January 27, 2015

Tony,

I’m sending the code that makes these plots, but I suspect you’ll do it a different way in R, so I’m including this write-up to describe what’s going on here as well.

The basic problem is that we want to make a plot of stacked distributions, colored in. We’ll annotate with labels on the y-axis and CIs to the right of the data…and maybe faint gray lines between sub-national provinces, etc. All of that is window-dressing, for which I’ll share the code, too, but the main problem I focus on here is how to calculate the x & y coordinates for the stacked distributions & how to plot them.

These plots have evolved over that last 8 or 9 months we’ve been making them, and even since our time together in Geneva in November.

Goals

1. We want to plot a stack of distributions representing the confidence levels of estimated binomial proportions.
2. Often the distributions are clipped at the upper and lower (horizontal) limits of the 95% CI, but not always. (We could clip them at some other values)
3. We want each distribution to have equal area on the page.
4. We want the area within a given confidence region to be proportionate. That is to say that if the normalized area of the full distribution is 1, then we want the area inside the 50% CI to be 0.5, within the 10% CI to be 0.1, and within the 90% CI to be 0.9. Anything else would be at least a little misleading.

Remarks

1. The distributions will be asymmetrical when the estimated proportion is not 50%. As it gets closer and closer to 0% or 100%, the distributions will be more and more skewed. This is appropriate for binomial estimation.
2. Distributions that are narrow horizontally (because of a large sample size or a very high or low estimated proportion) will be taller than those that are wider horizontally…because we want them to have equal area.

Approach

We calculate the x and y coordinates for a set of points that outline each distribution, and then select from a variety of methods of plotting. In all the plots you’ve seen, I plotted using a series of stacked horizontal lines. In the last few weeks I’ve found a nicer way in Stata…just what I wanted…a way to specify the x,y coordinates and say ‘for a closed polygon using these points and color it in’. So I’ll switch all my code to use that now and drop the stacked lines approach (which was a pain to jigger around with line thickness and number of lines…)

For each distribution, the conceptual inputs are as follows:

* Effective sample size
* Estimated proportion
* Color for distribution
* Flag to indicate whether to clip the distribution at the 95% CI (where to clip could be an input, as could whether it should be clipped on the left, right, or both, and if so, where)
* Flag to indicate whether to plot a thin theoretical 100% CI from 0% to 100% at the base of the distribution
* Flag to indicate whether to plot ticks for classification (currently always at 95% LCB & UCB, but this could be an input, too)
* More options (as you point out) to control what is labeled at the left and right edges of the plot

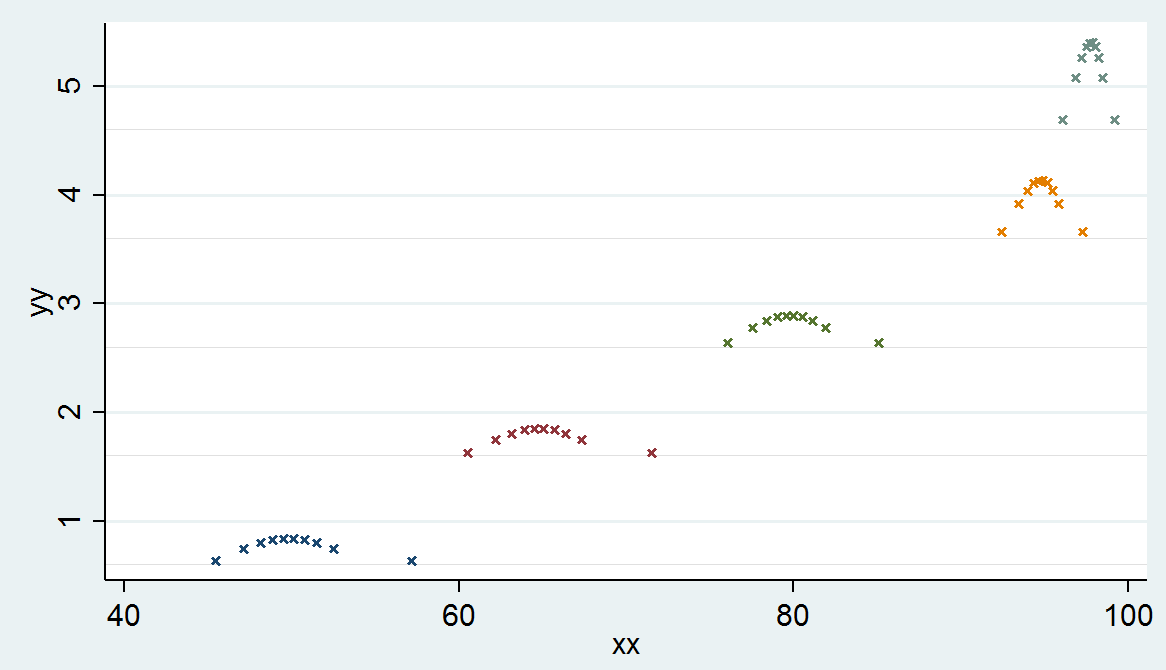
Here are the conceptual steps for calculating the x and y coordinates for a single distribution.

1. Calculate x coordinates

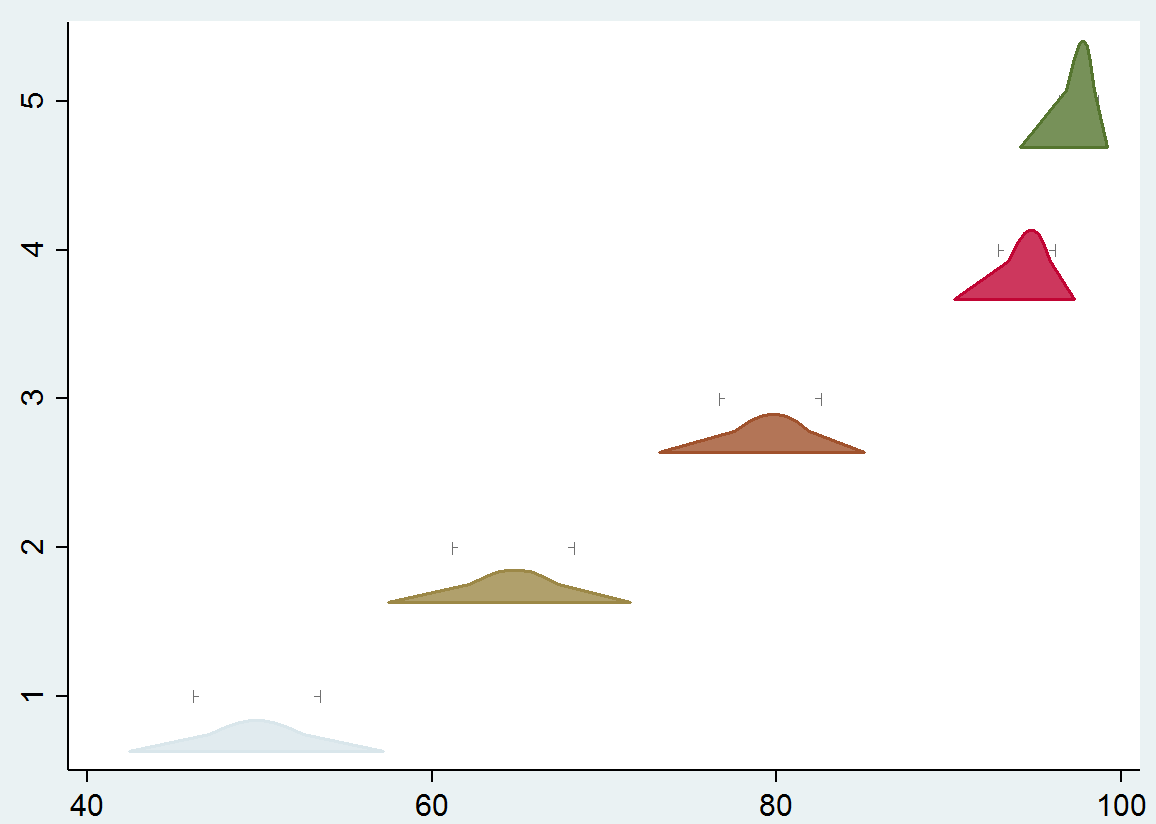
Select a set of values between 10% and 99.9%. Call these values Ai. Let i go from 1 up to some value *NL* like 20 (or more). For each Ai, ask the software to calculate the (asymmetric) limits of the Ai% confidence interval with a specified effective sample size and specified point estimate (or if the survey dataset is in memory…just ask it to calculate the Ai% CI of the proportion in question, using the complex sample design, weights, etc.). A1 is the smallest value…in my code it is always 10%. And ANL is usually 99.9%. The values in between can be anything…I space them equally from 10% up to 99.9%. The only constraint is that each should be bigger than the last. (When calculating CI limits, Stata requires that you ask for at least the 10% CI…nothing smaller…and then you can go up as high as 99.99%.)

For each i, you get a lower and upper x value. In my code I call them li and ui. These will be the x values for plotting. They will be farther and farther apart from each other for successively larger values of i (and correspondingly larger values of Ai.) If you could ask for the limits of the 100% CI, those limits would always be (0%, 100%).

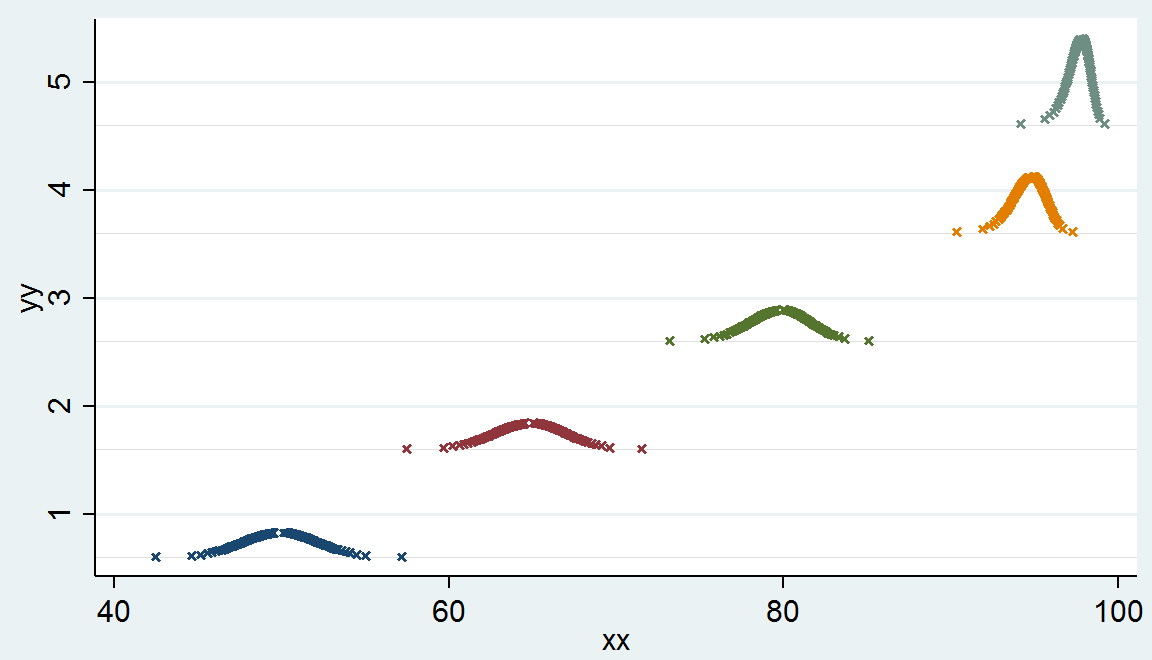
Here’s a scatter plot where NL is 5; so each distribution is represented by points at 5 different y values. There are two x-values for every y-value…the top-most points in each distribution are at the outer corners of the 10%CI. The bottom-most points are at the outer corners of the 99.9% CI. In practice I usually use a value of NL that is higher than 20, so the distributions will be very smooth. In later steps we’re going to connect these dots.

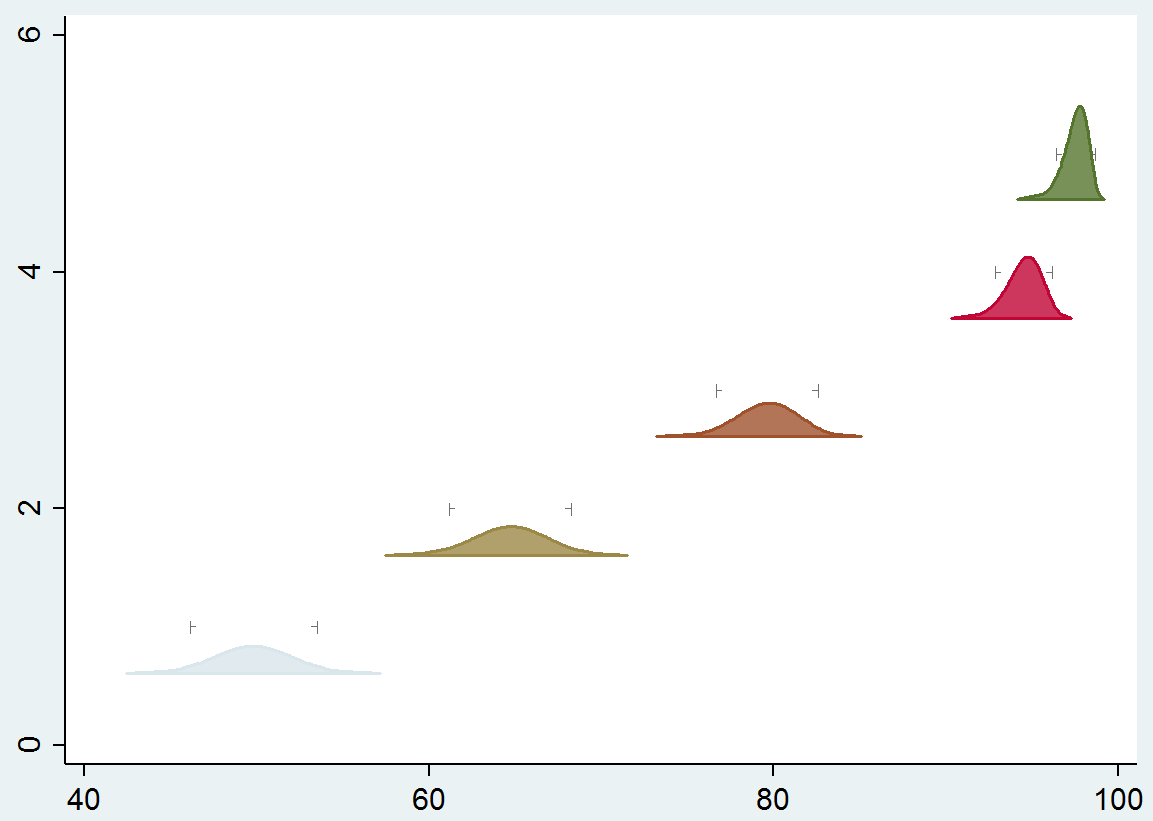


Here are the dots connected, when NL = 5.



And here are the same distributions with NL = 50:

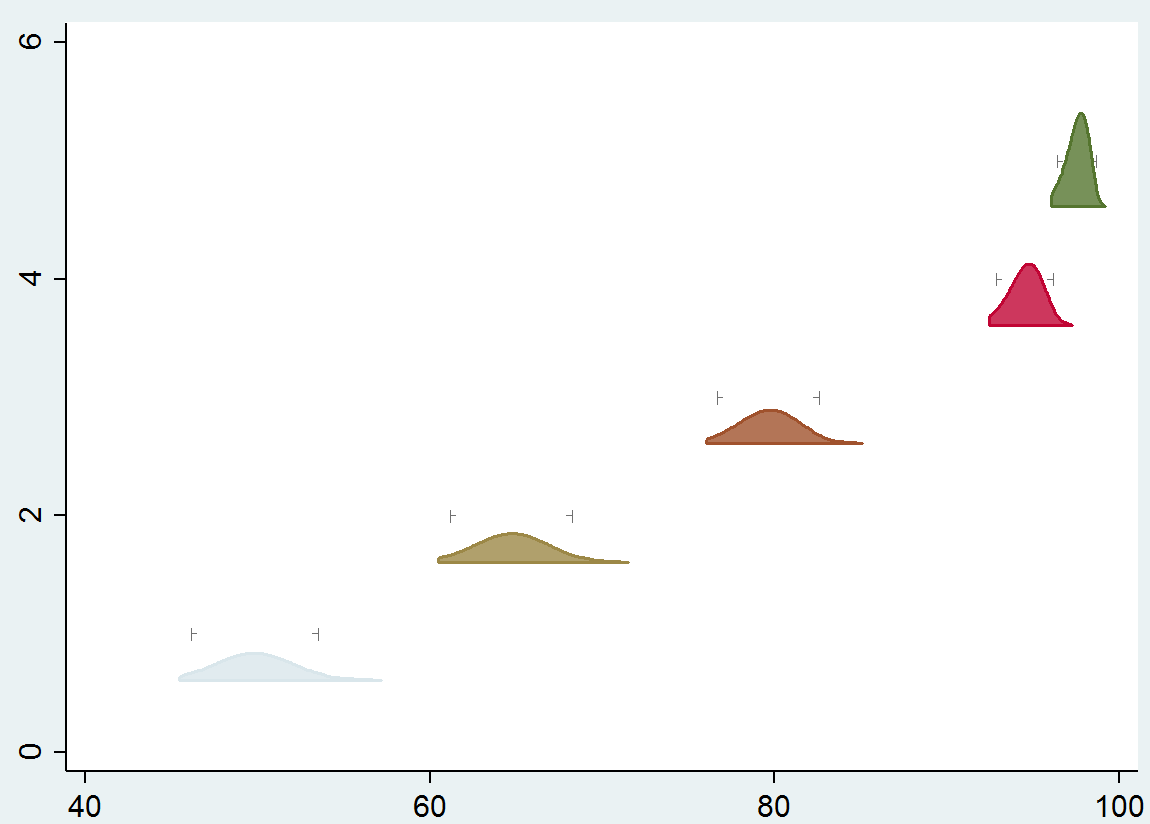




Clearly, the more values you calculate, the better the representation of the shape of the distribution in those parts where there is inflection in the curves. I like a value around 50. That means there are 50 x,y pairs that define the left side of the distribution and 50 x,y pairs that define the right side of the distribution…each blob is outlined by 100 x,y pairs.

Presumably R has a way to calculate confidence intervals after a survey proportion has been estimated, so you can calculate the x-coordinates in a similar manner, for a set of Cis ranging from something like 10% up to something like 99.9%.

No matter what the value of NL, my code always calculates the x coordinates that define the limits of the 95% CI and the 90% CI, because those x-coordinates come in handy later for a) clipping the distribution at the 95% CI, and plotting the classification ticks at the 90% CI. Here’s the NL=50 blob plot, with the distributions clipped at the 95% CI. (If we make the clipping locations user-specified, then we’ll just make those two numbers (90% and 95%) user defined…no problem.)



1. Calculate unscaled y-coordinates

Here we use the property that we want the distributions to have equal area. If we think of each value of i as defining a rectangle in the distribution…there is a 10% rectangle…this is the tallest rectangle in the distribution…then if A2 is 15% then there is a shorter wider 15% rectangle that extends to the left and right of the tall 10% rectangle. If the distribution is asymmetrical, then the 15% rectangle is offset with respect to the 10% rectangle. But the area of the 10% rectangle is 0.1 and the additional area of the 15% rectangle, to the right and left together, is .05. (The rectangles are an approximation…I’m ignoring the little triangles that would sit on top of the 15% rectangle and reach up to the top of the 10% rectancgle…we can add in the area of the triangles in a later step…it’ll just change the formula for y a very little bit…I’m blissfully ignoring it for now.) We use the fact that we know the width of each rectangle (from the x coordinates calculated above) and we know its area, to calculate its y-value. First we’ll calculate y-values that are not stacked…for each distribution in isolation…that assume that each distribution rests on a base at y = 0. Later we’ll re-scale the y values and shift them up so they are stacked.

So calculate the y-value for A1 and then loop from 2 up to NL and calculate the y-values of each of the i rectangles.

Y1 = a1 / (u1-l1)

Yi = (ai – ai-1) / ( (ui – li) – (ui-1 – li-1) ) for values if i > 1

1. Scale the y-coordinates

So now we have our x-coordinates…they fall between 0% and 100% because we used an appropriate binomial confidence interval generator that keeps the CI in those bounds. But our y-coordinates need to be scaled. If we are stacking 5 distributions, then I think of each one as being centered on the y integers 1, 2, 3, 4, and 5. I (arbitrarily) put the base of each distribution at the y-coordinate that is 0.4 points below the integer. And I scale the y-values so that the tippy-tallest distribution extends to a height 0.4 points above the integer value. So the space between 0.4 and 0.6 for each integer is always white space. If some distributions are taller than others then there will be even more white space.

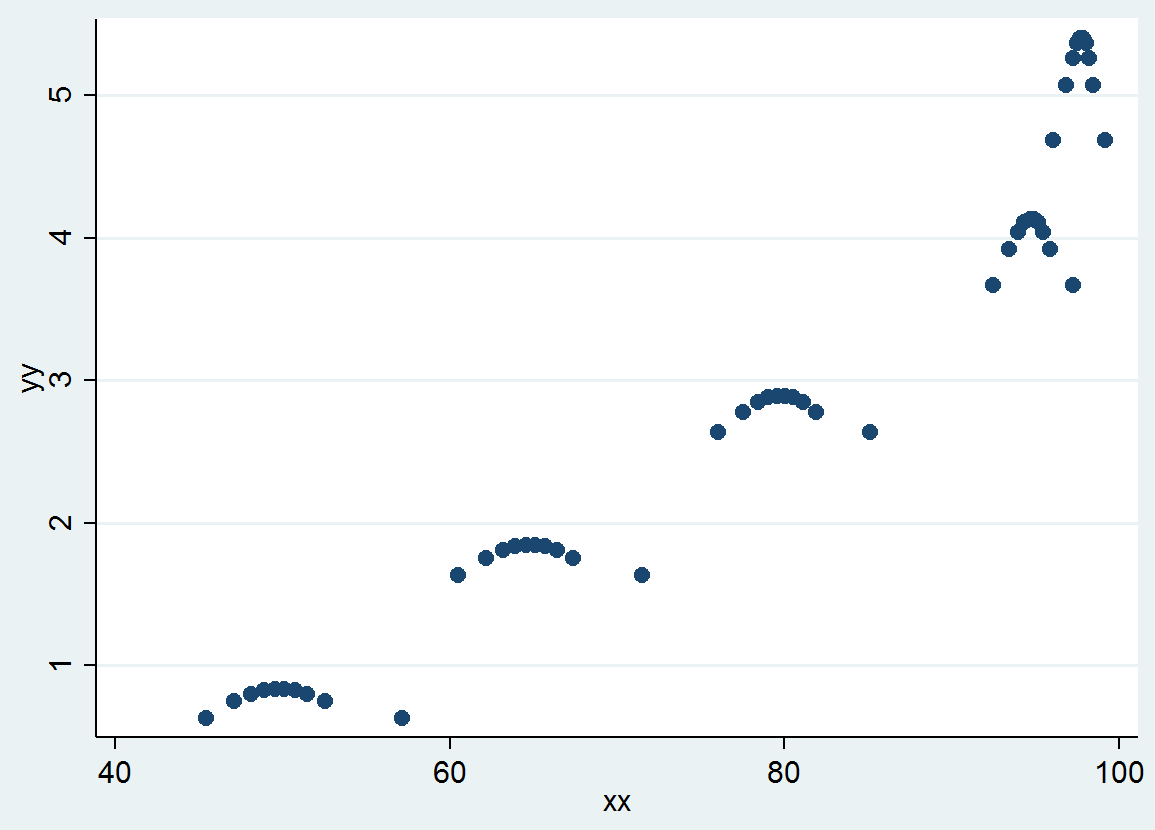
You have to figure out how you want to order the distributions in the stack…maybe you sort by estimated coverage…or maybe you sort the provinces by estimated coverage and within a province you sort the districts by estimated coverage. The sorting of the distributions is already handled…and then you re-scale the y-axes and add the appropriate values to get their bases to fall at their integer value minus 0.4.

Calculate the maximum y value across all distributions…call it y\_max. (It will be one of the values of y1.). Scale all the other y’s like so:

Scaled\_yi = unscaled\_yi \* 0.8 / y\_max   
( Note that 0.8 is the maximum height of any plotted distribution…from integer minus 0.4 up to integer plus 0.4)

1. Stack the y-coordinates

Now everything is scaled…the tallest distribution goes from y=0 up to y=0.8. Now simply add the appropriate (Y-1).6 value to move each distribution up to its appropriate spot in the stack. So all the y values for the first distribution get 0.6 added to them. All the y values for the second distribution get 1.6 added to them…etc. Now you have x,y pairs for each distribution such that if you just plot y versus x (in my code it is yy versus xx) then they look like this:



1. Now plot them, using code that connects the outline of each distribution, and colors the distribution according to your wishes..based on whatever classification scheme you’re using. In stata, I use the command ‘graph twoway area’ command to plot the blobs. I can specify a color for each. I can overlay the point estimate as a vertical bar in the distribution. I can add the classification ticks at the integer values of y, and the x-coordinates of the lower- and upper-limits of the 90% CI.

All the other stuff is just window dressing for the plot…using stratum names at the left…CI strings at the right…bold vertical lines at the programmatic coverage threshold.

The classification ticks at the LCB and UCB…they could be plotted with symbols. The x-coordinates are the bounds of the 90% CI and the y-coordinates are the integer values of y. (Because some distributions are not very tall, under my new way of plotting equal-area distributions, the ticks sometimes float up above the distribution…that’s okay with me…I’d rather have the ticks appear at regular y intervals than always plot them halfway up the side of the distribution…lower for short distributions and taller for tall ones. But that’s perhaps a personal preference.)

Anyway…I don’t plot them using symbols…I plot them using two lines…the first line has caps, so it looks like this:

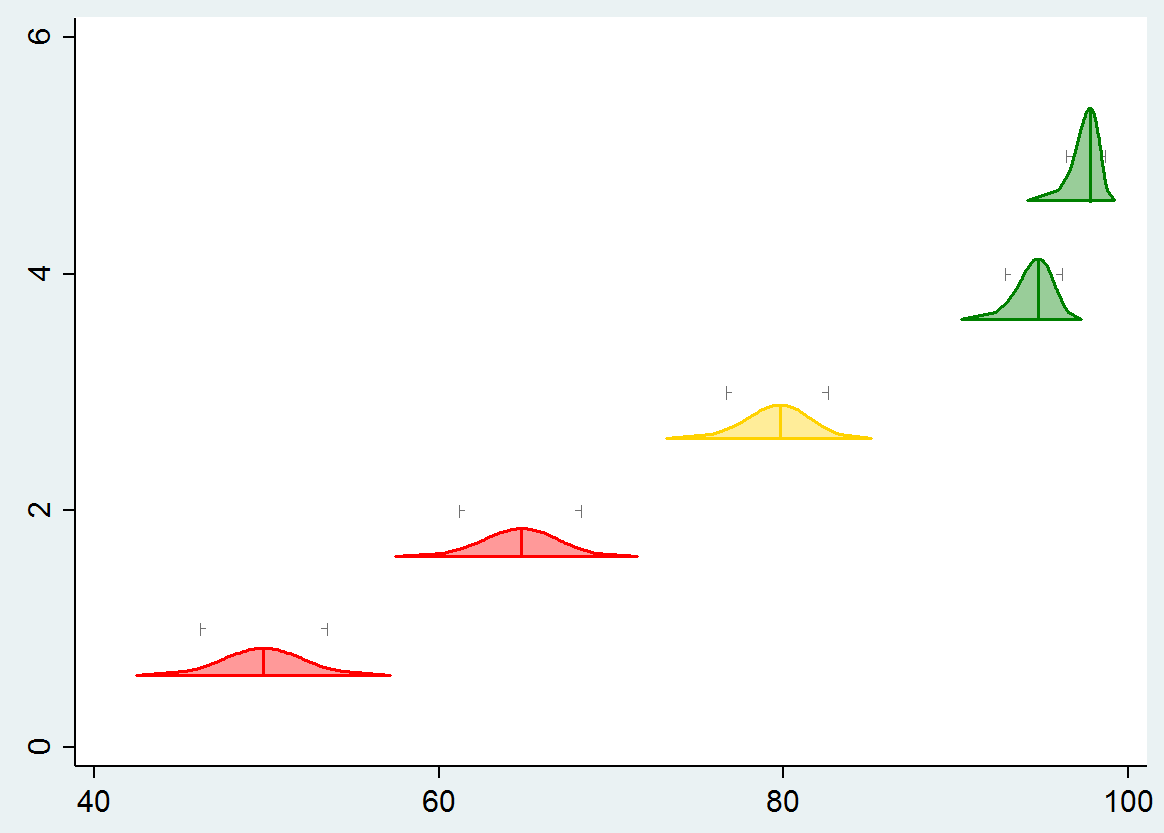
|----------|

and then I plot a white line inside with no cap, to cover most of the horizontal line, so we wind up with this:

|- -|

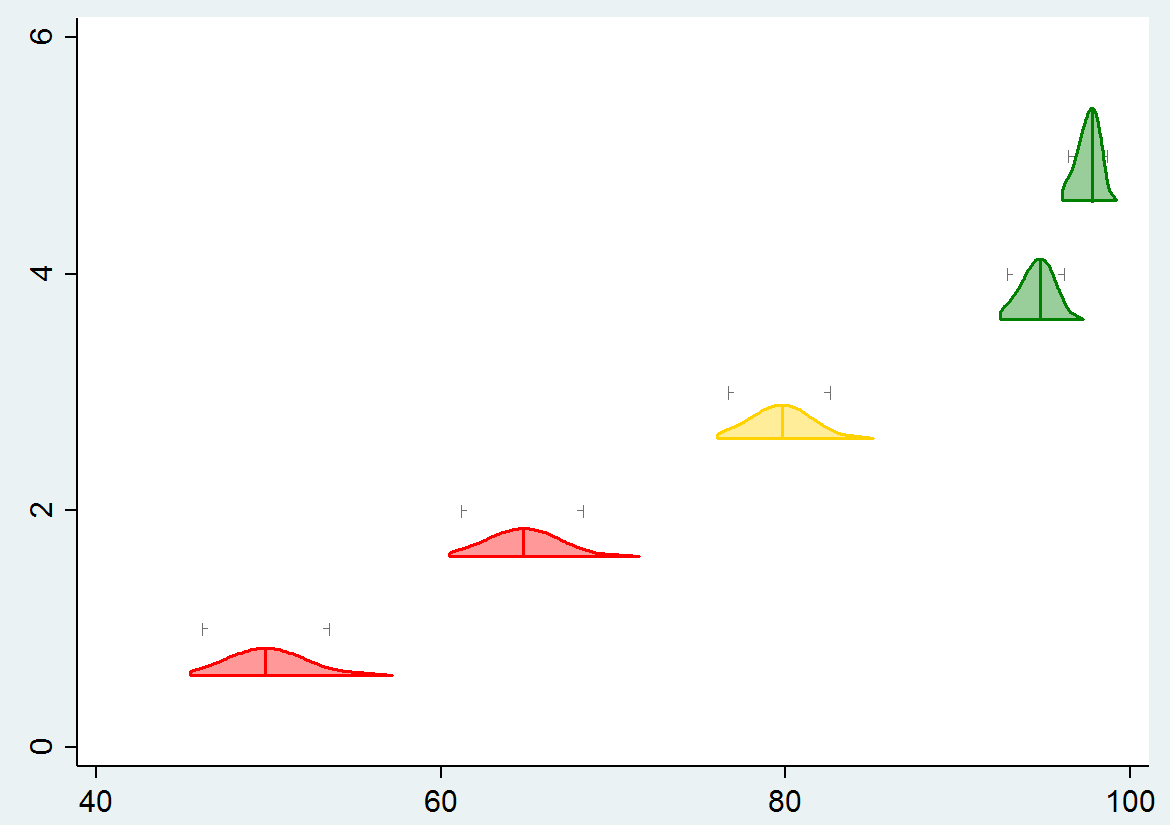
So conceptually, the plotting program does these things in about this order:

1. Plot the horizontal 90% CIs, using the capped lines, in gray.
2. Plot horizontal lines just inside the 90% CIs, in white, to cover most of the horizontal gray lines.
3. Plot lines that bound the polygon for each distribution, and color it in.
4. Overlay a vertical line on each distribution, to indicate the location of the point estimate.



(These go out to the 99.9th percentile…I’m thinking of adding an option to plot a line all the way from 0% to 100% just underneath the distribution, to drive home the fact that the 100% CI extends all the way across the x-axis.)

And if the user specifies to clip the distributions, it looks like this:



Here’s the code that makes that plot.

graph twoway ///

(rcap lb\_90pct ub\_90pct y , sort horizontal lcolor(gs7) lwidth(\*0.5)) /// // classification ticks

(rspike mfudgelo mfudgehi y , sort horizontal lcolor(white) lwidth(\*0.5)) /// // cover lines, leaving only ticks

(area yy xx if y == 5 & j > 0, fcolor(green\*0.5) lcolor(green) nodropbase ) /// // plot the distribution at y=5

(area yy xx if y == 4 & j > 0, fcolor(green\*0.5) lcolor(green) nodropbase ) /// // ditto y=4

(rspike ypemin ypemax vcvg if color == 1 & j == 1, lcolor(green)) /// // overlay green point estimate lines

(area yy xx if color == 2 , fcolor(gold\*0.5) lcolor(gold) nodropbase ) /// // plot the distribution for y=3

(rspike ypemin ypemax vcvg if color == 2 & j == 1, lcolor(gold)) /// // overlay gold point estimate line

(area yy xx if y == 2 & j > 0 , fcolor(red\*0.5) lcolor(red) nodropbase ) /// // plot distribution for y=2

(area yy xx if y == 1 & j > 0 , fcolor(red\*0.5) lcolor(red) nodropbase ) /// // ditto y=1

(rspike ypemin ypemax vcvg if color == 3 & j == 1, lcolor(red)) /// // overlay red point estimate lines

, legend( off) ylabel(,nogrid) // twoway plot options

This code is a little cheesy because it explicitly has a line for each distribution and hardwires the colors…that’s just because this is a toy example…in a bigger program like one to plot the Ethiopia data, I would have the program write those lines of code, based on the classification / color criteria, and then execute the code that it just wrote.

I’ll send you the full Monty program to make the Ethiopia plot soon…I’m retooling it to have equal areas.

This code does NOT include the window dressing of labels, CIs, etc…this is the basics…but as I say, I think this is the heart of it.

Agree with you that we’ll package this up inside a program with options to give people flexibility over what they see.

Hope this is helpful. Don’t hesitate to ask questions.