Working with tidy data in R, or: why starting tidy makes life in R palatable

Elizabeth Wickes, wickes1@illinois.edu, @elliewix

Getting started with data in R

As part of the workshop we'll be talking about R and doing a brief tour or RStudio. This handout is meant to capture the commands that were used and give you something to experiment with after the workshop. Starting out with tidy or tidyish data will allow you to (nearly...) immediately use data exploration tools.

Load the data

RStudio includes several tools to help you import data into your environment. You can find this on the upper right, in the "Environment" tab. Click "Import Dataset" and select the appropriate file type. You can use CSV for delimited plain text files, even if they aren't proper CSVs.

Select the file you'd like to import, and you'll be presented with a wizard to help you preview your data import options. You can play around with these options and view the changes in the preview box. All the defaults should work just fine with our example data file. Note that it is autofilling blank values with `NA` and does a decent job at detecting when a column is numerical/integer versus character. The wizard will have a box that is generating the import code as you make changes in the tool. You can click import or you can copy the text in.

```r
library(readr)

cleanedsurvey <- read_csv("~/Documents/Humanities-Data-Praxis/cleanedsurvey.csv")

View(cleanedsurvey)
```

Instead of a Pivot Table...

R has several commands for quickly putting together summary tables of your data. `xtabs()` will create a contingency table that sums either values or instances in categories. You will be using the `~` symbol to provide a formula. Items on the left will be summed and items on the right will be the groups. Use the `+` symbol to add more than one group.

```r
xtabs(~ Sex, data = cleanedsurvey) # count of how many male and female samples
```

```
## Sex
##  F  M
## 49 34
```

```r
xtabs(~ Sex + Species, data = cleanedsurvey) # count of how many male/female samples there are per spec
```

```
##    Species
## Sex DM DO DS OL OT PE PF
##   F 18 10 13  0  3  0  2
##   M 19  3  3  1  2  1  2
```

Say you need more than just tabulations of values, but you want to do some basic summary statistics as a method of exploration. You can use `summary()` over a single column, but we often care about how that value interacts with other factors. `aggregate()` is the tool for you. This uses a similar notation system, with the column you want to do the math stuff on the left of the ~ and the columns with the categories on right right, with additional columns connected via +.

```
aggregate(Weight ~ Sex, data = cleanedsurvey, FUN = mean) # mean weight by sex
```

```
##   Sex   Weight
## 1   F 65.51064
## 2   M 64.00000
```

```
aggregate(Weight ~ Sex + Species, data = cleanedsurvey, FUN = mean) # mean weight by sex and species
```

```
##    Sex Species    Weight
## 1    F      DM  39.00000
## 2    M      DM  42.06250
## 3    F      DO  41.90000
## 4    M      DO  48.33333
## 5    F      DS 124.00000
## 6    M      DS 134.66667
## 7    M      OL  35.00000
## 8    F      OT  24.66667
## 9    M      OT  25.00000
## 10   M      PE  22.00000
## 11   F      PF   7.00000
## 12   M      PF   7.50000
```

Graphics in R

`ggplot2` is a solid community standard in creating visualizations within R. We're going to step through some examples of how to use tidy data with `ggplot2`.

Load the library

You will first need to install the ggplot2 graphics package. Please use the `install.packages("ggplot2")` command to do so, and will only need to do this once. After it installs, you call `library()` with the module name, without quotes, to load the library.

```
library(ggplot2)
```

Make a basic histogram

This will be using ggplot2 to create a histogram. ggplot statements usually start with a `ggplot()` function with some basic declarations, then you use `+` to add graphic elements to the plot. You can layer these elements as desired.

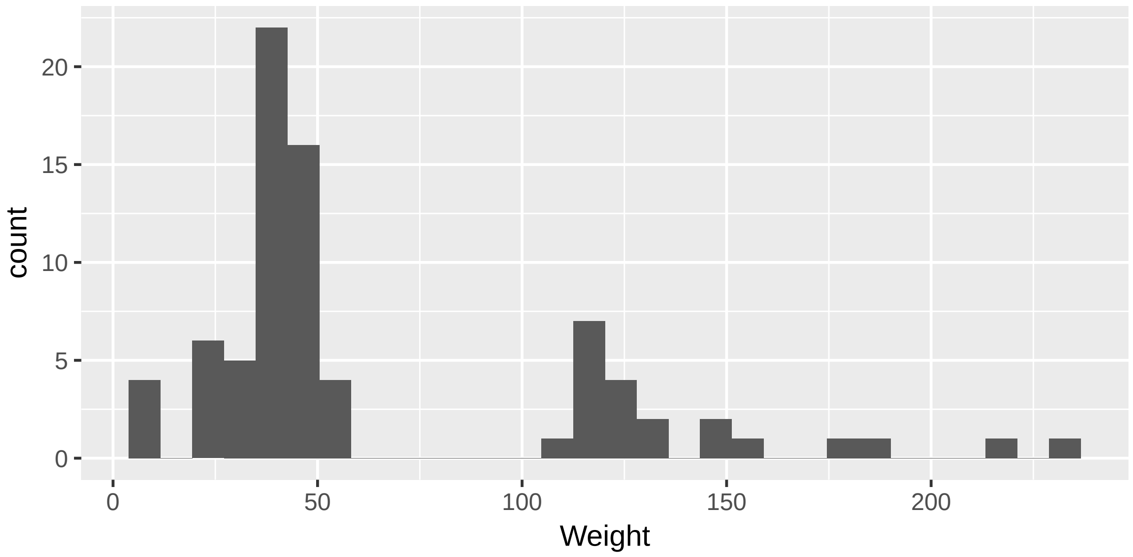
You can see in the code below that we are declaring `data = cleanedsurvey` in `ggplot()`, but not selecting any of the data columns yet. Just providing the data frame.

`+` then adds a `geom_histogram()` graphical layer, which is a histogram. We tell it that we want `x = Weight`. `Weight` is the column name without quotes. The case must also match what appears in the data.frame it is contained within. There are many types of graphical elements (http://docs.ggplot2.org/current/) within the ggplot2 library.

You'll see below that the plot is giving us a message that it is using a default bin value.

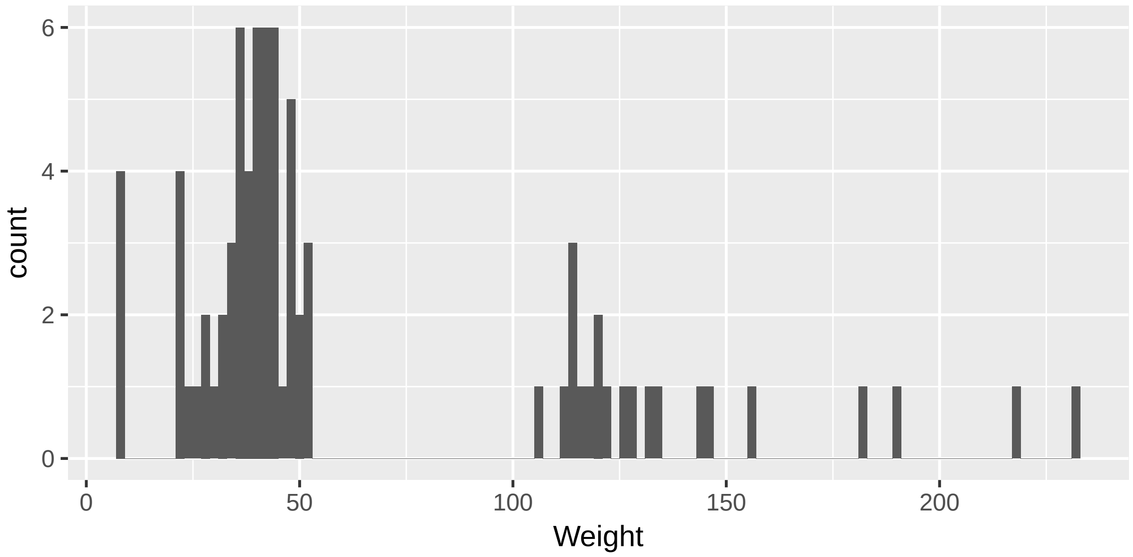
```r
ggplot(data = cleanedsurvey) + geom_histogram(aes(x = Weight))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As that error message hinted, `geom_histogram()` has additional options that you can play with. Place your curser just inside the last set of `()` press tab to see them. Let's play with binwidth.

```r
ggplot(data = cleanedsurvey) + geom_histogram(aes(x = Weight), binwidth = 2)
```



Now you

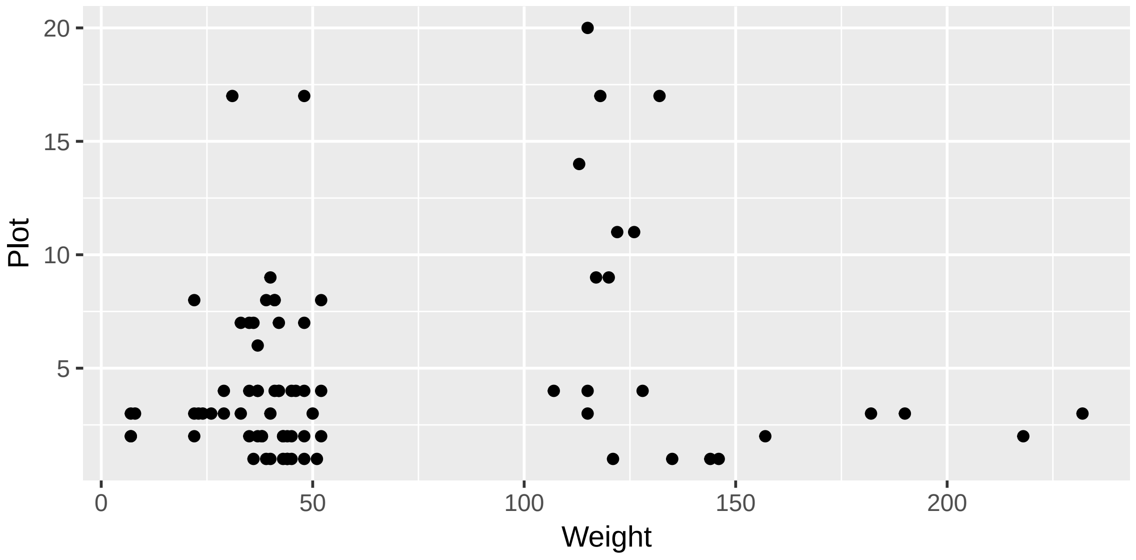
- Change some of the binwidth values to find the right balance for the data you are looking at
- Change the `x =` value to other columns and see what works and what doesn't.
- Which columns worked? Which columns didn't?
- What sort of errors did you get when you tried a column that didn't work?
- What other sort of errors did you see in this process?

Scatterplots and grouping

We can also use scatterplots to look at numerical data.

You can see that the syntax is already changing on us, but at least the x and y labels are showing our nice column headers.

```r
ggplot(cleanedsurvey, aes(x = Weight, y = Plot)) + geom_point()
```

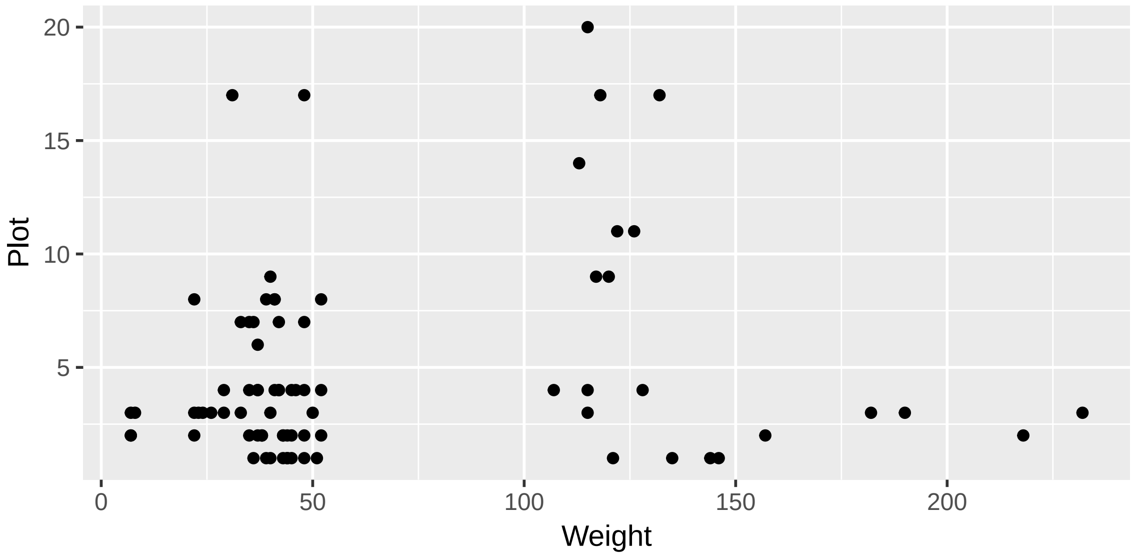


This is nice, but hard to tell a bit of what's going on. Let's group this in a few ways.

Because our code is getting long, we can save our basic ggplot code as a variable and reference it. We're going to call our ggplot() object `p`. Nothing will happen when we first create `p` because we must call p to make it appear. Calling `p` alone will show a blank graph. We need to add our graphical element layer to see the actual chart.

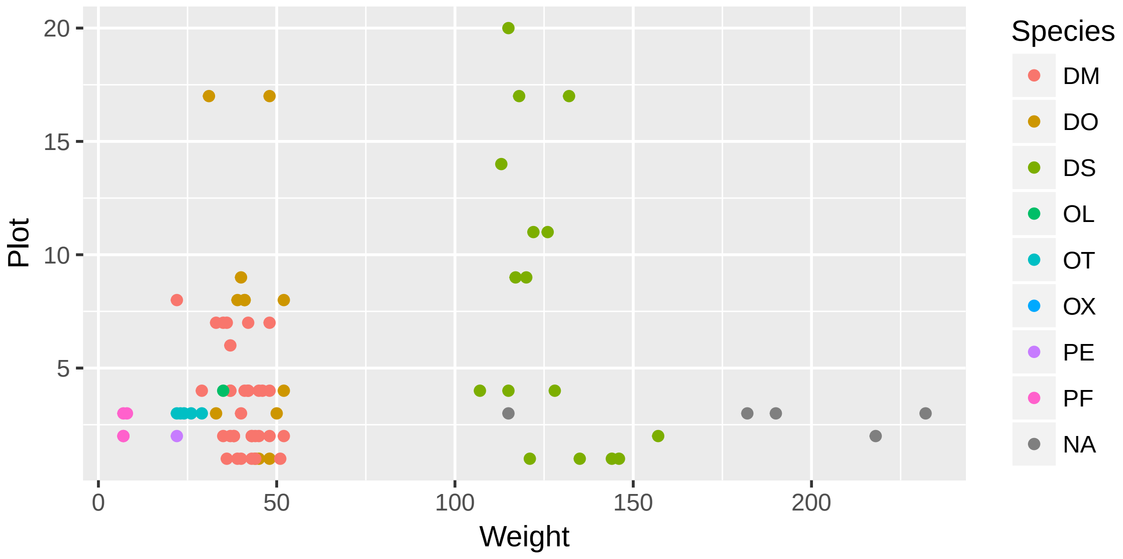
```r
p = ggplot(cleanedsurvey, aes(x = Weight, y = Plot)) # nothing will be plotted yet

p + geom_point() # the plot will appear when you evaluate it with a geometry element
```



Now we're seeing the power of having our data already in tidy format. Because all our labels are in a single column, associated with each observation, we can just point R to that column and have nice labels and colors without too much work.

```r
p + geom_point(aes(color = Species)) # color by the Species column
```



```r
#benign change
```

Now you

- Try changing the color variable to other columns within the data file.
- What worked? What didn't?
- What were the errors that you encountered?