```
> options(width=100)
> library(lattice)
> library(latticeExtra) # for useOuterStrips
> ## sp has plot methods for lattice. It may be overkill, but it
> ## also helps with binding data into maps. loaded by maptools.
> library(maptools)
> library(rgdal) #optional, for projections.
> library(arm)
> library(MRP)
> ## used for melt/cast
> library(reshape2)
> library(colorRamps) # another set of color ramps
> load("~/Desktop/censusMRP/naes00.Rdata")
```

1 School Vouchers, Simple Model

We know how to run the simple model of support for school vouchers; this code is not shown. This demonstrates how to get data out of it and prepare it for mapping with external packages. I begin with the fullest-featured mapping package I know, `sp`, which provides S4 classes for "spatial data frames" as well as reading and writing esri shapefiles such as the US Census tiger/Line ones.

This will be important later on – there are tiger/Line shapefiles for zip codes, MSAs, and congressional districts! The built-in 'state' dataset is not easy to merge with other things!

Here is where things get interesting. We begin by simply getting the data out of the `mrp` object. I use Wickham's `reshape` tools to convert this array into a data frame that we'll use to bind onto the map.

The *SpatialPolygonsDataFrame* object contains (among a couple others) a `data` slot separate from the `polygons`.

```
> favorVoucher2000 <- p (favorVoucher2000Mrp.simpleModel)
> ### Get full melted 'p' by income
> favorVoucher2000.byState <- melt(p (favorVoucher2000Mrp.simpleModel, c(TRUE, FALSE, TRUE)))
> favorVoucher2000.byState$value <- favorVoucher2000.byState$value - favorVoucher2000
> favorVoucher2000.byState$income <- as.ordered(relevel(favorVoucher2000.byState$income,
+                                                       "under $25,000",ordered=T))
> favorVoucher2000.byState <- dcast(favorVoucher2000.byState, stateAbbreviation~income)
> ## make relative to national average:
> ##spmap.states <- readShapeSpatial("~/mrp/malecki/fe_2007_us_state")
> ## this one is smaller
```

```
> spmap.states <- readShapeSpatial("~/mrp/malecki/s_01au07",proj4string=CRS("+proj=longlat +datum=NAD27"))
> spmap.states <- spTransform(spmap.states, CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96"))
> ## One in this has no 2letter abbrev.
> spmap.states <- spmap.states[-4,]
> ## Going to use abbrev as rownames, need to change feature ids.
> spmap.states <- spChFIDs(spmap.states, levels(spmap.states@data$STATE)[spmap.states@data$STATE])
> rownames(spmap.states@data) <- levels(spmap.states@data$STATE)[spmap.states@data$STATE]
> rownames(favorVoucher2000.byState) <- favorVoucher2000.byState$stateAbbreviation
> ## Factors are a bit of a pain here, get rid of them in two places.
> favorVoucher2000.byState$stateAbbreviation <- as.character(levels(favorVoucher2000.byState$stateAbbreviation)[
> spmap.states$STATE <- as.character(levels(spmap.states$STATE)[spmap.states$STATE])
> ## remove DC,AK,HI from fitted model, and then pare the sp object to match.
> favorVoucher2000.byState <- subset(favorVoucher2000.byState,
+                                    !(stateAbbreviation %in% c("DC","AK","HI")))
> spmap.states <- spmap.states[spmap.states$STATE %in% favorVoucher2000.byState$stateAbbreviation,]
> ## the main trick of working with sp objects is the order of the data.
> spmap.states@data <- cbind(spmap.states@data,
+                       favorVoucher2000.byState[rownames(spmap.states@data),]
+                       )
> ## the reshaped data has invalid column names.
> prettyNames <- colnames(spmap.states@data)[7:11]
> names(spmap.states@data)[7:11] <- c("one","two","three","four","five")
```

1.1 P only by income

The `spplot` function of `sp` uses Lattice and facets or conditions plots based on z values in columns of the `@data` slot of the `SpatialPolygonsDataFrame` object. For our purposes we will generally want to reshape and "cast" the data using something like

```
state ~ age + income
```

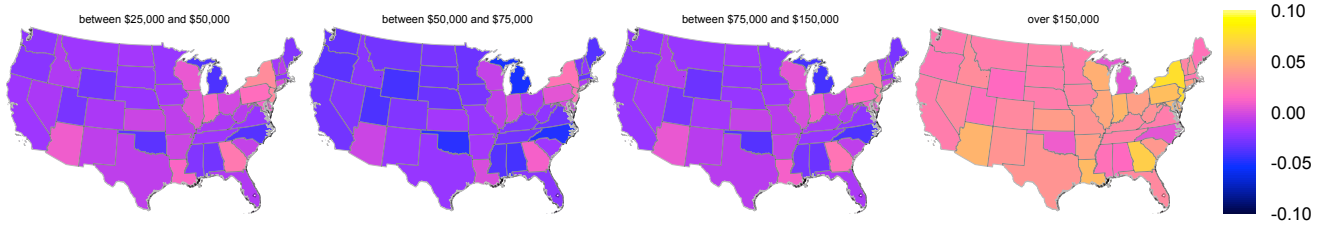
We will need to keep track of the dimensions of both of these, and then layout the plot in the right number of rows and columns. We will also want to make custom labels based on the original `levels` of the conditioning variables.

1.2 Modifying the display, round one

The biggest reason to do things this way is how easy it becomes to customize the plot. Here I change the color scheme by subbing in another list of colors of arbitrary length – by default it is

Figure 1. Simplified Census map from shapefile using packages sp and maptools, plotted using Lattice. Support for vouchers by income.

```
> print(
+       spplot(spmap.states,
+              8:11, # columns of the @data to plot.
+                    # TODO : keep track of 2way conditioning and order
+                    # and set layout automatically.
+              layout=c(4,1),
+              at=seq(-.1,.1,.002),
+
+              names.attr=prettyNames[2:5], # should use levels of original 'p' df
+              between=list(x=.25,y=.25),
+              par.settings=list(
+                add.line=list(col="#00000044",lwd=.2), # state borders
+                add.text=list(cex=.5), # strip text
+                axis.line=list(lwd=0), # boxes around each
+                layout.heights=list(strip=1.2),
+                strip.background=list(col="transparent"),
+                strip.border=list(col="transparent"),
+                regions=list(col=bpy.colors())
+                )
+              )
+       )
```



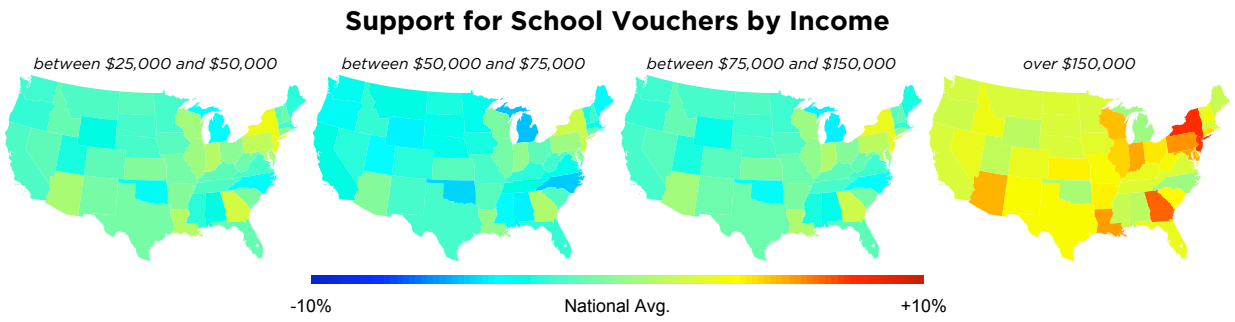
100 which is perceptually continuous – but doing it this way lets the graphics package rather than the user do the mapping of levels into cutpoints into colors.

1.3 Two-way Poststratification 'by hand'

```
> favorVoucher2000 <- p (favorVoucher2000Mrp.simpleModel)
> ### Demo use of ordered factor on levels:
> ### Get full melted 'p' by religionAndEthnicity and income
> favorVoucher2000.byState <- melt(p (favorVoucher2000Mrp.simpleModel, c(TRUE, TRUE, TRUE)))
> favorVoucher2000.byState$value <- favorVoucher2000.byState$value - favorVoucher2000
> favorVoucher2000.byState$income <- as.ordered(relevel(favorVoucher2000.byState$income,
```

Figure 2. Support for vouchers by income, with alternate color scheme, no state border lines, and scale displayed at bottom.

```
> quartzFonts(gotham=quartzFont(c(
+               "Gotham-Book","Gotham-Bold",
+               "Gotham-BookItalic","Gotham-BoldItalic")))
> print(
+       spplot(spmap.states,
+              8:11, # columns of the @data to plot.
+                    # TODO : keep track of 2way conditioning and order
+                    # and set layout automatically.
+              layout=c(4,1),
+              names.attr=prettyNames[2:5], # should use levels of original 'p' df
+              main="Support for School Vouchers by Income",
+              at=seq(-.1,.1,.002),
+              ## details that are hard in base graphics
+              colorkey=list(
+                labels=list(labels=c("-10%","","National Avg.","","+10%")),
+                space="bottom",height=.5,width=.5
+                ),
+              between=list(x=.25,y=.25),
+              par.settings=list(
+                par.main.text=list(fontfamily="gotham"),
+                add.line=list(col="#00000000",lwd=0), # state borders
+                add.text=list(cex=.7,fontface="italic",fontfamily="gotham"),
+                axis.line=list(lwd=0),
+                layout.heights=list(strip=1.2),
+                strip.background=list(col="transparent"),
+                strip.border=list(col="transparent"),
+                regions=list(col=(blue2green2red(100)))
+                )
+              )
+       )
```



```
+                                                        "under $25,000",ordered=T))
> ## hold on to levelnames for plotting
> dimlabels <- sapply(favorVoucher2000.byState, levels)
> plotframe <- dcast(favorVoucher2000.byState, stateAbbreviation~religionAndEthnicity+income)
> ## args: dcast data frame and previously extracted list of labels
> ## returns: levels associated with known dimensions
> pretty.names <- function(plotframe, dimlabels){
+   statedim <- which(names(plotframe)==names(dimlabels))
+   firstdim <- which(sapply(sapply(unlist(sapply(dimlabels, function(x) {x[1]})),
+                                   function(x){
+                                     grep(paste("^",x,sep=""),
+                                          names(plotframe),perl=TRUE) }),
+                            length)!=0)
+   lastdim <- which(sapply(sapply(unlist(sapply(dimlabels, function(x) {x[length(x)]})),
+                                  function(x){
+                                    grep(paste("\\Q",x,"\\E","$",sep=""),
+                                         names(plotframe),perl=TRUE) }),
+                           length)!=0)
+   if(length(dimlabels)==5){
+     middledim <- which(!names(dimlabels) %in%
+                        c(names(dimlabels)[c(statedim,firstdim,seconddim)],"value"))
+     prettynames <- list(page=dimlabels[[firstdim]],row=dimlabels[[middledim]],
+                         col=dimlabels[[lastdim]],state=dimlabels[[statedim]])
+   } else {
+     prettynames <- list(row=dimlabels[[firstdim]],
+                         col=dimlabels[[lastdim]],
+                         state=dimlabels[[statedim]])
+   }
+   return(prettynames)
+ }
> ## get variable names
> plotframe.names <- pretty.names(plotframe,dimlabels)
> ## Have to sanitize names of object for spplot
> names(plotframe) <- make.names(names(plotframe))
> ## make relative to national average:
> ##spmap.states <- readShapeSpatial("~/mrp/malecki/fe_2007_us_state")
> ## this one is smaller
> spmap.states <- readShapeSpatial("~/mrp/malecki/s_01au07",proj4string=CRS("+proj=longlat +datum=NAD27"))
> spmap.states <- spTransform(spmap.states, CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96"))
> ## One in this has no 2letter abbrev.
> spmap.states <- spmap.states[-4,]
> ## Going to use abbrev as rownames, need to change feature ids.
```

```
> spmap.states <- spChFIDs(spmap.states,
+                          levels(spmap.states@data$STATE)[spmap.states@data$STATE])
> rownames(spmap.states@data) <- levels(spmap.states@data$STATE)[spmap.states@data$STATE]
> rownames(plotframe) <- plotframe$stateAbbreviation
> ## Factors are a bit of a pain here, get rid of them in two places.
> plotframe$stateAbbreviation <- as.character(levels(plotframe$stateAbbreviation)[plotframe$stateAbbreviation])
> spmap.states$STATE <- as.character(levels(spmap.states$STATE)[spmap.states$STATE])
> ## remove DC,AK,HI from fitted model, and then pare the sp object to match.
> plotframe <- subset(plotframe,
+                                    !(stateAbbreviation %in% c("DC","AK","HI")))
> spmap.states <- spmap.states[spmap.states$STATE %in% plotframe$stateAbbreviation,]
> ## the main trick of working with sp objects is the order of the data.
> spmap.states@data <- cbind(spmap.states@data,
+                       plotframe[rownames(spmap.states@data),]
+                       )
```

Figure 3. Simplified Census map from shapefile using packages sp and maptools, plotted using Lattice. Support for vouchers by income and race.

```
> plotframe.names$row[4:7] <- c("White\nCatholics","White\nEvangelicals","White\nNon-E. Protestant", "White\nOth
> print(
+       spplot(spmap.states,
+              7:41, # columns of the @data to plot.
+                    # TODO : keep track of 2way conditioning and order
+                    # and set layout automatically.
+              layout=c(
+                length(plotframe.names$col),
+                length(plotframe.names$row)),
+              #names.attr=paste(rep(plotframe.names$row,each=length(plotframe.names$col)),
+              #  "\n",plotframe.names$col,sep=""), # should use levels of original 'p' df
+              main="Support for School Vouchers by Income",
+              strip=strip.custom(
+                factor.levels=rep(plotframe.names$col,
+                  each=length(plotframe.names$row))),
+              strip.left=strip.custom(horizontal=FALSE,
+                factor.levels=rep(plotframe.names$row,
+                 length(plotframe.names$col))),
+              #at=seq(-.1,.1,.002),
+              ## details that are hard in base graphics
+              #colorkey=list(
+              #  labels=list(labels=c("-10%","","National Avg.","","+10%")),
+              #  space="bottom",height=.5,width=.5
+              #  ),
+              between=list(x=.25,y=.25),
+              par.settings=list(
+                par.main.text=list(fontfamily="gotham"),
+                add.line=list(col="#00000000",lwd=0), # state borders
+                add.text=list(cex=.7,fontface="italic",
+                  fontfamily="gotham"),
+                axis.line=list(lwd=0),
+                ## Here we are going to do some
+                ## strip and strip.left magic.
+                layout.heights=list(strip =
+                  c(rep(0, length(plotframe.names$row)-1),
+                    1)), # should by dynamic for linebreaks
+                layout.widths=list(strip.left=
+                  c(2, rep(0,length(plotframe.names$col)-1))),
+                strip.background=list(col="transparent"),
+                strip.border=list(col="transparent"),
+                regions=list(col=(blue2green2red(100)))
+                )
+              )
+       )
```