Workflows for Visualization

Introduction

Visual analogies often play a key role in research, helping us articulate hypotheses and understand results of hypothesis tests. Yet, while hypothesis testing is a core component of science education, visualization, and, more generally, the acts of perception and imagination that necessarily precede hypothesis testing, do not receive as much attention in our curricula, despite their critical importance.

For one thing,the basics of making good figures, which are explicitly known and practiced by most successful scientists but rarely formally taught. Indeed, I believe the association between knowing how to make a good figure and being a successful scientist is causal.

More generally, as biologists we are often working with complex systems. Visualizing complex systems requires us to bring into focus specific emergent properties of a system, by mapping to a simpler context, and by leaving most things out. In this sense, visualizing complex systems is similar to building mathematical models, indeed visualizing complex systems requires some mathematics, and catalyzes further quantitative analyses.

Here's a cute argument: a huge component of our brains is evolved for spatial reasoning - for instance, to help us find food for ourselves and our families, and not become food for another creature. Think of this like the GPU (graphics processing card) of your brain-computer. It is soooo much more powerful than your abstract processing abilities (could think of this like the CPU of the computer). We can do better science by working on our GPUs...

Example of a basic visualization workflow

Here we chronicle the journey from a simple dataset to an acceptable figure, in base R.

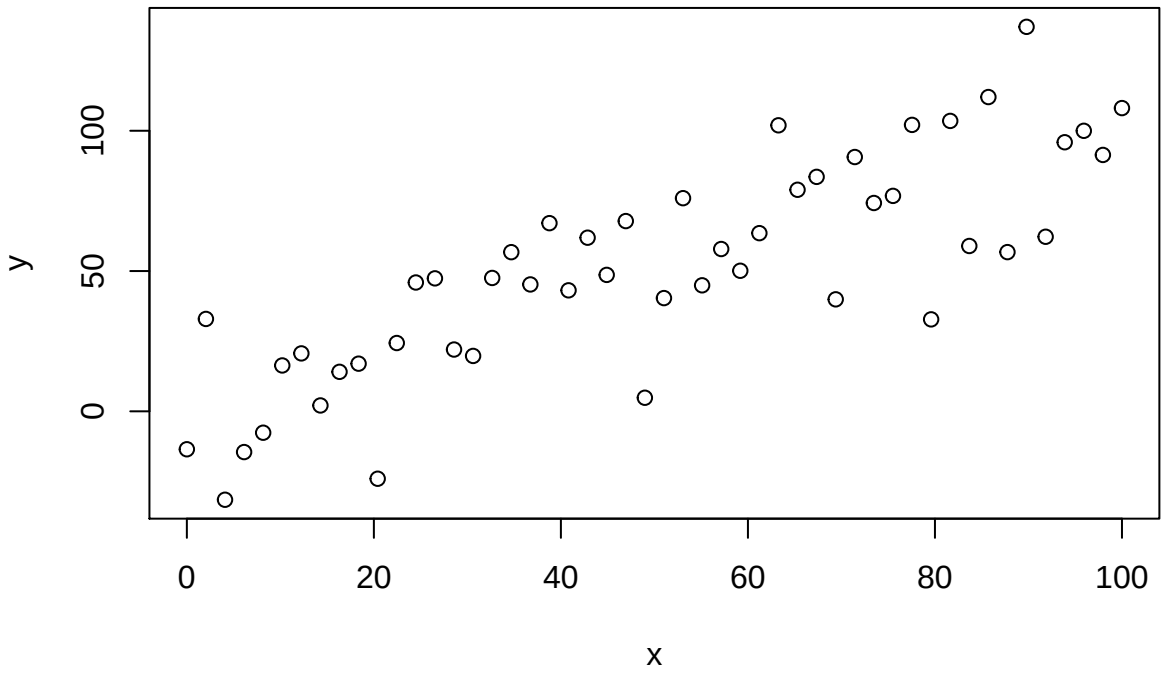
Setup

We will start by generating some data, as an additive combination of a linear "signal" and some white noise

```
n <- 50
a <- 1
b <- 2
sig <- 20
noise <- rnorm(n, 0, sig)

x <- seq(0, 100, length.out = n)
signal <- a * x + b
y <- signal + noise
```

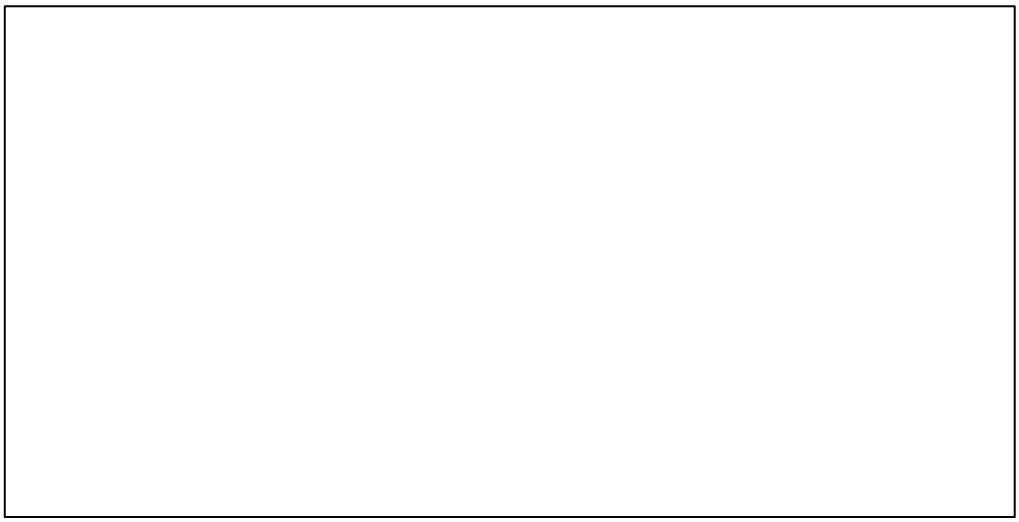
At this point it is worth spending some time in the help files for `plot()` and `par()` if you have not already. This is just to get a sense of the potential to control things, and how the basic control process works. Most people need to continually refer to these help files as they work on specific applications.

```
plot(x, y)
```



Starting a plot from the VERY beginning - (i.e. all defaul elements turned off) - if you want to control everything manually.

```
plot(x, y,
     type = "n", #do not even plot the data
     #bty = "n",  #box type is "n" means don't draw a box around the plot
     xaxt = "n", #similarly, don't make an x axis...
     yaxt= "n",  #"... or a y axis"
     xlab = "",  #and axis labels are blank
     ylab = ""
     )
```



Before tweaking optional things, there are some minimum standards to take care of. Every figure should have informative axis labels. And these ned to be big enough to be read when the figure appears in a published article. To this end, it can be helpful to control the size of a figure right from the start. If you have not already, spend some time in the help files for `plot()` and `par()`.

```
par(fin = c(4,4))           #figure dimensions (width, height)
par(mai = c(1,1,1,1))       #plot margins (bottom,left,top,right)
plot(x, y,
     xlab = "Years since Mastodon reintroduction",
     ylab = "Population size")
```

By labelling the axes I have just realized there are y values that make no sense given what y is: population size. This is an example of how disciplined practices in making figures (e.g. always having informative axis labels from the start) can improve the whole scientific process: now we have an opportunity to improve our data stream by figuring out what to do with our negative y values. For our purposes here, let's just remove them:

```
y[y < 0] <- NA
```

Now lets start work on minimizing the amount of effort a readers brain has to do to "get" the figure. There are some initial things that apply to every figure that we can do right away. Then in the next section we can look at making specific messages come forward.

Little things can go a long way. Custom tick marks are usually needed to make professional-grade figures. To do that, we need just enough tick labels to orient the reader to the "space" of the plot and help them quickly measure distances; any tickmarks beyond that just become visual noise the reader has to work to filter out. We start by turning off the default axes. Then we add our own, with axis limits hardcoded.

From a reproducibility standpoint, hardcoding is a liability. The alternative is to write code that determines what the best limits should be on the fly, but we will not be distracted by that now.

```
par(fin = c(4,4))
par(mai = c(1,1,1,1))
plot(x, y,
     xlab = "Years since Mastodon reintroduction",
     ylab = "Population size",
     xaxt = "n",
     yaxt = "n",
     xlim = c(0, 100),     #hardcoded axis limits...
```

```
     ylim = c(0, 120)
     )
axis(1, c(0, 50, 100))    #... and tick locations
axis(2, c(0, 60, 120))
```

Tune for impact

Now that we have something that meets minumum formatting requirements we can ask: what are we trying to say with this figure and how can we make that pop? Let us say our main message here is that the Mastodon population is increasing in a predictable way. That would be the first line of a figure legend: "ie Figure 1: Following reintroduciton in 2023, the Mastodon population has increased in a predictable way." This is also the first sentence you would say in a presentation of this slide. The task at hand is to make the visual say that loud and clear. We need a regression line.

```
fit <- lm(y ~ x)
```

We could add this to the figure a number of ways, for example, using the function `abline()` and supplying the fitted model as input

```
par(fin = c(4,4))
par(mai = c(1,1,1,1))
plot(x, y,
     xlab = "Years since Mastodon reintroduction",
     ylab = "Population size",
     xaxt = "n",
     yaxt = "n",
     xlim = c(0, 100),
     ylim = c(0, 120)
     )
axis(1, c(0, 50, 100))
axis(2, c(0, 60, 120))

abline(fit)                    #quick way to add regression line to an existing plot
```

We get a lot more control of how the predictions are displayed if we generate an additional dataset that is the predicitons, then plot that. First generating the predictions

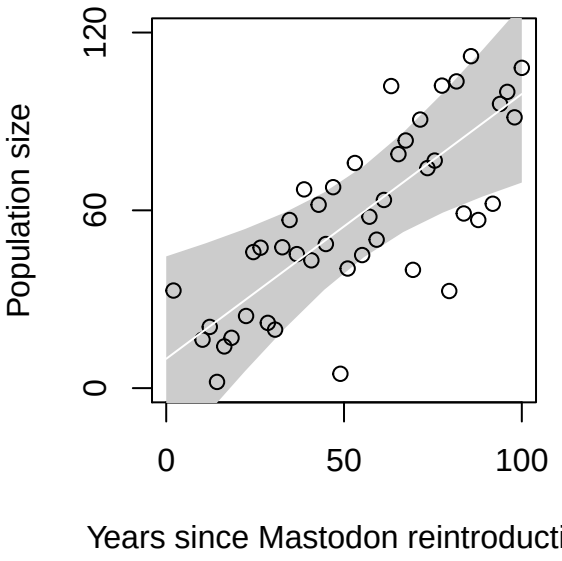
```
npred <- 10
xpred <- seq(min(x), max(x), length.out = npred)                          #points at which we want to pre
ypred <- predict(fit, newdata = data.frame(x = xpred), se.fit = TRUE)    #predicting corresponding y val
```

Now plotting:

```
par(fin = c(4,4))
par(mai = c(1,1,1,1))
plot(x, y,
     xlab = "Years since Mastodon reintroduction",
     ylab = "Population size",
     xaxt = "n",
     yaxt = "n",
     xlim = c(0, 100),
     ylim = c(0, 120)
     )
axis(1, c(0, 50, 100))
axis(2, c(0, 60, 120))

lines(xpred, ypred$fit)
```

It is often easy to reproduce tidyverse functionality in base R. Let's make our plot look like ggplot

```
# Original code, modified to not plot the points at first
par(fin = c(4,4))
par(mai = c(1,1,1,1))
plot(x, y,
     type = "n",                                    #first plot with points missing
     xlab = "Years since Mastodon reintroduction",
     ylab = "Population size",
     xaxt = "n",
     yaxt = "n",
     xlim = c(0, 100),
     ylim = c(0, 120)
     )
axis(1, c(0, 50, 100))
axis(2, c(0, 60, 120))

# Draw a polygon for the confidence envelope
xpoly <- c(xpred, rev(xpred))                       #add confidence envelope as bottom layer
ypoly <- c(ypred$fit + 5 * ypred$se.fit,
           rev(ypred$fit - 5 * ypred$se.fit))

polygon(x = xpoly, y = ypoly,
        border = NA,
        col = grey(0.8))

# Add in the points and line
points(x,y)                                         #then data
lines(xpred, ypred$fit, col = 'white')              #then trendline
```



That takes care of our primary messsage: "Mastodon population is increasing in a predictable way." What about secondary messages? Suppose we are interested in what caused the population to grow especially quickly or slowly in some years. Let's use symbol color and shape to highlight points that are outside the confidence envelope.

To start with, it will be helpful to have the model prediction and standard error for each observation point, rather than at 10 evenly spaced points as it was before

```
yhat <- predict(fit, newdata = data.frame(x = x), se.fit = T)
```

Now let's identify points that are boom and bust years, meaning those above and below the confidence envelope for our fitted model

```
is_boom_year <- y > yhat$fit + 5*yhat$se.fit
is_bust_year <- y < yhat$fit - 5*yhat$se.fit
```

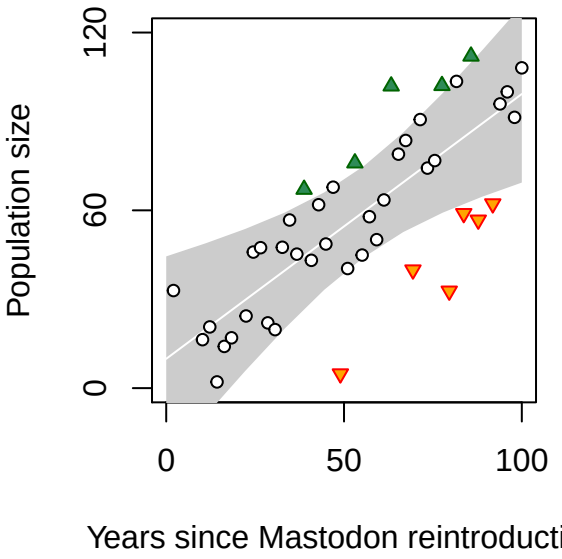
I like the following procedure when working with symbology. First specify size, shape and color as vectors, mapping from data. Then we call plot function. I think this is better than doing the mapping inside the plot function call, because that is hard to read and debug.

```
# Vectors for symbology
cex <- rep(1, n)        #size
col <- rep("black", n)  #color
pch <- rep(21, n)       #shape.  See ?points


# Change them to highligh features in data
cex[is_boom_year || is_bust_year] <- 2
col[is_boom_year] <- "green"
col[is_bust_year] <- "red"
pch[is_boom_year] <- 2
pch[is_bust_year] <- 6
```

The most important thing is that it pops: do not ever rely on subtle differences in color or shape to get across key results. One way to achieve that is to simultaneously alter multiple features, such as changing shape, color and size together.

```
# Vectors for symbology
cex <- rep(0.8, n)        #size
```

```r
col <- rep("black", n)  #color
bg <- rep("white", n)        #background color
pch <- rep(21, n)       #shape.  See ?points


# Change symbology to highligh features in data
cex[is_boom_year || is_bust_year] <- 1.3
bg[is_boom_year] <- "seagreen"
bg[is_bust_year] <- "orange"
col[is_boom_year] <- "darkgreen"
col[is_bust_year] <- "red"
pch[is_boom_year] <- 24
pch[is_bust_year] <- 25


# Make plot first without points
par(fin = c(4,4))
par(mai = c(1,1,1,1))
plot(x, y,
     type = "n",
     xlab = "Years since Mastodon reintroduction",
     ylab = "Population size",
     xaxt = "n",
     yaxt = "n",
     xlim = c(0, 100),
     ylim = c(0, 120)
     )
axis(1, c(0, 50, 100))
axis(2, c(0, 60, 120))


# Draw a polygon for the confidence envelope as bottom layer
xpoly <- c(xpred, rev(xpred))
ypoly <- c(ypred$fit + 5 * ypred$se.fit,
           rev(ypred$fit - 5 * ypred$se.fit))

polygon(x = xpoly, y = ypoly,
        border = NA,
        col = grey(0.8))


#Add trendline
lines(xpred, ypred$fit, col = 'white')

# Add in the points, with their symbologies pre-specified above (no calculations here)
points(x,y,
       pch = pch,
       col = col,
       bg = bg,
       cex = cex)
```



Now this figures says:

1. "Mastodon population is increasing in a predictable way."
2. "We are going to focus on the years where growth was particularly fast or slow relative to those predictions"

Future graphs could use the same color scheme and symbology to denote analyses pertaining to the boom and bust years.

Remark

A corrolary of this approach is that figures should have a small number of clear, concrete messages. For any figure you show your audience, you should be able to explicitely populate a (short) list like the one above ("This figure says:"). Having done that, you can evaluate how efficiently the figure delivers its message. A figure does not have to show every aspect of the data, or even most aspects. Every display element should be critical to the core message. In other words, if you can leave it out with compromising the core message, you should do so. If you are worried you are misleading readers by leaving something out, include a note in the figure legend about what was omitted, and consider including a more complete (less efficient) version of the figure in the supplemental materials.