Lecture 3: Plotting, Part 1 (base R)

Kelly

Admin and Annoucements

1. Discussion: how to get help, and expectations for me/us.

- What are the best strategies for getting help when you hit a wall?
- What is a reasonable turnaround time for email?
- How might we make sure everyone gets the support they need, mindful that there is only one of me?

2. Introducing Ramón

3. Problem Set post-mortem: what were the sticking points?

- Again, apologies for roundhouse().
- PDF problems are separate. What are the R problems?

Lecture

One of the things people love about working in **R** is the ability to quickly make and change high-quality plots. And it's true! Plotting is awesome and fun, and you will find yourself doing a bunch of it.

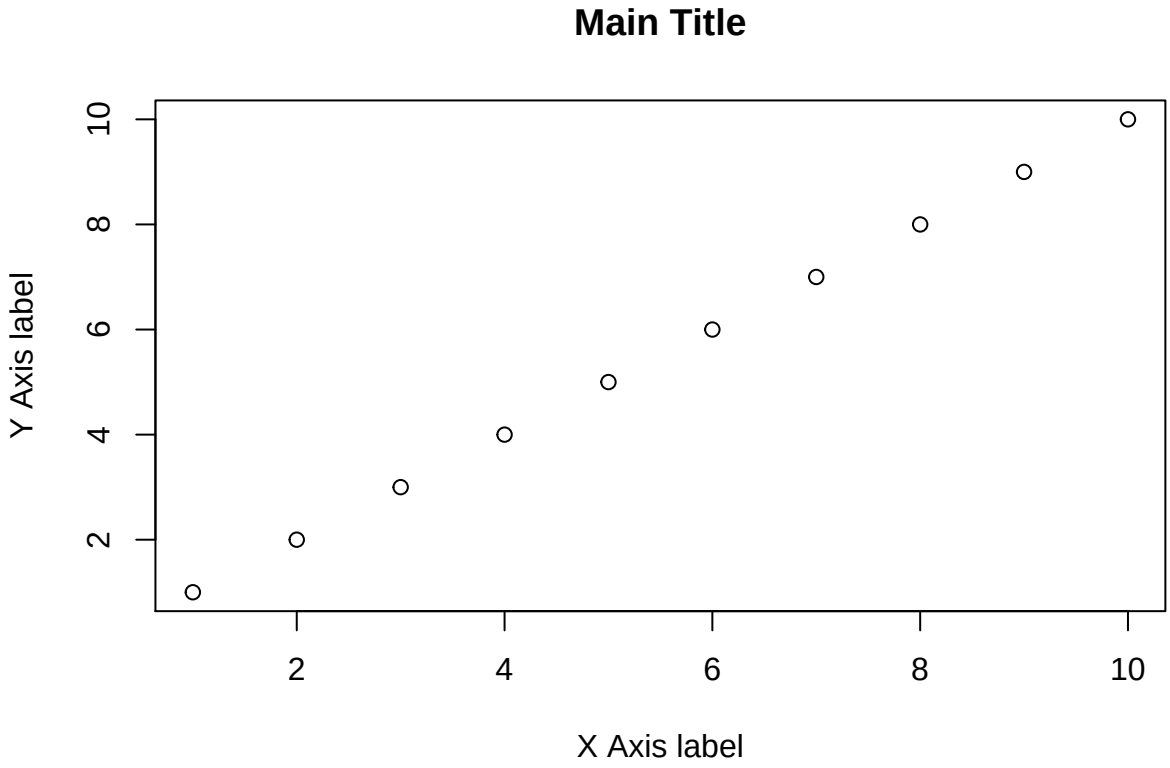
We'll look mainly at plotting with the main **R** graphics (called "base **R**"), and also briefly preview a different set of plotting tools called **ggplot2** (which has a totally different syntax that can be confusing, but that makes great graphics. ggplot2 is part of a larger suite of tools known as the *tidyverse* or *tidyR*, which is super useful but super different from base **R**, so I'll not mention it for now. Shhh. . . pretend I said nothing.)

Here again, we'll use code from the *Pirate's Guide*, and annotate it as necessary.

How to think about basic plotting in R

- A plot is like a canvas, and you can only work on one canvas at a time
- Some functions (like *plot()*) call entirely new canvases, others (like *points()*) add stuff to existing canvases
- The important thing isn't the plot; it's the code that made the plot. You can always quickly make a new graphic if you have the code.

```
# A basic scatterplot
plot(x = 1:10,  #this creates a vector from one to 10
     y = 1:10,  #this does the same
     xlab = "X Axis label",
     ylab = "Y Axis label",
     main = "Main Title")
```



Note that in the code we've specified the x and y variables (they are numeric vectors, in this case), and the *plot()* function knows we want to plot them against one another.

- If the vectors weren't the same length, what would happen?

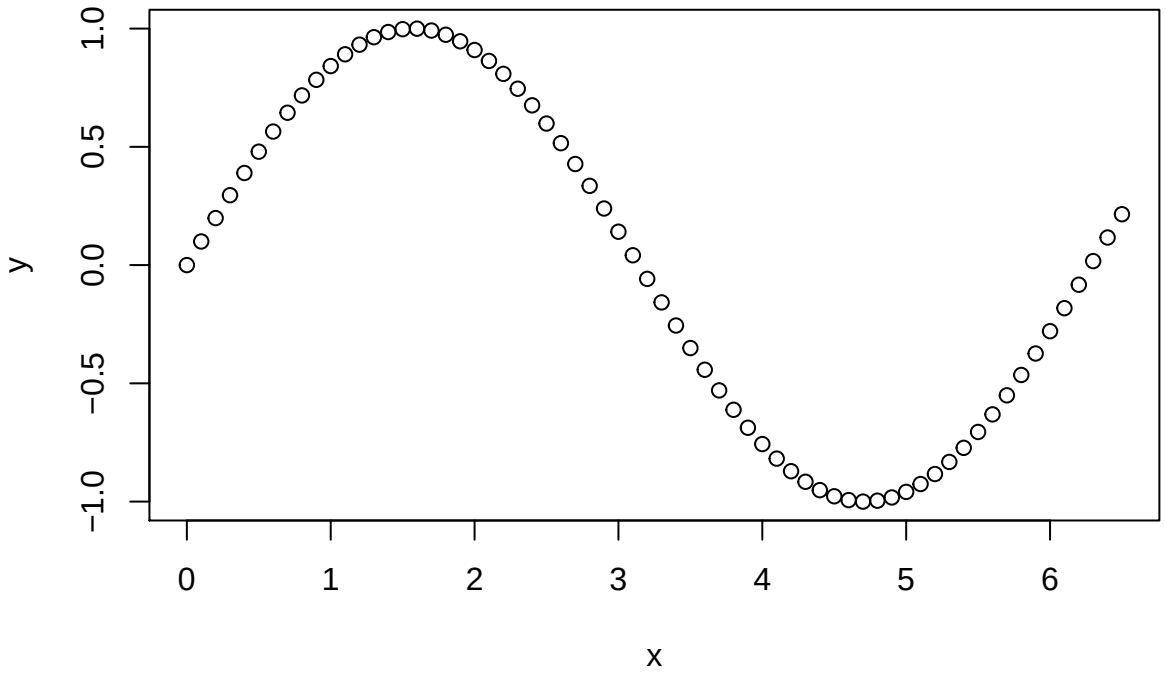
```
plot(y=1:10, x=1:11)
```

```
## Error in xy.coords(x, y, xlabel, ylabel, log): 'x' and 'y' lengths differ
```

As seen above, it doesn't make any sense to plot two vectors against one another if they aren't the same length.

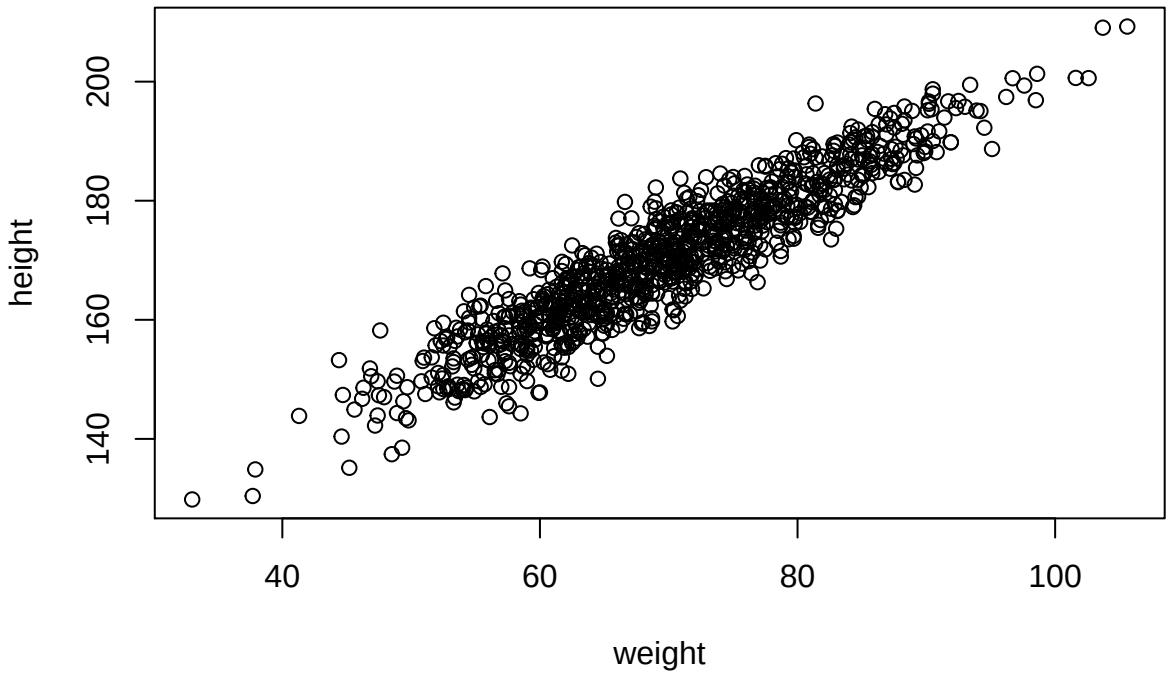
Note also that you can use *plot()* by specifying x and y, as above `plot(x = 1:10, y=1:10)` OR equivalently by using formula notation `plot(y~x)`. If we've already defined x and y somewhere else, it's clear what to plot. For example:

```
x = seq(0,6.5, 0.1)  #this creates a numeric vector from 0 to 6.5 by increments of 0.1
y = sin(x)  #this takes the sin(x)
plot(y~x)
```



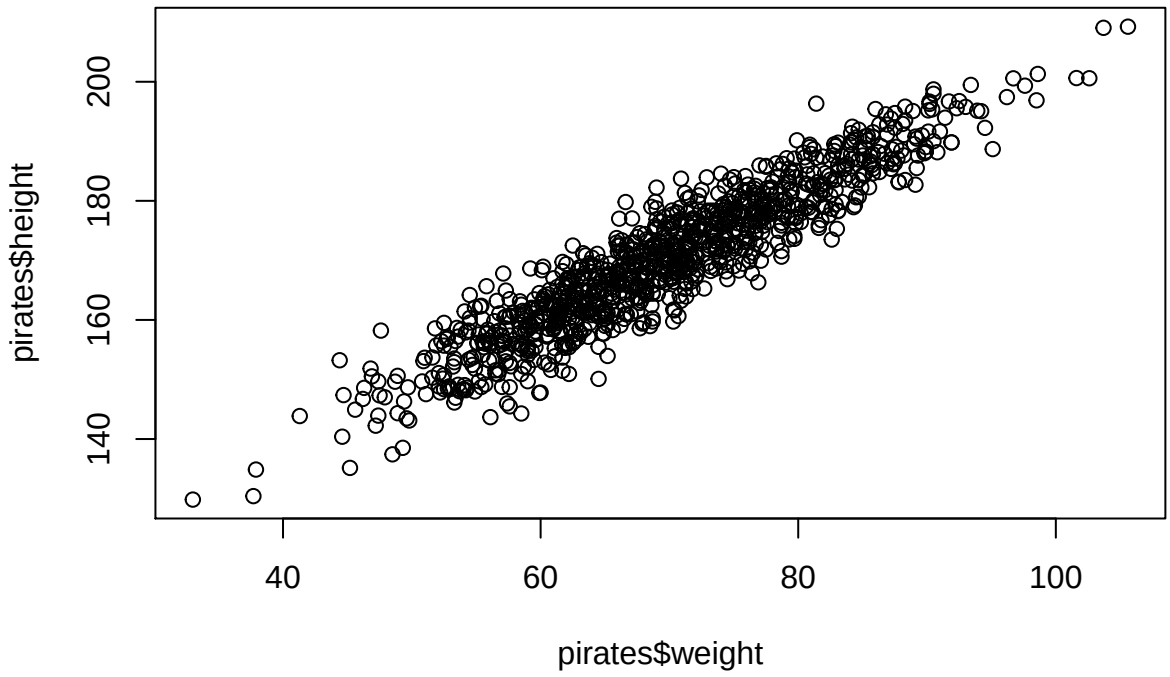
If we haven't defined the variables elsewhere, we can call them by specifying a dataframe that they came from. For example:

```
plot(height~weight, data=pirates)
```




```
#or equivalently,

plot(pirates$height~pirates$weight)
```



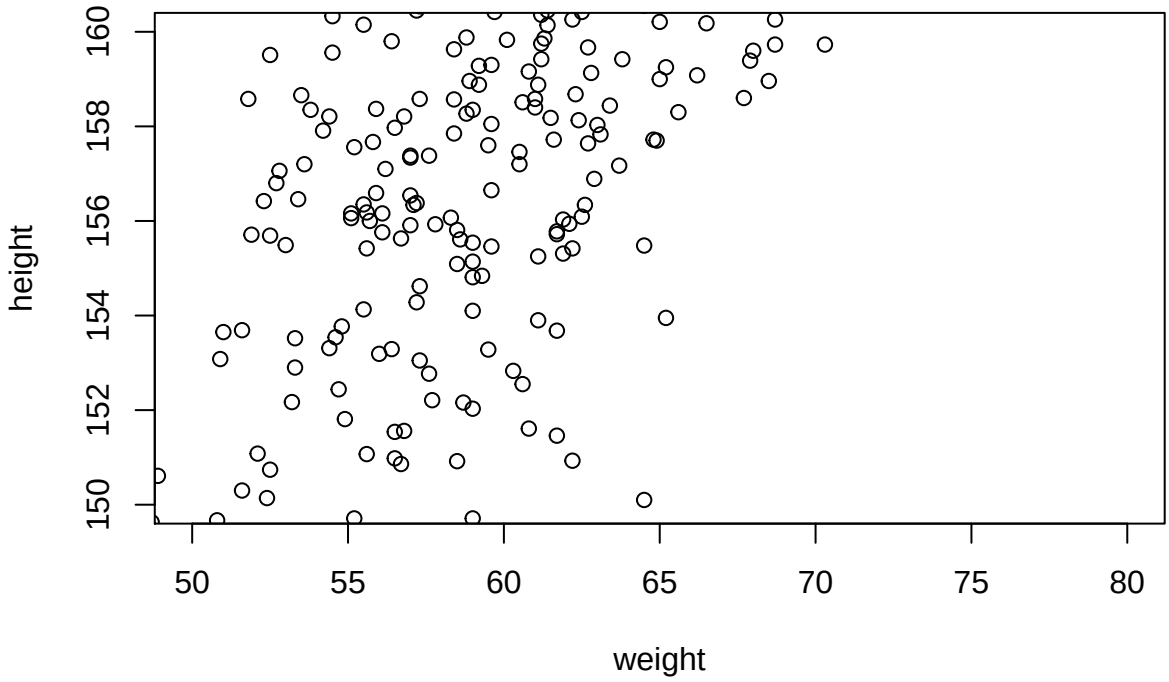
Note also – as shown above – that if you don't specify a title or x or y axis labels, that's OK... the plot just is a bit boring, but **R** still makes the plot for you.

Customizing plots

Scale

You might want to play with the x- or y-scales. Keeping our same plot as above, we can do this easily with the arguments *ylim()* and *xlim()* by providing a two-number vector: the min and max you want to see shown on the graph.

```
plot(height~weight,
     data=pirates,
     ylim=c(150,160),
     xlim=c(50,80)
     )
```



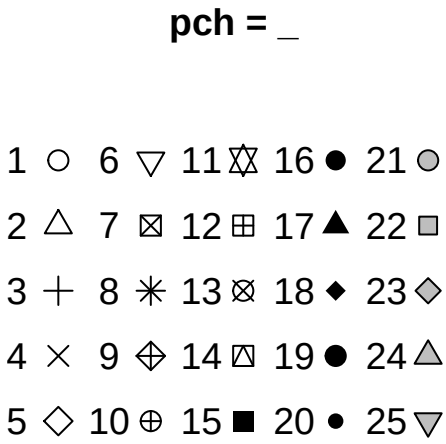


Figure 1: The symbol types associated with the pch plotting parameter.

Plotting Character

Now, let's say you want to plot in squares instead of circles. Sure. You can do that. Or filled circles. Or whatever. There's an argument for that, called *pch*, which stands for "plot character".

Using the code from the *Pirate's Guide*:

And you can even decide you want to plot using letters or other character strings, by giving *pch* a character instead of a numeral. For example, if today we really like the letter W. . .

```
plot(x = 1:10,       #let x be the integers 1:10
     y = rnorm(10),  #let y be 10 random numbers from a normal distribution (w default mean = 0, sd = 1
     pch = "w",      #use the plotting character "w"
     xlab = "",      #remove the x axis label
     ylab = "")      #remove the y axis label
```

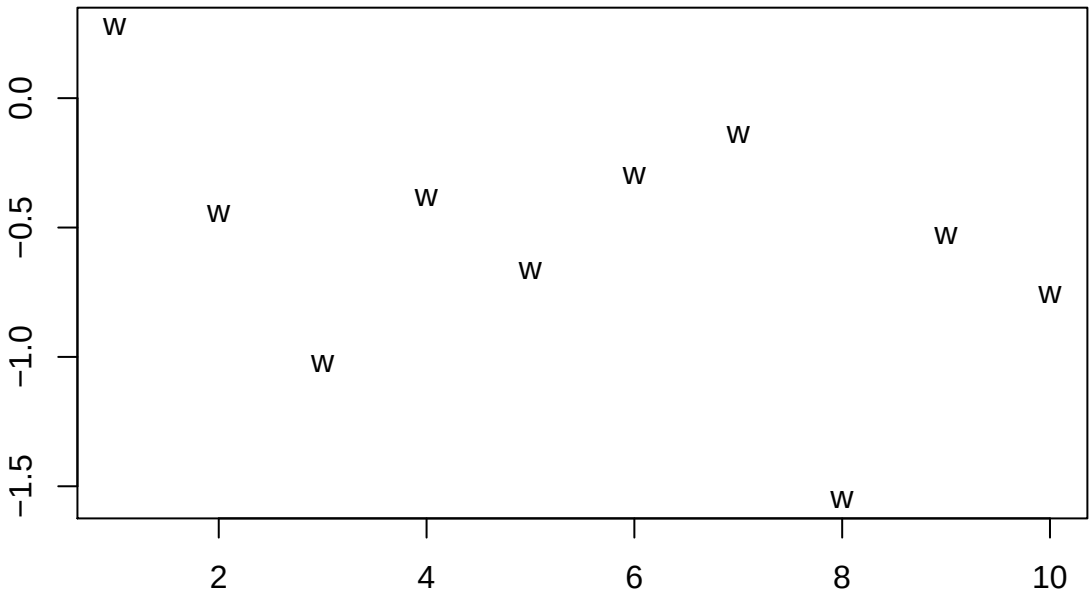


Figure 2: 100 random named colors (out of all 657) in R.

Color

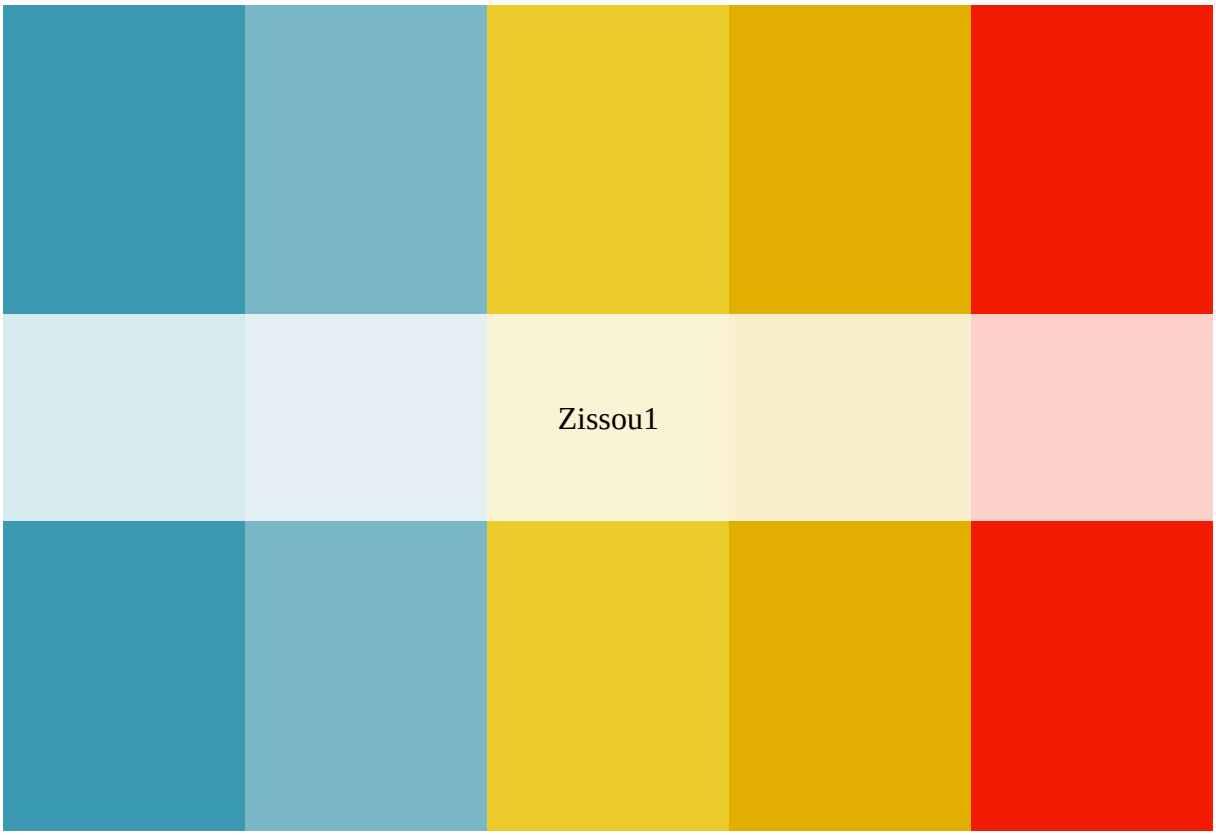
R has infinite color options; some are built-in, and you can generate new ones yourself. To borrow again from the *Pirate's Guide*:

> The easiest way to specify a color is to enter its name as a string. For example `col = "red"` is R's default version of the color red. Of course, all the basic colors are there, but R also has tons of quirky colors like `"snow"`, `"papayawhip"` and `"lawngreen"`. Figure @ref(fig:randomcolors) shows 100 randomly selected named colors.
>
> To see all 657 color names in R, run the code `colors()`. Or to see an interactive demo of colors, run `demo("colors")`.

Some awesome people have created color palettes based on movies, directors, etc. The *wesanderson* set of palettes might be my favorite. For example:

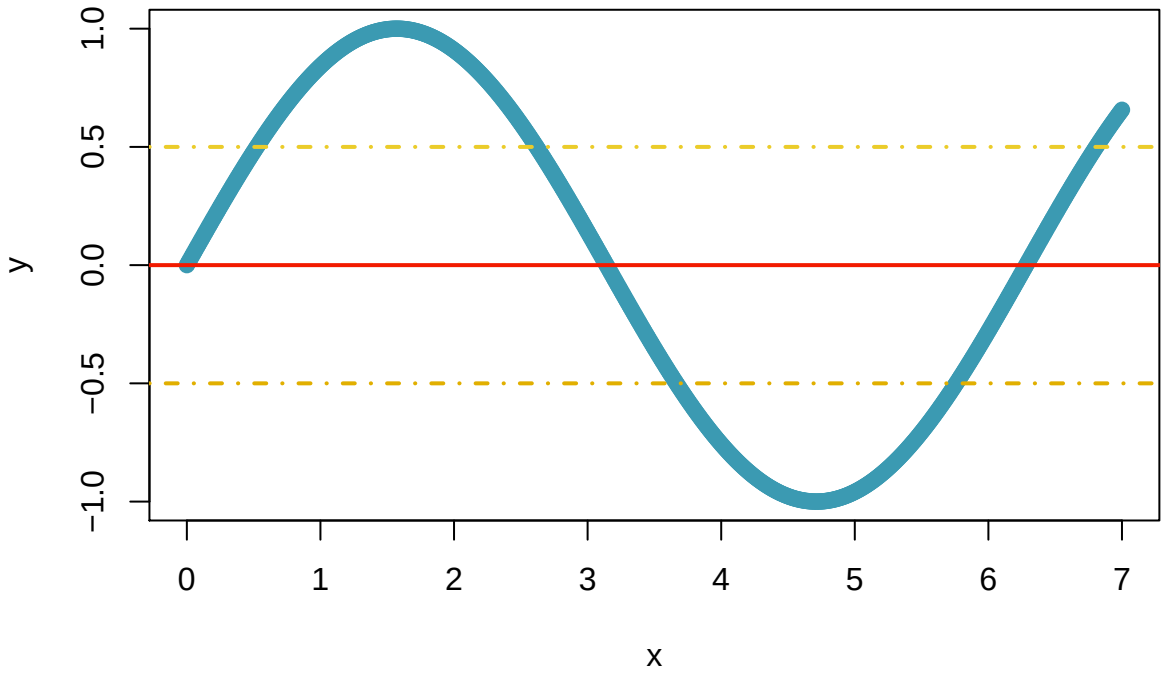
```
library(wesanderson)
```

```r
wes_palette("Zissou1")  #show the colors in the wesanderson palette "Zissou1"
```




```r
x <- seq(0, 7, 0.01)  #create a sequence of numbers from 0 to 7 by units of 0.01
y <- sin(x)  #calculate the sin(x)

plot(y~x,
     col=wes_palette("Zissou1")[1],  #note: wes_palette() is a function.
                                      # using this function with the argument "Zissou1" generates 5 col
                                      #and here, we're using a numerical index, [1], to say which color
     pch=19) #make the sine plot, but using the first
               #Zissou1 color and use solid circles as the plot character
abline(h=0,
       col=wes_palette("Zissou1")[5],
       lwd = 2) #draw a horizontal line at y = 0, using a Zissou1 color
abline(h=0.5,
       col=wes_palette("Zissou1")[3],
       lwd = 2,
       lty = 4) #draw a horizontal line at y = 0.5, using a Zissou1 color,
                  #and make it a dashed line (lty ... "for line type")
abline(h=-0.5,
       col=wes_palette("Zissou1")[4],
       lwd = 2,
       lty = 4) #draw a horizontal line at y = -0.5, using a Zissou1 color,
```




```
#and make it a dashed line (lty ... "for line type")
```

Many more parameters

See ?par for the complete set of base **R** plotting parameters. You can change just about anything about a graph: font, margins, rotation of labels, etc.

Non-Scatterplots

Life isn't all about scatterplots, of course. Let's see some other useful kinds of plots, again quoting from the *Pirate's Guide*:

Histogram: `hist()`

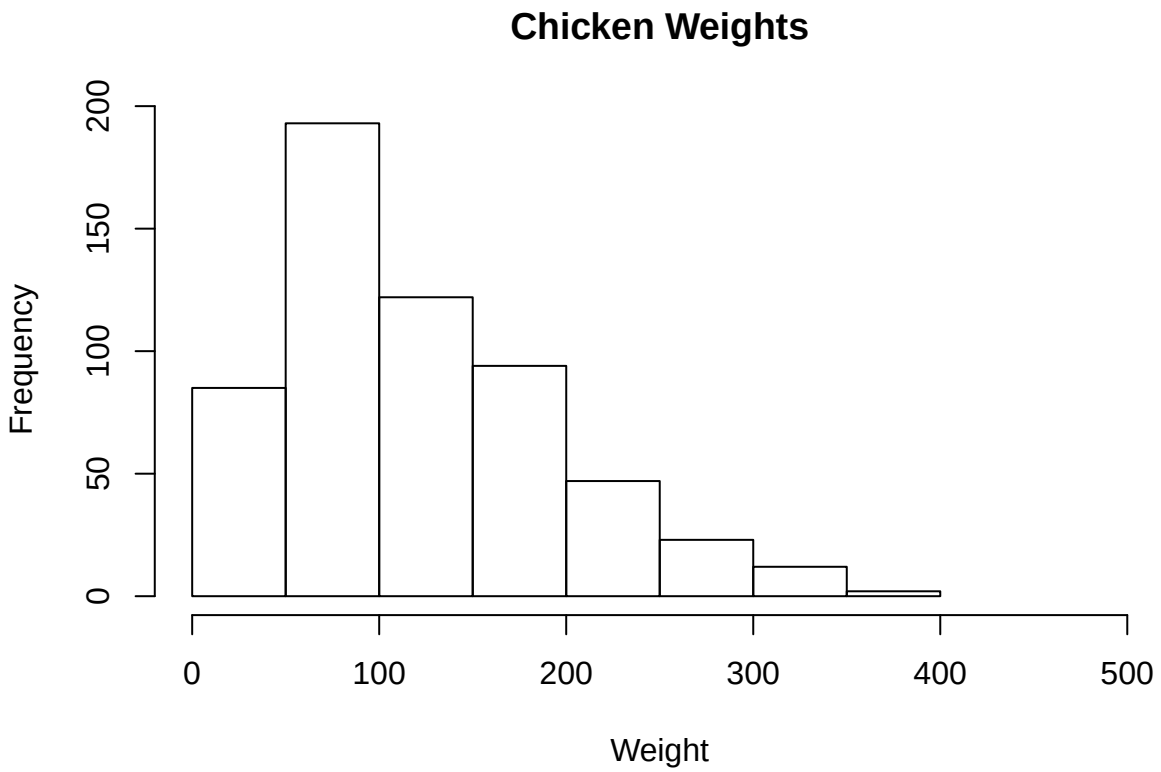
Table 1: (#tab:hist) `hist()` function arguments

Argument	Description
`x`	Vector of values
`breaks`	How should the bin sizes be calculated? Can be specified in many ways (see `?hist` for details)
`freq`	Should frequencies or probabilities be plotted? `freq = TRUE` shows frequencies, `freq = FALSE` shows probabilities.
`col`, `border`	Colors of the bin filling (`col`) and border (`border`)

Histograms are the most common way to plot a vector of numeric data. To create a histogram we'll use the `hist()` function. The main argument to `hist()` is a `x`, a vector of numeric data. If you want to specify how the histogram bins are created, you can use the `breaks` argument. To change the color of the border or background of the bins, use `col` and `border`:

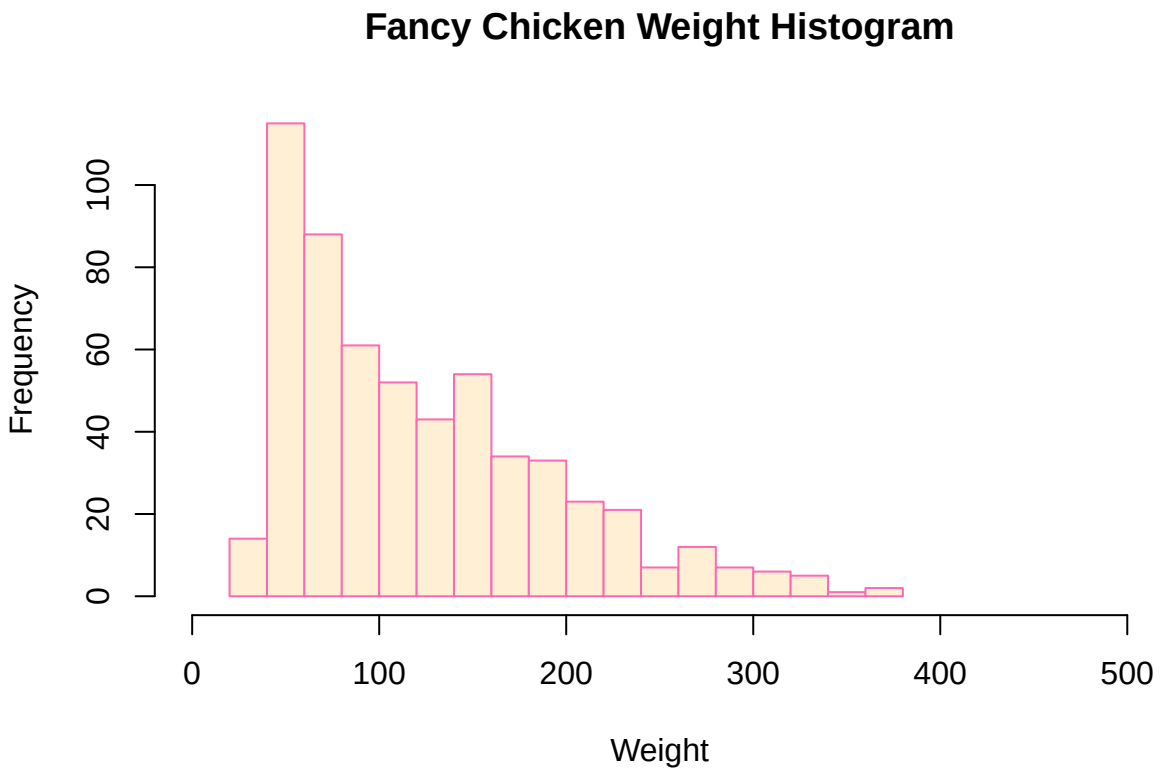
Let's create a histogram of the weights in the ChickWeight dataset:

```r
hist(x = ChickWeight$weight,
     main = "Chicken Weights",
     xlab = "Weight",
     xlim = c(0, 500))
```



We can get more fancy by adding additional arguments like `breaks = 20` to force there to be 20 bins, and `col = "papayawhip"` and `bg = "hotpink"` to make it a bit more colorful:

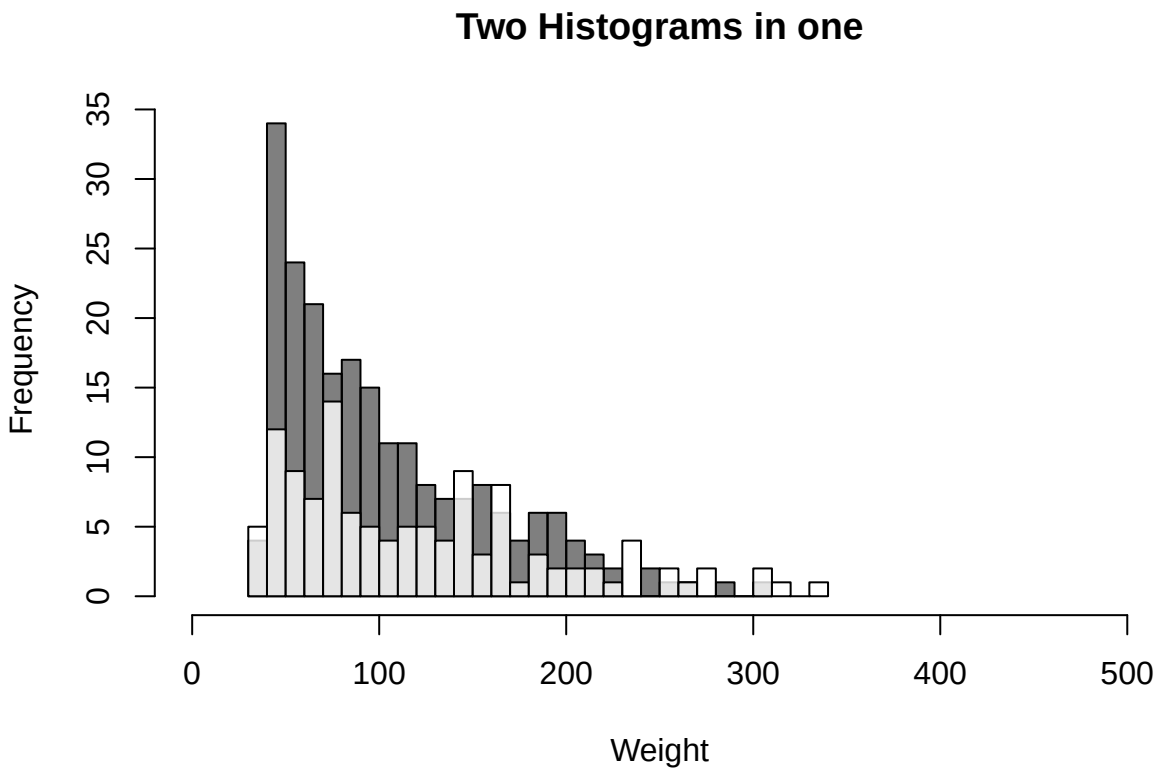
```r
hist(x = ChickWeight$weight,
     main = "Fancy Chicken Weight Histogram",
     xlab = "Weight",
     ylab = "Frequency",
     breaks = 20, # 20 Bins
     xlim = c(0, 500),
     col = "papayawhip", # Filling Color
     border = "hotpink") # Border Color
```



If you want to plot two histograms on the same plot, for example, to show the distributions of two different groups, you can use the `add = TRUE` argument to the second plot.

```
hist(x = ChickWeight$weight[ChickWeight$Diet == 1],
     main = "Two Histograms in one",
     xlab = "Weight",
     ylab = "Frequency",
     breaks = 30,
     xlim = c(0, 500),
     col = gray(0, .5))

hist(x = ChickWeight$weight[ChickWeight$Diet == 2],
     breaks = 30,
     add = TRUE, # Add plot to previous one...
                 #note this LOOKS like a stacked bar chart, but is not!
                 #it is merely one histogram in front of another.
                 #which is dangerous, because it hides data.
     col = gray(1, .8))
```



And More

You get the point. For any plot you can imagine (and more), there's a way to do it in **R**. People have created additional packages for things like *waffleplots* , *network diagrams* and all kinds of things.

ggplot2

We're going to look more at this in two weeks, but just as a quick preview, there are other packages for **R** that people have created to create graphics in different ways. One of those is *ggplot2*, which has different syntax, and can make cool things like this:

```
library(ggplot2)
```

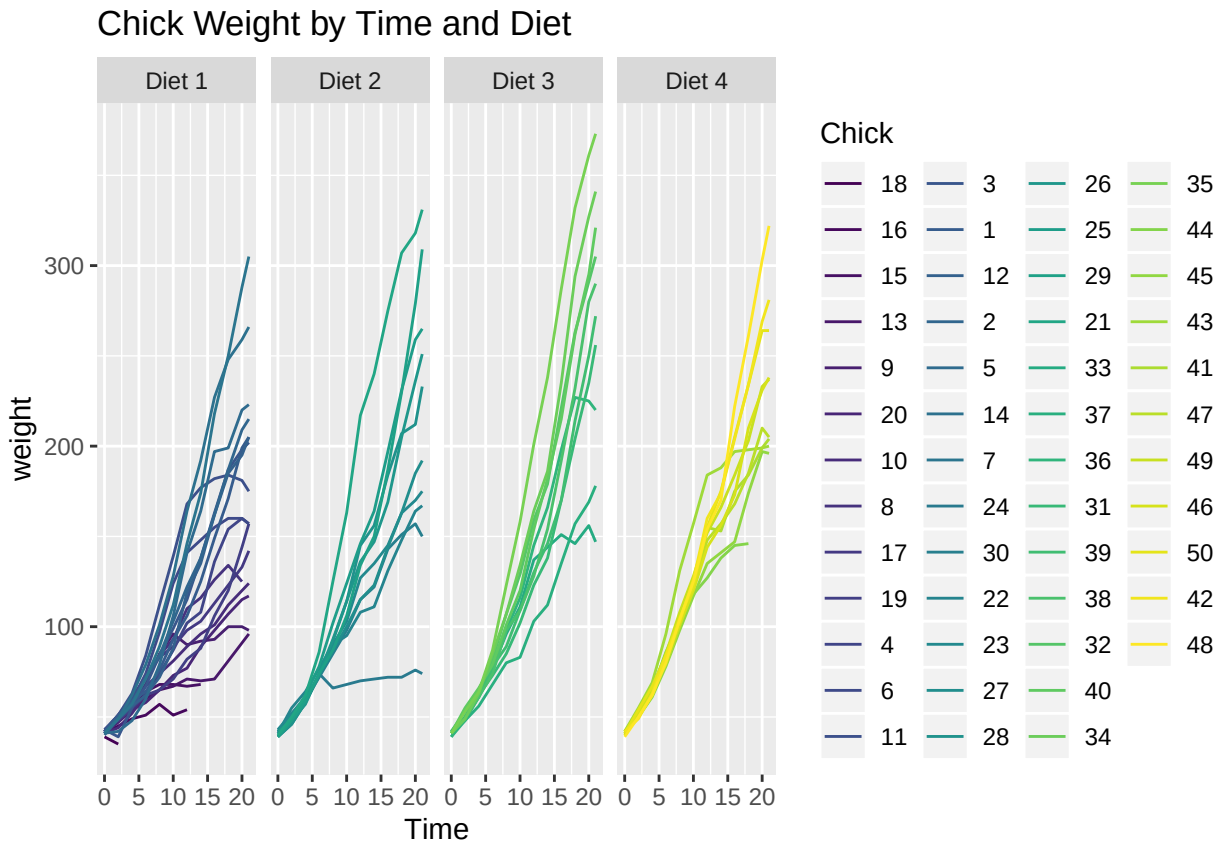
```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:yarrr':
##
##     diamonds
```

```
library(viridis) #cool color palettes
```

```
## Loading required package: viridisLite
```

```
levels(ChickWeight$Diet)<-c("Diet 1", "Diet 2", "Diet 3", "Diet 4")  #relabel Diet categories for clari
  ggplot(data = ChickWeight, aes(x=Time, y=weight, color=Chick))+
    geom_line()+
    facet_grid(~Diet)+
    guides(color=guide_legend(ncol=4))+
    labs(color="Chick")+
    ggtitle("Chick Weight by Time and Diet")
```




```
  ggplot(data = ChickWeight, aes(x=Time, y=weight, color=Diet))+
    geom_point()+
    geom_smooth() +
```

```
    ggtitle("Mean Chick Weight by Time and Diet") +
    scale_color_viridis(discrete = TRUE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```