Complex Figures in Base R

James Meadow

Base R is among the most powerful software packages for static data visualization, but most users are unaware of the many options for creating complex, publication-quality figures. This script lays out just a few of the many, many ways to combine datasets into complex, multipanel figures.

The data exlored here are downloaded from the World Bank data repository, which is enabled by the `rWBclimate` package from Scott Chamberlain and the ROpenSci organization.

First, install the packages below.

- The `maps` and `mapdata` packages make plotting maps really nice and simple.
- The `devtools` package is a necessity for advanced R analysis. It has a really nice interface for installing packages from GitHub, among other repositories.
- Then the `rWBclimate` from GitHub. This requires the `devtools` packages be installed, so do that first.

Notice that each `install.packages` command is commented out so that Rstudio doesn't install them all each time the script is run. It is best to run each of those commands manually in the console, then they will be on your system each time you load them with the `library` command.

```
# install.packages('maps')
# install.packages('mapdata')
library(maps)
library(mapdata)
# install.packages('devtools')
library(devtools)
# install_github('ropensci/rWBclimate')
library(rWBclimate)
```

We can get some open-access data from the World Bank database with the `get_historical...` commands. For this example, we want data for Mexico, Guatemala, Honduras, and Costa Rica.

```
country.list <- c("MEX", "GTM", "HND", "CRI")
country.precip <- get_historical_precip(country.list, "year")
country.temp <- get_historical_temp(country.list, 'year')
```

system('head precip.txt')

Check out the datasets to make sure they look right.

```
kable(head(country.precip))
```

year	data	locator
1901	53.69402	MEX
1902	50.42450	MEX
1903	55.96359	MEX
1904	55.00967	MEX
1905	61.11870	MEX
1906	57.21757	MEX

```r
kable(head(country.temp))
```

year	data	locator
1901	20.52810	MEX
1902	20.80419	MEX
1903	20.17612	MEX
1904	20.64074	MEX
1905	20.43752	MEX
1906	20.48463	MEX

And then run a couple of checks to see that they are the same format and can be safely combined.

```r
dim(country.precip)
```

```
[1] 436   3
```

```r
dim(country.temp)
```

```
[1] 436   3
```

```r
identical(country.precip$year, country.temp$year)
```

```
[1] TRUE
```

```r
identical(country.precip$locator, country.temp$locator)
```

```
[1] TRUE
```

Since they are identical except for the `data` column, we can rearrange the columns to combine them into a single data frame.

```r
names(country.precip)[2] <- 'precip'
country.precip$temp <- country.temp$data
head(country.precip)
```

```
  year   precip locator     temp
1 1901 53.69402     MEX 20.52810
2 1902 50.42450     MEX 20.80419
3 1903 55.96359     MEX 20.17612
4 1904 55.00967     MEX 20.64074
5 1905 61.11870     MEX 20.43752
6 1906 57.21757     MEX 20.48463
```

```
dat <- country.precip[, c('year', 'locator', 'precip', 'temp')]
```

Before plotting, set up a few global variables that will save lots of unnecessary code repetition later on.

- Get the range of temperature and precipitation values - this is useful for setting the limits of the plots.
- Get an index for each country's data with the `which` command.
- Put those index values in a list to call later - sometimes in a loop of 4 countries, and sometimes individually by calling countries by name.
- Also create a vector of the 4 country names. Those will be useful for plotting names and when creating maps.
- Pick 4 nice colors, one for each country, in the same order as the `countries` vector. R knows 657 colors by name, and you can see them all by entering the `colors()` command.
- Finally, pick out a few evenly spaced years that will make for a nice, minimal x-axis.

```
temp.range <- range(dat$temp)
precip.range <- range(dat$precip)
mex <- which(dat$locator == 'MEX')
gtm <- which(dat$locator == 'GTM')
hnd <- which(dat$locator == 'HND')
cri <- which(dat$locator == 'CRI')
countries <- list(mex, gtm, hnd, cri)
countryNames <- c('Mexico', 'Guatemala', 'Honduras', 'Costa Rica')
cols <- c('darkorange', 'olivedrab4', 'cornflowerblue', 'cyan2')
yearAxis <- c(seq(1900, 2000, 20), 2009)
```

Separate plots

The first attempt will be to plot the lines, each in their own plot. This is useful sometimes when there is no need to plot the lines all together in the same plot. In this example, the `mfrow` option is set in the `par` command at the top. This divides the plot into 4 equal spaces, with 4 rows and 1 column. Another much more flexible way to do this would be the `layout` command, which is better when the individual areas should not be the same size.

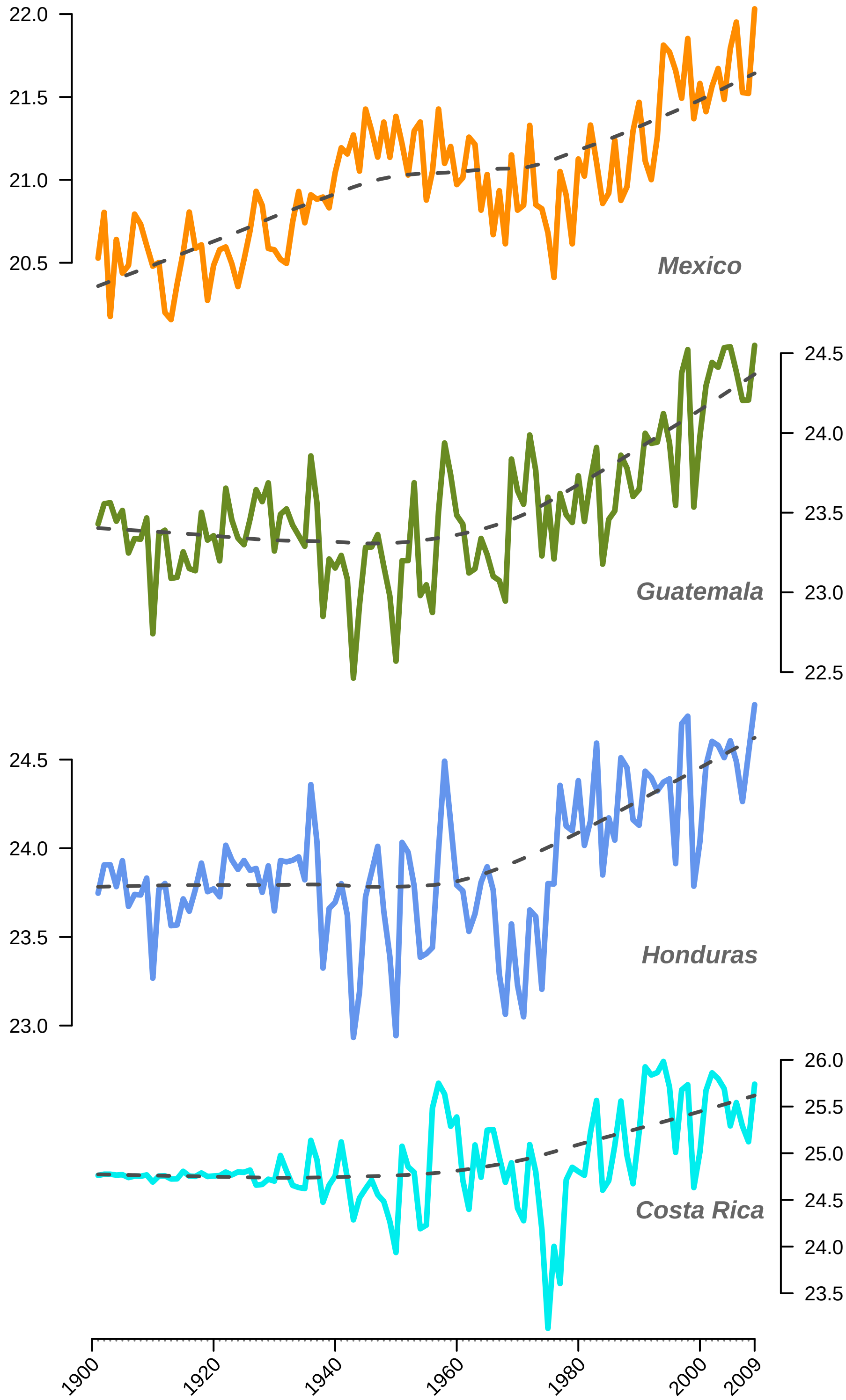
- The `par` command splits up the plotting space into 4 equal parts.
- Then create a `mars` list that holds the margins for each of the 4 plots. Each one is called in turn during the loop below it.
- Loop through and create each of the 4 plots. This uses several of the options that were set in the setup commands above. This is where a few lines earlier really saves unnecessary repetative coding.
- Set a few parameters with `par` and `these`. This object called `these` will get used often, so it is important to set it at the top and avoid repeating the code over and over.

- Create the plot, and include the line, but no axes. These get added by hand next.
- The `if` statements put the y-axes on either the left or right sides. That way they are not cluttered on one side.
- Add country names to each line. We could add a legend, but it is usually better to avoid the legend and mark up the plot itself with any labelling.
- Add a simple loess curve that acts similar to a running average. There are lots of more statistically robust ways to model climate, but this is just an example of adding a curve to the plot.
- Finally add the bottom x-axis to the bottom plot by hand.

```r
par(mfrow=c(4, 1))

mars <- list(c(0, 4, 1, 4),
             c(0, 4, 0, 4),
             c(0, 4, 0, 4),
             c(3, 4, 0, 4))

for(i in 1:4) {
  par(mar=mars[[i]])
  these <- countries[[i]]
  plot(dat$temp[these] ~ dat$year[these],
       type='l', lwd=3, col=cols[i],
       axes=FALSE, xlab='', ylab='')
  if(i %in% c(1, 3)) axis(2, las=1)
  if(i %in% c(2, 4)) axis(4, las=1)
  text(2000, mean(dat$temp[these])-.5,
       countryNames[i],
       col='gray40', font=4, cex=1.3)
  points(loess.smooth(dat$year[these], dat$temp[these]),
         type='l', lty=2, lwd=2, col='gray30')
  }
axis(side=1, labels=FALSE, at=unique(dat$year),
     tck=-.006, col='gray50')
axis(side=1, labels=FALSE, at=yearAxis)
text(x=yearAxis-2, y=par()$usr[3]-.4,
     yearAxis, srt=45, xpd=TRUE)
```



If the goal is to see that temperatures are rising across the area, this does the job, but it is pretty boring. The lines aren't plotted together to give each context, and we have only one data type in the figure. This might be a missed opportunity for a more data-rich figure.

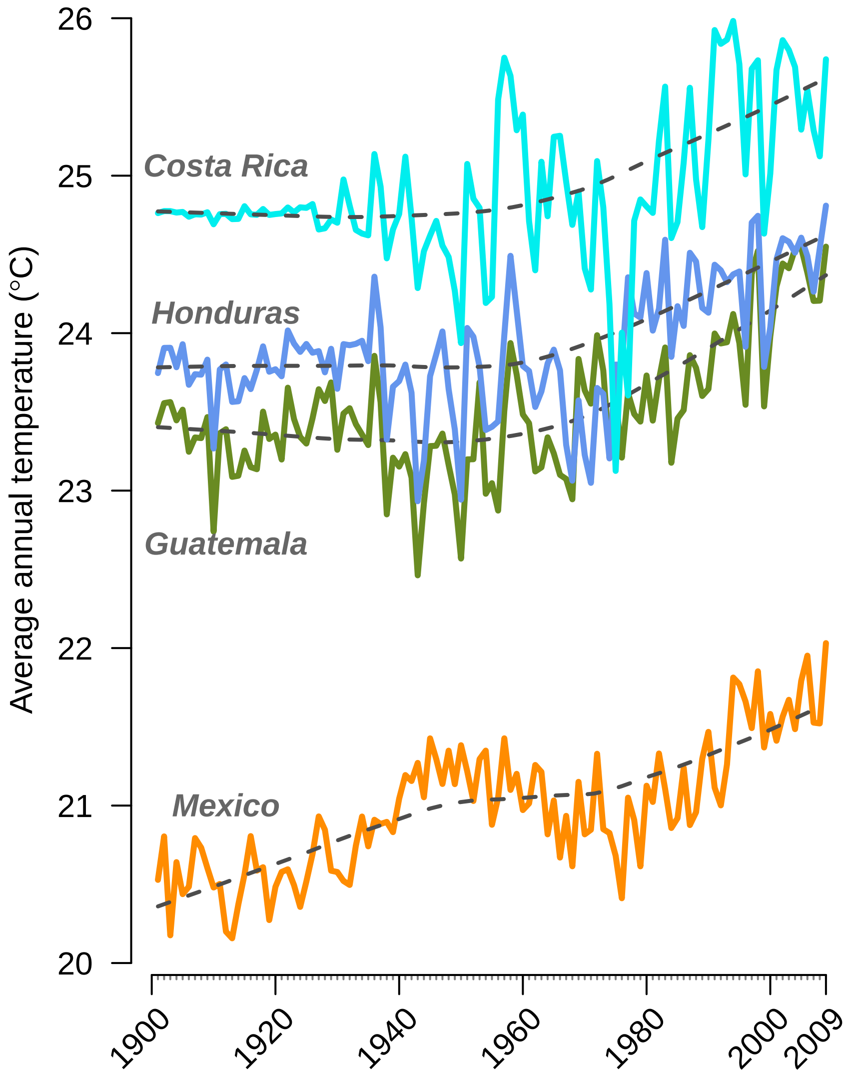
Plotting lines together

Since these countries are on a lattitude gradient, we can plot the lines together and see them compared to one another. This time we only create one plot, but add lines to it one at a time. There are lots of ways to automate the process without the `for` command, but this makes it easy to customize and easily readable.

- Set an offset vector for plotting the names.
- Make the plot, but make it empty. Lines will get added one at a time.
- Loop through 4 countries and add lines and labels
- `these` subsets the data by country.
- `points` adds the lines
- Add the country names with `text`
- and add the loess curve for each country.
- Add an axis to the bottom, but make it with a series of 3 commands that each puts on a small piece.
- Add the y-axis, and the axis label with the `mtext` command.

```
countryNamesOffset <- c(.4, -.8, .3, .3)
plot(dat$temp ~ dat$year,
     type='n', bty='n', axes=FALSE, ann=FALSE)
for(i in 1:length(countries)) {
  these <- countries[[i]]
  points(dat$temp[these] ~ dat$year[these],
         type='l', lwd=3, col=cols[i])
  text(1912, dat$temp[these][9] + countryNamesOffset[i],
       countryNames[i], font=4, col='gray40')
  points(loess.smooth(dat$year[these], dat$temp[these]),
         type='l', lty=2, lwd=2, col='gray30')
  }

axis(side=1, labels=FALSE, at=unique(dat$year),
     tck=-.006, col='gray50')
axis(side=1, labels=FALSE, at=yearAxis)
text(x=yearAxis-2, y=par()$usr[3]-.4,
     yearAxis, srt=45, xpd=TRUE)
axis(2, las=1)
mtext(expression(paste('Average annual temperature (', degree, 'C)', sep='')),
      side=2, line=2.2)
```



So that is more dense and gives better context for each country.

Precipitation data as polygons

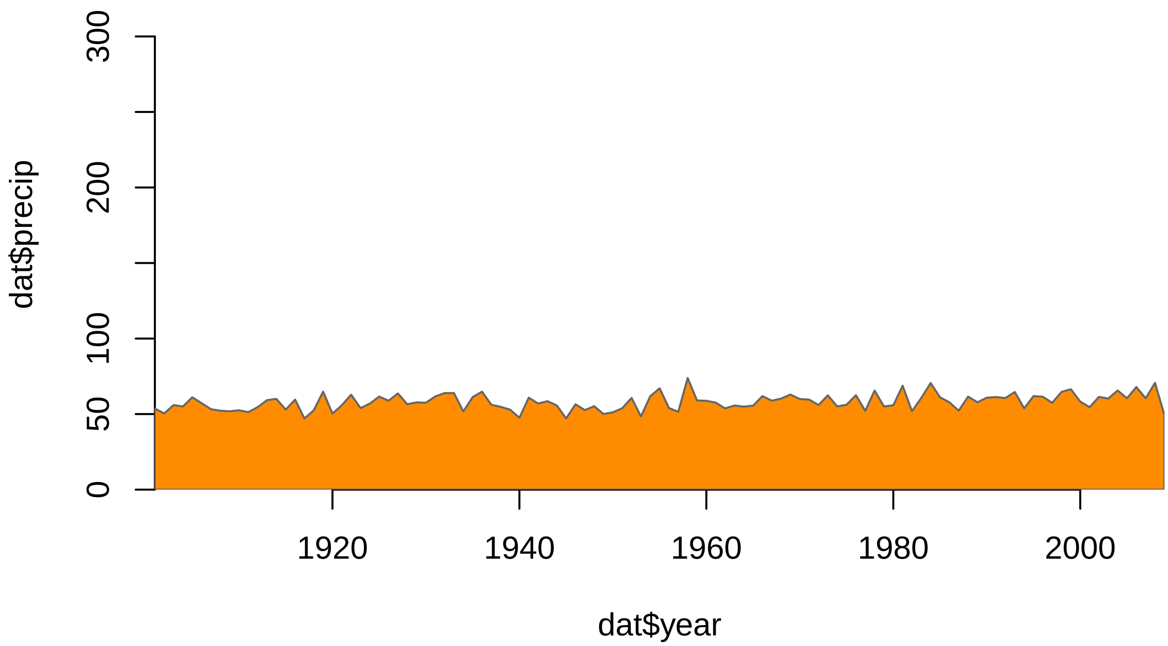
Polygons are a really great way to shade backgrounds, show confidence intervals, or show depth and accumulation. So that might be a good way to visualize the precipitation data so that it stands out from the temperature data.

Each polygon is a vector of x-coordinates and y-coordinates, and they must be lined up.

- The x-coordinates are simply the years + the same years in reverse.
- The y-coordinates are the actual data values + a string of zeros for the bottom edge.

Here is an example with Mexico's data.

```
plot(dat$precip ~ dat$year, type='n',
     xaxs='i', yaxs='i', bty='n',
     ylim=c(0, max(dat$precip)))
polygon(c(dat$year[mex], rev(dat$year[mex])),
        c(dat$precip[mex], rep(0, length(mex))),
        col=cols[1], border='gray40')
```



To scale up this plot, we'll plot the data one on top of another, so they have to be in a slightly different order.

- Reorder the countries in a new sorted list, highest to lowest rainfall.
- Also reorder the colors.
- Make a small dataset of country means with the `aggregate` function. This is used to plot the average values at the far right edge.
- Then round those values to 2 decimal points.

The next part is really important. We'll use this code again below, so there is no reason to duplicate the whole thing. Put all of the plotting code in its own function, and then the function can be called wherever it is needed without writing lots of repatative code. This will also be done for the temperature data in the next step.

Within the function:

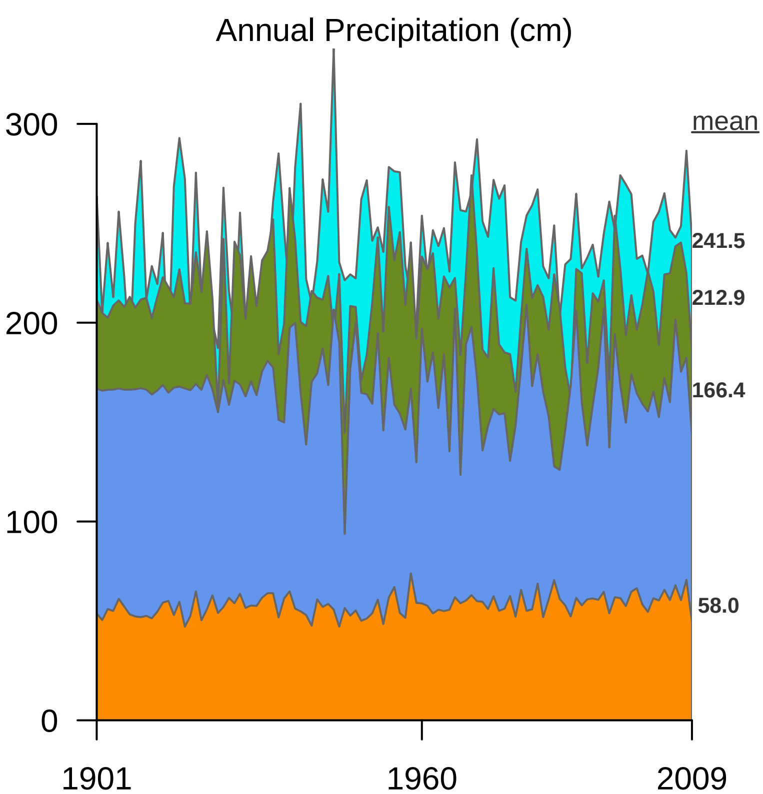
- Make the empty plot.
- Make each polygon with the `for` loop.
- Make a simple axis since this plot will shrink in the final version.
- Add the mean values to the right side.
- Make an x-axis.

Then call the function and the plot gets made!

```r
countriesSort <- list(cri, gtm, hnd, mex)
colSort <- c('cyan2', 'olivedrab4', 'cornflowerblue', 'darkorange')
precipMeans <- aggregate(dat$precip, by=list(dat$locator), FUN='mean')
meanLabels <- format(precipMeans$x, digits=4)

plotPrecip <- function() {
  plot(dat$precip ~ dat$year, type='n',
       axes=FALSE, ann=FALSE, xaxs='i', yaxs='i',
       ylim=c(0, max(dat$precip)))
  for(i in 1:4) {
    polygon(c(dat$year[countriesSort[[i]]], rev(dat$year[countriesSort[[i]]])),
            c(dat$precip[countriesSort[[i]]], rep(0, length(countriesSort[[i]]))),
            col=colSort[i], border='gray40')
    }
  axis(2, las=1, at=c(0, 100, 200, 300))
  mtext('Annual Precipitation (cm)', side=3, line=0)
  mtext(expression(underline('mean')), side=4, at=300,
        las=1, font=2, col='gray20', line=0, cex=.8)
  mtext(meanLabels, side=4, at=precipMeans$x,
        las=1, font=2, col='gray20', line=0, cex=.7)
  yearSubs <- c(1901, 1960, 2009)
  axis(side=1, labels=yearSubs, at=yearSubs)
  }

par(mar=c(4, 8, 1, 9))
plotPrecip()
```



Temperature lines function

Here is the same function idea applied to the line plots. These will be repeated on the big final figure, so it is best to make a function with a few minor options, and then call up the one-line function wherever it is needed. This is nearly the same plot used above, but with some fancy options thrown in.

```
makeLines <- function(these=these, i=1) {
  plot(dat$temp[these] ~ dat$year[these],
       type='l', lwd=3, col=cols[i],
       axes=FALSE, xlab='', ylab='', bty='l')
  axis(2, las=1)
  text(1980, par()$usr[3],
       countryNames[i], pos=3,
       col='gray40', font=4, cex=1)
  points(loess.smooth(dat$year[these], dat$temp[these]),
         type='l', lty=2, lwd=2, col='gray30')
  yearSubs <- c(1901, 1960, 2009)
  axis(side=1, labels=yearSubs, at=yearSubs)
  maxTemp <- which(dat$temp[these] == max(dat$temp[these]))
  points(dat$year[these][maxTemp], dat$temp[these][maxTemp],
         pch=16, col='tomato', cex=1.2, xpd=TRUE)
  text(dat$year[these][maxTemp], dat$temp[these][maxTemp],
       format(dat$temp[these][maxTemp], digits=4),
       col='tomato', cex=.7, pos=4, font=2, xpd=TRUE)
  }
```

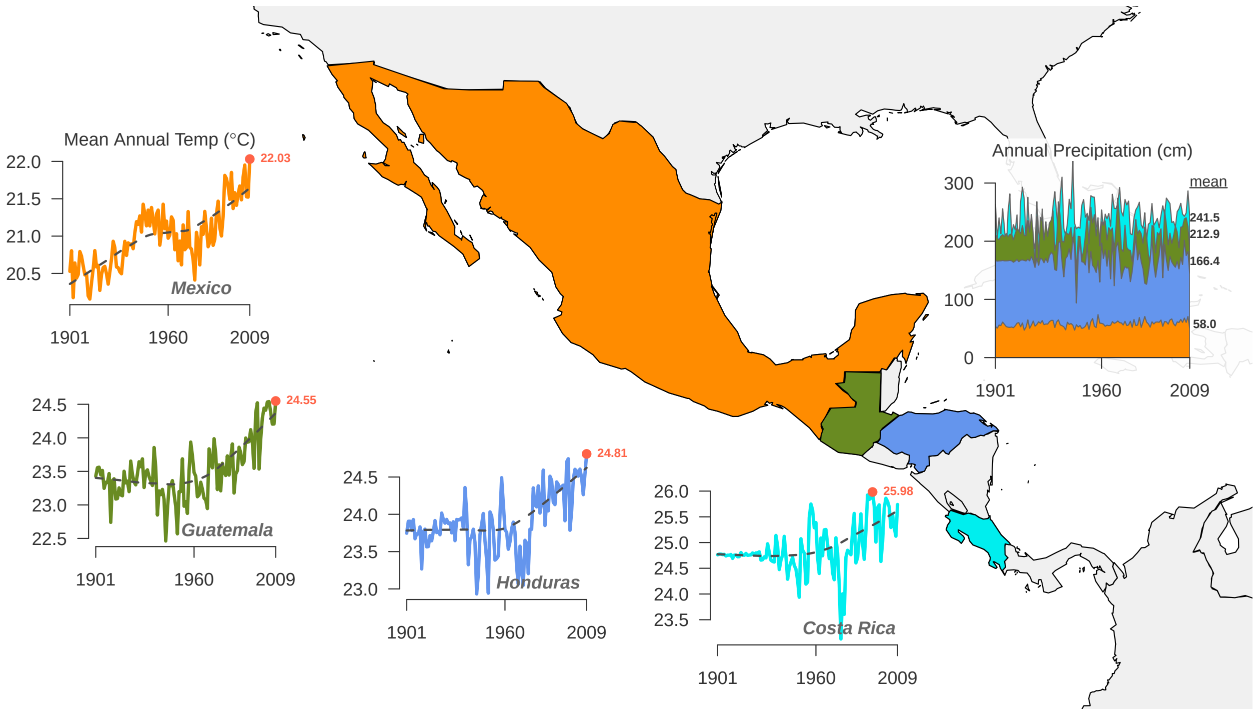
Combine maps, temp, and precip

The maps are surprisingly easy with the `maps` package. This time, we want the map sitting behind everything else, so the `plt` option in `par()` is the best option. The `fig` option would do just as well - `fig` includes room for margins, `plt` does not, so they are good for slightly different applicaitons.

- The `pdf` command is commented out (as is the `dev.off()` command at the bottom), but it stays in place if we want to create a high-quality figure for journal submission. The `eps` or `png` functions would be fine also if that is the desired output. Since we are dealing with maps and fine lines, best to use either `pdf` or `eps` since they are vector-based graphics. Notice that the `pdf` command takes the same width and height measurements as the actual code chunk - They are both measured in inches.
- Draw the world map, but use the `xlim` and `ylim` options to zoom in on our desired location.
- Add each of our 4 countries in color.
- Put the Carribean behind a semi-transparent rectangle so the precip plot can be put there with minimal distraction.
- The use the `plt` option in `par()` to delineate each individual piece of the figure. It takes coordinates from 0-1, and keeps the underlying plots if `new=TRUE`.
- The `makeLines` function is the one we created above. First for Mexico and then for the other 3 countries.
- The last plot added is the precipitation polygon plot.

```
# pdf('bigMap.pdf', width=12, height=7)
map('worldHires', mar=c(.1, .1, .1, .1),
    xlim=c(-135, -71), ylim=c(2, 35),
    col='gray94', bg='white',  fill=TRUE)
for(i in 1:4) {
  map('worldHires', countryNames[i],
```

```
      add=TRUE, fill=TRUE, col=cols[i])
  }
rect(-86, 17, -70, 29,
     border='transparent',
     col=rgb(1, 1, 1, .9))
par(plt=c(0.08, .23, .57, .77), new=TRUE,
    fg='gray20', col.axis='gray20', col.lab='gray20')
makeLines(these=mex, i=1)
mtext(expression(paste('Mean Annual Temp (', degree, 'C)', sep='')),
      font=2, col='gray20')
par(plt=c(.1, .25, .25, .45), new=TRUE)
makeLines(these=gtm, i=2)
par(plt=c(.34, .49, .18, .38), new=TRUE)
makeLines(these=hnd, i=3)
par(plt=c(0.58, .73, .12, .33), new=TRUE)
makeLines(these=cri, i=4)
par(plt=c(0.80, .95, .5, .76), new=TRUE)
plotPrecip()
```




```
# dev.off()
```