Multipanel plotting in R (with base graphics)

BISC 888–1 Simon Fraser University

Sean C. Anderson
sean@seananderson.ca

November 6, 2013

Edward Tufte, *Envisioning Information*:

> "At the heart of quantitative reasoning is a single question: *Compared to what?* Small multiple designs, multivariate and data bountiful, answer directly by visually enforcing comparison of changes, of difference among objects, of the scope of alternatives. For a wide range of problems in data presentation, small multiples are the best design solution."

1 Multipanel approaches in R

To my knowledge, there are five main approaches to multipanel layouts in R.

Do them by hand Manually combine your plots in graphics software outside of R. Advantages: you get complete control over your layout. Disadvantages: just about everything else. Your figure is no longer reproducible. This can become increasingly annoying as analyses inevitably get re-run. It can also be time-consuming to perfectly line up your panels. I try and avoid this at all costs, but occasionally it's your only or best choice.

grid graphics, `lattice`, `ggplot2` Packages like `ggplot2` and `lattice` are great. Where I think they excel is in exploratory data analysis. You might be able to generate ten `ggplot` figures in the time it would take you to do the same in base graphics. Data analysis involves a lot of exploratory data plotting, so don't underestimate the value of this. Base graphics shine when it comes to plot customization. Data presentation for publication often consists of making highly-customized plots tailored to your specific situation. I use both, but almost always base graphics for publication. Learning a `ggplot2` can be very helpful, but you still need to learn base graphics. This workshop will focus on base graphics.

`par(mfrow)` The simplest method in base graphics. Works well for simple grid layouts where each panel is the same size.

`layout()` In addition to what you can do with `par(mfrow)`, `layout()` lets you combine panels.

`split.screen()` In addition to what you can do with `par(mfrow)` and `layout()`, `split.screen()` lets you specify the co-ordinates of your panels. Panels no longer have to be simple ratios of each other.

2 Where I make a silly analogy to explain the increasing levels of complexity

`par(mfrow)`, `layout()`, and `split.screen()` are all capable of basic equal-sized-panel grid layouts. If you think of creating a small multiple layout in R to be like putting screws into a wall: `par(mfrow)` would be the equivalent of grabbing your Leatherman to hang one picture frame — it's all you need and it's fast. `layout()` would be the equivalent of hunting around for a proper screwdriver to hang a bunch of picture frames. `split.screen()` would be the equivalent of finding and plugging in your power drill — more of a hassle to set up, but much more powerful in the end. Don't grab a tool that's more complex than it needs to be, but don't try and build a house with a Leatherman.

3 Questions to ask yourself when making a multipanel plot

1. What comparison do I want to emphasize?

2. How can I use order to enhance the comparison?
3. Is this a series of plots or does the grid layout matter? (`facet_wrap` vs. `facet_grid` in `ggplot2` terminology)
4. What's a reasonable number of panels to show? Everything? A sample?
5. Which axes can I fix and which need to vary? Would a log transformation be appropriate and allow the axes to be combined?
6. What chart junk can I remove?
7. What's important in my plots and what is necessary but less-important elements do I want to de-emphasize?
8. Can I make it all smaller and increase the information density without detracting from readability? (Almost always, yes.)
9. If the layout is complicated, have I drawn it out on paper first?

4 Margin space

Extra margins are usually wasted space and a break in the comparisons between panels. You will almost always want to shrink your margins. Set your margins for each panel with `mar` and your outer margins with `oma`. If all the axes can be shared then set `mar = c(0,0,0,0)`. These numbers refer to the space on the bottom, left, top, and right. Then you can use `par(oma)` to set your outer margins to create the necessary space for axes. If your content won't show up in the outer margins, you'll need to set `par(xpd = NA)`. See Figure 1 for an illustration of setting margin space with `mar` and `oma`.

5 Ways to iterate through your data

Common approaches are to use a `for` loop with subsetting or an `apply` function. You could also manually make all your plots, but unless you were only making a few panels this would get tedious and error prone.

A favourite approach of mine is to use `d_ply()` from the `plyr` package. This takes a data frame, splits it up, does something with it (plots it), but doesn't return a value.

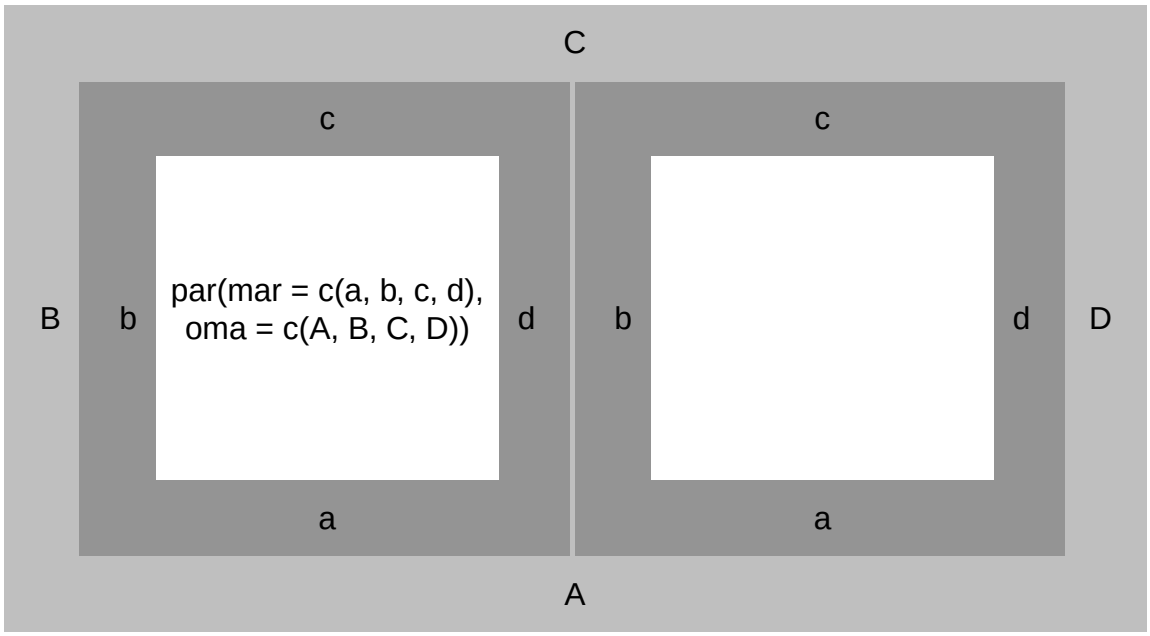


Figure 1: An illustration of how `mar` and `oma` interact to create margin space. `mar` controls the panel margins and `oma` controls the outer plot margins. If you only want axis labels on the outer panels, then you will often set `par(mar = c(0, 0, 0, 0))` and only use outer margins.

6 Labelling your panels

There are a number of ways of doing this.[1] Here, we're going to use this function to add alphabetical labels to our panels:

```
#' @param xfrac The fraction over from the left side.
#' @param yfrac The fraction down from the top.
#' @param label The text to label with.
#' @param pos Position to pass to text()
#' @param ... Anything extra to pass to text(), e.g. cex, col.
add_label <- function(xfrac, yfrac, label, pos = 4, ...) {
  u <- par("usr")
  x <- u[1] + xfrac * (u[2] - u[1])
  y <- u[4] - yfrac * (u[4] - u[3])
  text(x, y, label, pos = pos, ...)
}
```

R comes with two built-in character vectors that are useful for labelling panels:

```
letters[1:8]
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```

```
LETTERS[1:8]
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
```

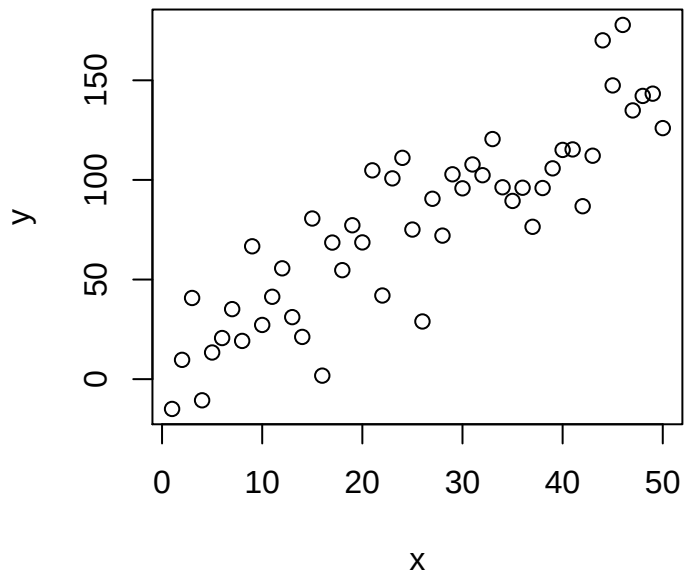
7 Basic multipanel layouts with `par(mfrow)`

For most basic grid layouts, `par(mfcol)` or `par(mfrow)` are your simplest option. `mfrow` plots row by row and `mfcol` plots column by column. `mfrow` is therefore likely the most commonly used option. You're going to give `mfrow` vector of length two corresponding to the number of rows followed by the number of columns.

Let's try a basic example with 2 rows and 3 columns:

[1] I describe some other approaches here:
`http://seananderson.ca/2013/10/21/panel-letters.html`

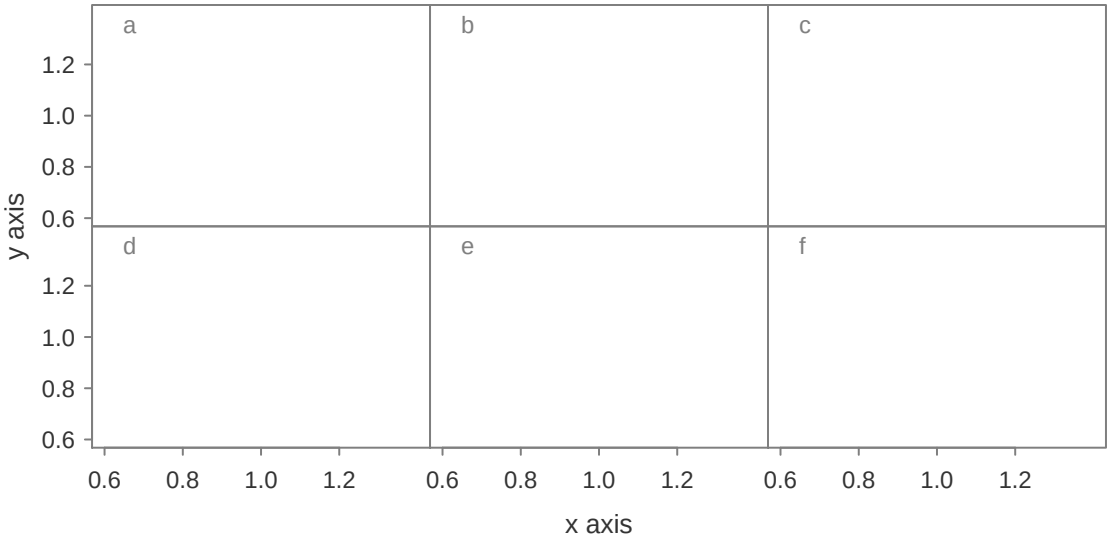
```
par(mfrow = c(2, 3))
par(cex = 0.75)
par(mar = c(3, 3, 0, 0), oma = c(1,1,1,1))
for(i in 1:6) {
  plot(1, 1, type = "n")
  add_label(0.05, 0.15, letters[i])
}
```



We can eliminate the redundant axes, remove margin space, and reduce the emphasis on the structural (non-data) elements of the figure. These are some of the frequent "tricks" you can use to create a basic multipanel layout that will focus the reader's attention on trends in the data. If you aren't familiar with an option for `par()`, look up the help: `?par`.

```
par(mfrow = c(2, 3))
par(cex = 0.75)
par(mar = c(0,0,0,0), oma = c(4, 4, .5, .5))
par(tcl = -0.25, las = 1)
par(mgp = c(2, 0.6, 0))
for(i in 1:6) {
```

```
  plot(1, axes = FALSE, type = "n")
  add_label(0.05, 0.1, letters[i], col = "grey50")
  if(i %in% c(4, 5, 6)) axis(1, col = "grey50", col.axis =
    "grey20", at = seq(0.6, 1.2, 0.2))
  if(i %in% c(1, 4)) axis(2, col = "grey50", col.axis =
    "grey20", at = seq(0.6, 1.2, 0.2))
  box(col = "grey50")
}
  mtext("x axis", side = 1, outer = TRUE, cex = 0.8, line =
    2.2, col = "grey20")
  mtext("y axis", side = 2, outer = TRUE, cex = 0.8, line =
    2.2, col = "grey20", las = 0)
```



8 Fancy multipanel layouts with `layout()`

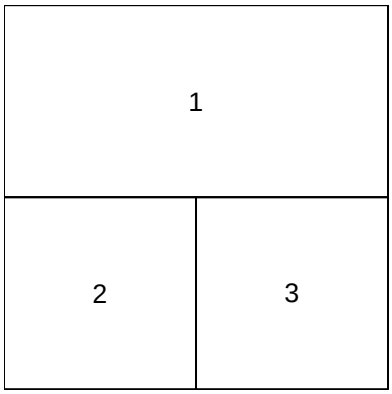
Say you wanted to make a figure with one wide panel on top and two smaller panels underneath. We can't do that easily with `par(mfrow)`? This is where `layout()` becomes useful. It takes a matrix and turns it into a layout. The shape of the matrix corresponds to individual cells. The numbers in the matrix correspond to the order of the plots. Cells with the same number represent a single panel. I often find it easiest to create a matrix with `rbind()` and a series of vectors representing the rows.

First I will use the `layout.show()` command to see how the figure will look. The numbers correspond to the order that the panels will be plotted in.

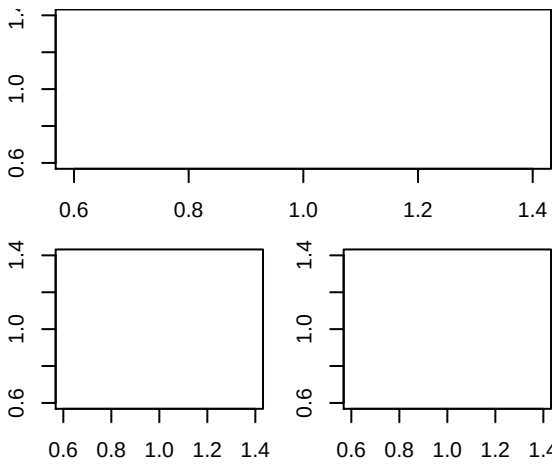
```
m <- rbind(c(1, 1), c(2, 3))
print(m)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    3
```

```
layout(m)
layout.show(3)
```




```
layout(m)
par(mar = c(3, 3, 0, 0), cex = 0.7)
for(i in 1:3) {
  plot(1, 1, type = "n")
}
```

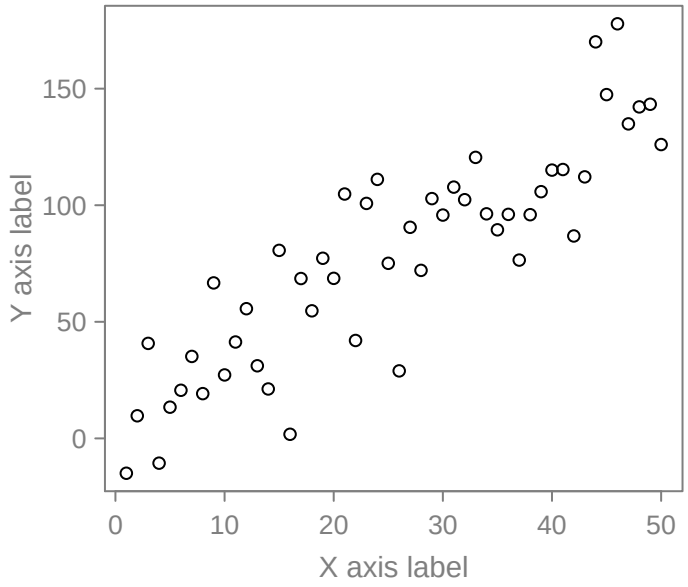


You can create some fairly complex layouts with `layout()` once you realize that numbers that are higher than the number of plots you make will be left empty. They can be used to generate margin space. For example, take this matrix:

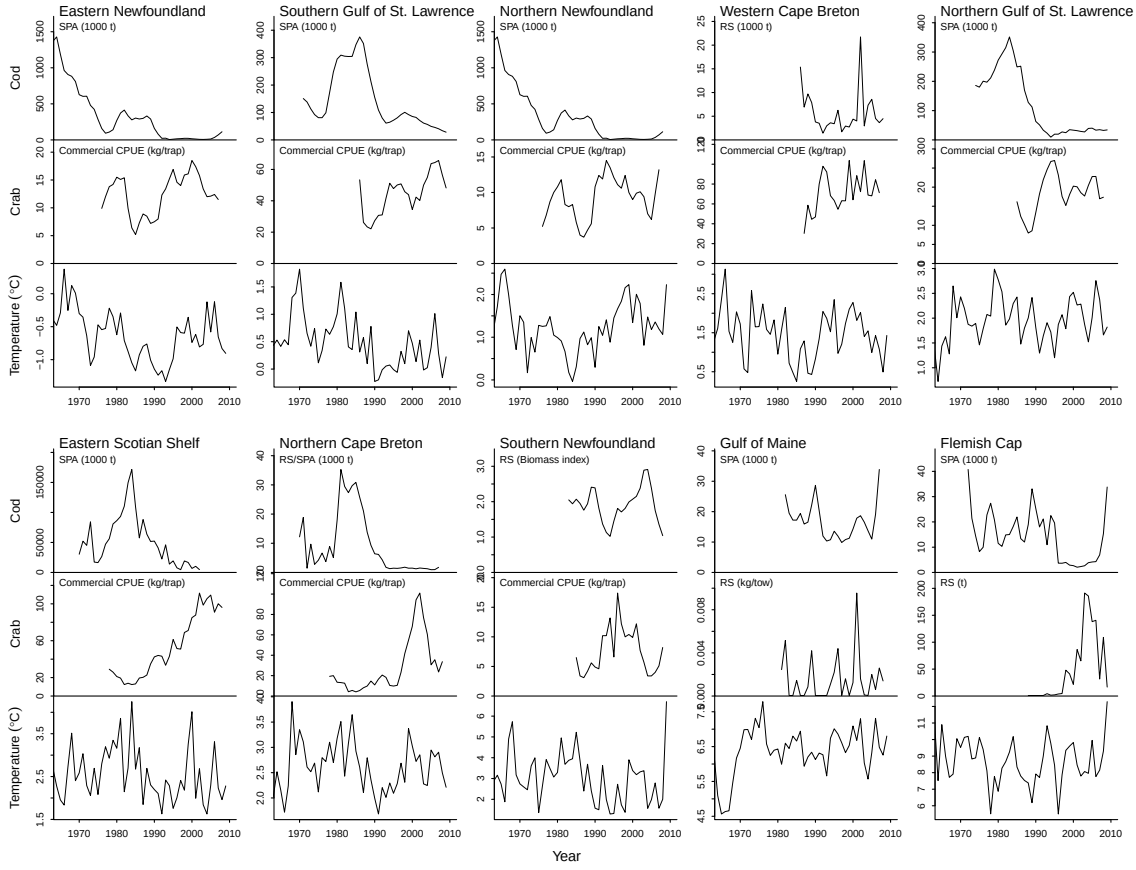
```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]   31   33   35   37   39
##  [2,]    1    4    7   10   13
##  [3,]    1    4    7   10   13
##  [4,]    1    4    7   10   13
##  [5,]    1    4    7   10   13
##  [6,]    2    5    8   11   14
##  [7,]    2    5    8   11   14
##  [8,]    2    5    8   11   14
##  [9,]    2    5    8   11   14
## [10,]    3    6    9   12   15
## [11,]    3    6    9   12   15
## [12,]    3    6    9   12   15
## [13,]    3    6    9   12   15
## [14,]   32   34   36   38   40
## [15,]   41   43   45   47   49
## [16,]   16   19   22   25   28
## [17,]   16   19   22   25   28
## [18,]   16   19   22   25   28
## [19,]   16   19   22   25   28
## [20,]   17   20   23   26   29
## [21,]   17   20   23   26   29
## [22,]   17   20   23   26   29
## [23,]   17   20   23   26   29
```

```
## [24,]   18   21   24   27   30
## [25,]   18   21   24   27   30
## [26,]   18   21   24   27   30
## [27,]   18   21   24   27   30
## [28,]   42   44   46   48   50
```

This matrix can be turned into the layout shown below. Note that there are only 30 panels that are plotted. The rest of the panels make up margin space.



Here's the finished product:



This is about as complicated as it's worth getting with `layout()`. In fact, this would probably be simpler to accomplish with `split.screen()`, which we'll cover next.

9 Crazy multipanel layouts with `split.screen()`

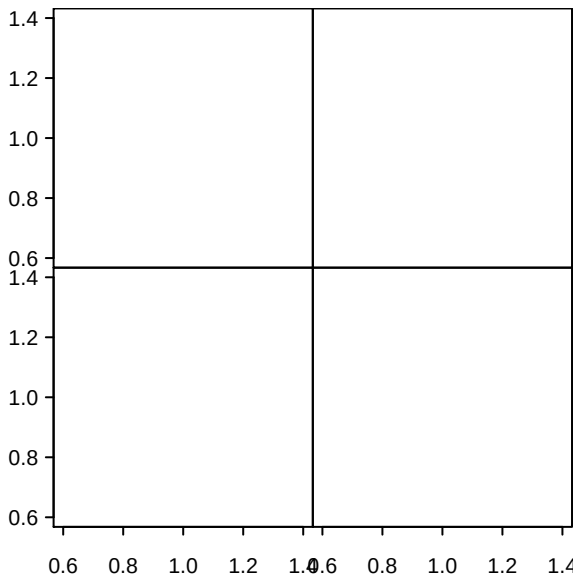
What if you want different panels to be different sizes and the sizes don't correspond to some simple ratio that you could divide up with `layout()`? One common use for this is to keep the unit scales in different panels the same, without using extra margin space. (In other words, the ticks are equally spaced on all panels but the panels are different sizes.) `split.screen()` is limited only by your imagination and ability to figure out the co-ordinates of the panels.

`split.screen()` takes a set of vectors or a matrix to specify the layout. I typically use the matrix notation. In this case each row describes a `screen` with values for

left, right, bottom, and top.[2] 0 corresponds to the left or bottom of the plot. 1 corresponds to the top or right of the plot. Sometimes it's easiest to figure out this matrix in a spreadsheet before and then read it in as a .csv file.

First let's generate a "simple" two-by-two layout with `split.screen()`.

```
m <- rbind(c(0.1, 0.55, 0.55, 1), c(0.55, 1, 0.55, 1),
  c(0.1, 0.55, 0.1, 0.55), c(0.55, 1, 0.1, 0.55))
split.screen(m)


## [1] 1 2 3 4


for(i in 1:4) {
  screen(i)
  par(mar = c(0, 0, 0, 0), cex = 0.7, mgp = c(2, 0.4, 0), las
    = 1, tck = -0.03)
  plot(1, axes = FALSE, type = "n")
  box()
  if(i %in% c(3, 4)) axis(1)
  if(i %in% c(1, 3)) axis(2)
}
```

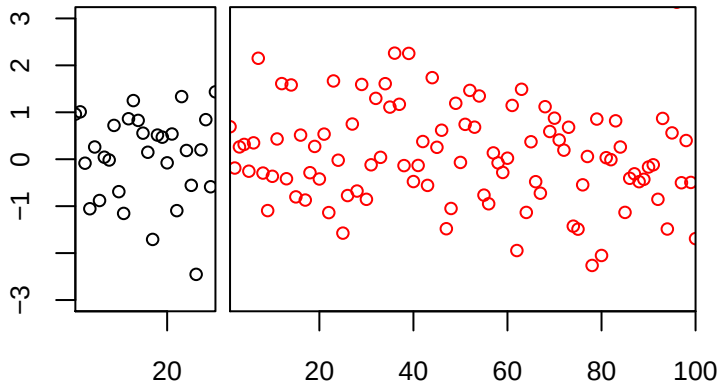


[2]Note that this is different than the order used in `par` statements!

```
close.screen(all.screens = TRUE)
```

Here's an example of setting up the panels to arbitrary sizes so that the units are equal on both x-axes. I figured out these values in a spreadsheet first.

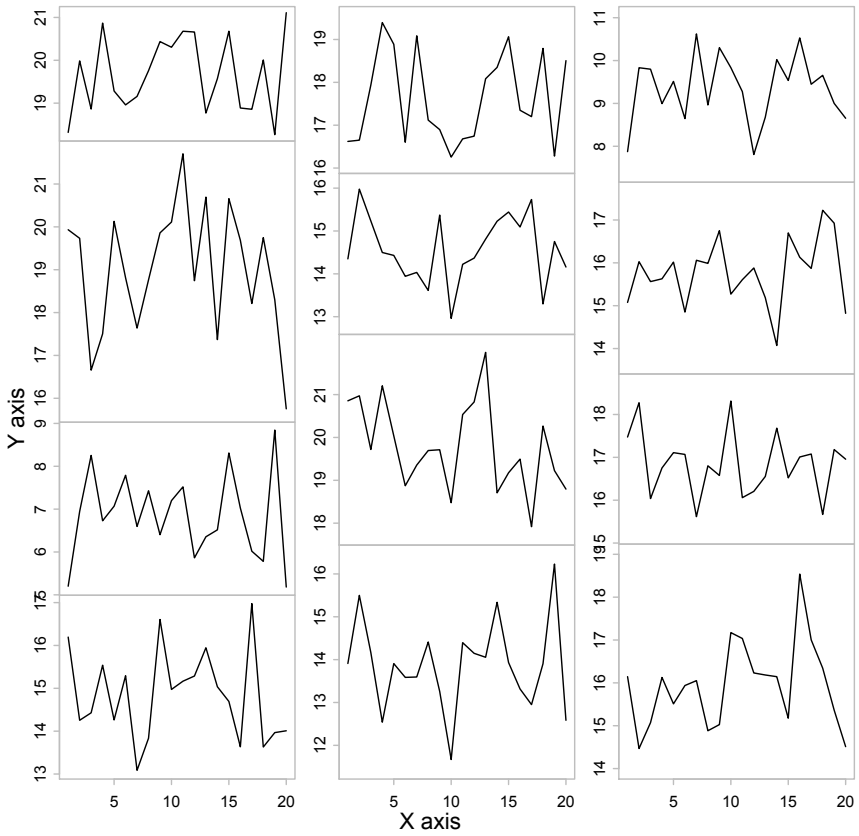
```
split.screen(rbind(c(0.1, 0.292, 0.1, 0.98),
    c(0.312, 0.95, 0.1, 0.98)))

## [1] 1 2

screen(1)
par(mar = c(0, 0, 0, 0), oma = c(1, 1, 0, 0), cex = 0.8)
plot(1:30, rnorm(30), xaxs = "i", ylim = c(-3, 3), xaxt = "n")
axis(1, at = seq(0, 30, 20))

screen(2)
par(mar = c(0, 0, 0, 0), oma = c(1, 1, 0, 0), cex = 0.8)
plot(1:100, rnorm(100), xaxs = "i", ylim = c(-3, 3), yaxt =
  "n", col = "red")
```




```
close.screen(all.screens = TRUE)
```

With `split.screen()` you can apply the same concept to vertical axes. With a bit of planning you can make fairly elaborate layouts.[3]

[3]See Branch et al. 2010, Nature, The trophic fingerprint of marine fisheries, for a great example of this layout.


Y axis
X axis