GridExtraPresentation

Colleen Yancey

March 26, 2018

This package is really great for creating and customizing tables of data. GridExtra will help you to make professional and publishable quality tables that will nicely compliment graphed data. ###Call Libraries needed

library(ggplot2)  
library(gridExtra)  
library(grid)  
library(lattice)  
library(gtable)

### Build Matrix to Work with

For this excerise we will create a simple matrix to work with. You can obviously use this program on data that has been imported to R via a .csv, but I am using a matrix for demonstration purposes.

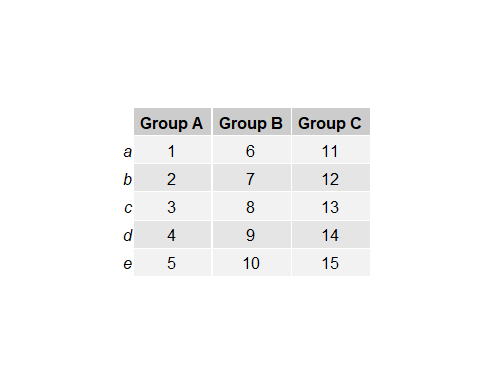
myData <- matrix((1:15),  
 nrow=5,  
 ncol=3,)  
colnames(myData) <- c("Group A", "Group B", "Group C")  
rownames(myData) <- c("a", "b", "c", "d", "e")  
print(myData)

## Group A Group B Group C  
## a 1 6 11  
## b 2 7 12  
## c 3 8 13  
## d 4 9 14  
## e 5 10 15

### Build the Data Table

Now the simplest command to make a table look nice is really easy:

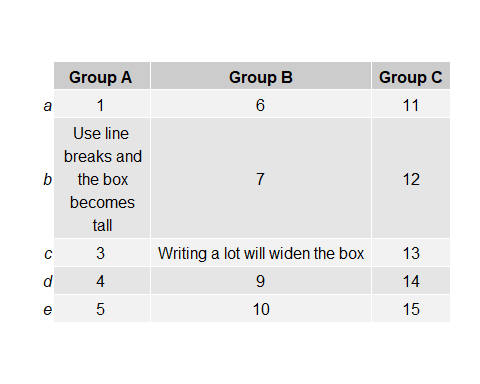
grid.table(myData)



### Table Customizations

Making a basic table is nice, but we can alter them to look much nicer. #### Grid Spacing Grid Spacing will change with the contents inside of each block. This is better than microsoft products because you won’t have to adjust cell size to fit your text; it is automatically done. Standard writing conventions of code will work in tables (ie the use of LaTex).

myData[3,2] <- "Writing a lot will widen the box"  
myData[2,1] <- "Use line\n breaks and\n the box\n becomes\n tall"  
grid.table(myData)

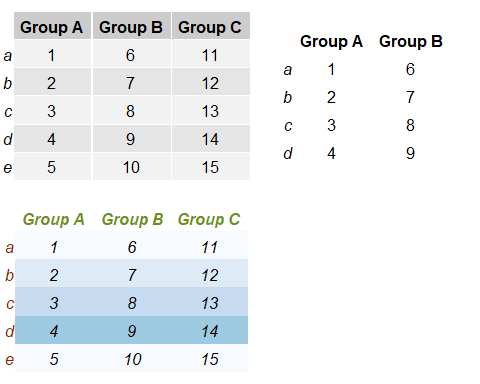
 Revert Matrix Back to original:

myData[3,2] <-8  
myData[2,1] <- 2

#### Table Themes

There are two table themes that are in default, ttheme\_default, and ttheme\_minimal, but you can also customize tables to change colors in any way that you want. To customize, you take ttheme\_minimal and build in colors and parameters of your choice. Blues9 is a color vector that is the default already saved into R. If you want another color, you can create your own color vector using densCols. Fg\_params is used to modify any color or size issue to do with font, bg\_params is used to modify structures and layouts of boxes. (I like to thinkg f in fg\_params =font, and b in bg\_params= box). Font faces are something you can play around with and change as well, these are defaults that I found in demo code that seem to work really well.

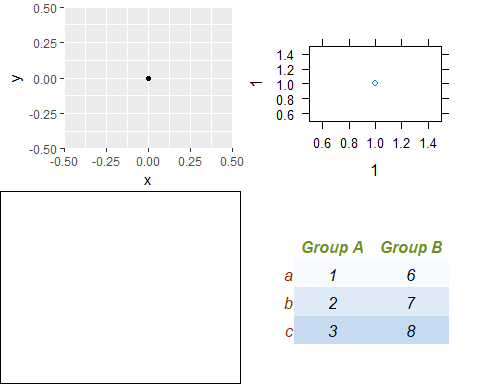
Tdefault <- ttheme\_default()  
Tmin <- ttheme\_minimal()  
TSpecial <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = blues9[1:4], col=NA),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="olivedrab", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orangered4", fontface=3L)))  
  
#using another cool trick for side by side comparison, grid.arrange can display many tables at once:  
grid.arrange(  
 tableGrob(myData[1:5, 1:3], theme=Tdefault),  
 tableGrob(myData[1:4, 1:2], theme=Tmin),  
 tableGrob(myData[1:5, 1:3], theme=TSpecial),  
 nrow=2) #changing nrow will fit your data into desired number of rows



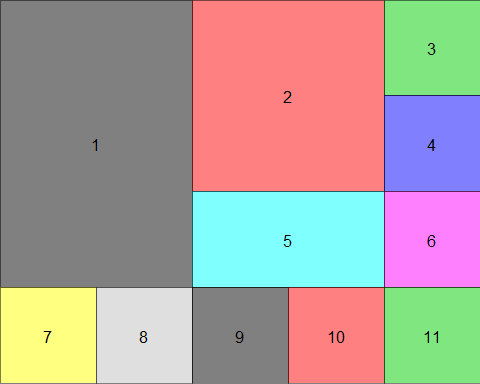
#### grid.arrange Further Explored

Grid arrange and the grob function are really helpful if you want to make a personlized array of data. You can also incorperate graphs into your grid arrangements. I think this is really helpful if you want to export one large figure that has multiple parts.

p <- qplot(0,0)  
p2 <- xyplot(1~1)  
r <- rectGrob(gp=gpar(fill="white"))  
t <- tableGrob(myData[1:3, 1:2], theme=TSpecial)  
grid.arrange(p, p2, r, t, ncol=2)



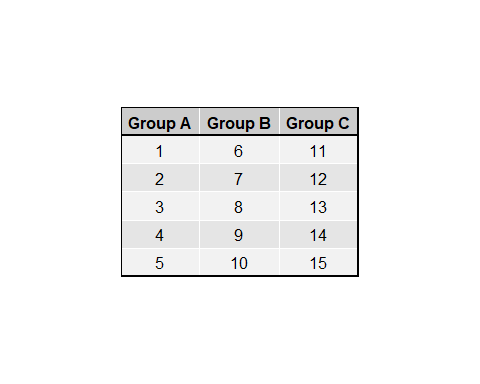
# adding a little more complexity  
g <- lapply(1:11, function(i)  
 grobTree(rectGrob(gp=gpar(fill=i, alpha=.5)), textGrob(i)))  
pattern <- rbind(c(1,1,2,2,3),  
 c(1,1,2,2,4),  
 c(1,1,5,5,6),  
 c(7,8,9,10,11))  
grid.arrange(grobs=g, layout\_matrix= pattern)



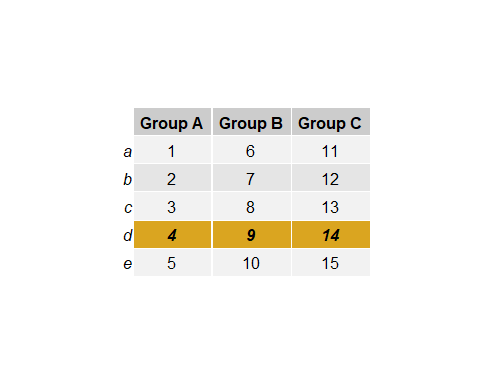
#delete 11 and see what happens, needs to be a complete shape  
#but using NA instead of a number will create an empty space

Once you have your overall layout made, you can call specific items to the boxes. #### Back to Tables: Borders This will make a table that has nice established borders and gets rid of row numbers. You need to understand the following to use the gtable function: t= top extent of the grob r= right extent l=left extent b=bottom gp= graphical parameters calling (fill, thickness of a line etc.)

g <- tableGrob(myData[1:5, 1:3], rows = NULL)  
g <- gtable\_add\_grob(g,  
 grobs = rectGrob(gp = gpar(fill = NA, lwd = 2)),  
 t = 2, b = nrow(g), l = 1, r = ncol(g))  
g <- gtable\_add\_grob(g,  
 grobs = rectGrob(gp = gpar(fill = NA, lwd = 2)),  
 t = 1, l = 1, r = ncol(g))  
grid.draw(g)

 ###Emphasizing a row in the table This is actually pretty annoyin to do and the code is sort of complex. I don’t really recommend this.

t1 <- ttheme\_default(core=list(  
 fg\_params=list(fontface=c(rep("plain", 3), "bold.italic")),  
 bg\_params = list(fill=c(rep(c("grey95", "grey90"),  
 length.out=3), "goldenrod"),  
 alpha = rep(c(1,0.5), each=5))  
 ))  
  
grid.table(myData[1:5, 1:3], theme = t1)

 ### Final Notes This might seem trouble than what it is worth to create a table. I think it is great for making basic simple tables, but if you want to add in emphasis and special effects to specfic boxes you might want to consider another way. However, once I started really working with the code it started to make much more sense about how to set parameter, etc. In the future, I will use this to make quick tables to display with my figures so that I can easily move and manipulate them when writing results and publications.