451	m	8.49E-06	660	1.01E-06		
8	7	25	3.21887582	f	8.01E-06	543	1.01E-06		
9	8	15	2.7080502	m	7.00E-06	505	7.04E-07		
10	9	22	3.09104245	f	8.81E-06	552			
11	10	4.9	1.58923521	m	5.69E-06	347	6.39E-07		
12	11	5.85	1.76644166	m	5.16E-06	356			
13	12	7.12	1.96290773	m	5.53E-06	400	6.31E-07		
14	13	16	2.77258872	m	8.38E-06	435	7.91E-07		
15	14	13	2.56494936	f	7.75E-06	468	5.93E-07		
16									
17									
18									

BlueCrabData | BabyCrabData | +

Ready

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.

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:
    - **`tidyr`** 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
# Note: data frame with new column names is not stored
clean_names(babycrab.data)
```

Description: df [25 × 16]

number <int>	size_mm <dbl>	log_size <dbl>	sex <fctr>	hair_diameter <dbl>	hair_length <int>
number <int>	size..mm. <dbl>	log.size <dbl>	sex <fctr>	Hair.diameter <dbl>	Hair.length <int>
4	5.00	1.6094		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

old column names

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

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

Description: df [25 × 13]

	number <int>	size..mm. <dbl>	log.size <dbl>	sex <fctr>	Hair.diameter <dbl>	Hair.length <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**

Additional Resources

https://www.stat.auckland.ac.nz/~ihaka/787/lectures-trellis.pdf –
The Trellis system in lattice (PDF lecture slides)

http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/ –
Converting between long and wide format with reshape2, tidyr, and base R