Data Visualization using R

Damian W. Betebenner
National Center for the Improvement of Educational Assessment (NCIEA)
Dover, New Hampshire
DBetebenner@nciea.org

October 27, 2016

Getting Started

This primer provided a basic introduction to the data visualization capabilities of R (R Development Core Team, 2016). This primer is by no means meant to be comprehensive, but will hopefully provide an overview of the power of R's data visualization capabilities and, more generally, introduce people to the benefits derived from programatic drawing—that is, programming one's data visualizations instead of using standard *what you see is what you get* (WYSIWYG) interfaces.

You don't need a lot to begin creating visualizations using R. There are three basic ingredients:

1. The Rsoftware environment that can be downloaded from the Comprehensive RArchive Netword (`http://cran.r-project.org`)
2. Some data visualization ideas
3. Enthusiasm

And best of all, each ingredient doesn't have to cost anything! To get you started with all three pieces, after you download and install Rfrom CRAN, type the following at the prompt:

```
demo(graphics)
```

Note: If you ever need help in Rwith a command (particularly to know what arguments it takes), you can always type `?command` at the prompt and it will return the help page for the command. Try

```
?demo
?plot
```

Diving In

The best way to start building visualizations with Ris just to get your hands dirty and work from constructing simple visualizations to building much more complex ones. Because R is syntax based, it Because Ris syntax based, one needs a syntax (i.e. text) editor to write code in. Windows, for example, comes with `notepad` and `wordpad` which work well. Word doesn't work well because it doesn't save a text file and the editor itself has hidden characters that cause Rto choke. Ideally, what you want to get is an editor (and there are numerous free ones) that color codes the text and shows whether parentheses/brackets are balanced/closed. My favorite is `vim/vi`, however it has its own learning curve.

Step 1: Data in R

Rhas numerous data types to work with. Some of these include vectors, arrays, matrices, data.frames, and lists. Getting used to the object oriented way in which Rworks with data is one of the initial challenges a beginning encounters. However, after getting used to it one will find (at least I did) that the data types are very intuitive and powerful.

```
#### Data in R

# vectors

vector1 <- c(1, 3, 4)
vector2 <- 1:10
vector3 <- rnorm(50)

# Arrays, matrices, and data.frames

array1 <- array(rnorm(10), dim=c(5,2))
matrix1 <- matrix(rnorm(100), nrow=20)
data.frame1 <- data.frame(ID=1:10, Score=runif(10))
```

Note that the `<-` operator assigns what's on the right of the operator to what is on the left. One can also use `=`. After copying and pasting the above code into R, type `vector1`, `vector2`, and `data.frame1`.

Ralso comes with a number of built in data sets. To view all of the built in data sets, just type `data()`. For example, the built in data set `trees` has girth, height and volume for black cherry trees and `UCBAdmissions` has data regarding student admissions at UC Berkeley. Enter the following

```
trees
head(trees)
tail(trees)
head(trees, 10)
class(trees)
dim(trees)
names(trees)
UCBAdmissions
tail(UCBAdmissions, 15)
class(UCBAdmissions)
dim(UCBAdmissions)
```

Step 2: Graphics first steps

Rhas numerous built in functions in the base graphics for producing standard visualizations. These visualizations can be customized in almost limitless ways. And when these functions don't do the trick, there is more advanced functionality in other packages like grid and the lattice and ggplot packages that are built on grid. The Rplatform provides (in my humble opinion) the best platform for constructing static data visualizations available. Here's a list of the charts available in the base graphics: `barplot`, `dotchart`, `hist`, `density`, `stripchart`, `qqnorm`, `plot`, `smoothScatter`, `qqplot`, `pairs`, `image`, `contour`, `persp`, `interaction.plot`, `sunflowerplot`. The help associated with each of these chart types provide examples toward the bottom of the help that shows both what the chart type looks like but how some of the arguments function. The examples in the help are invaluable.

Let's try some basic plots. Note what the arguments within the function do to the resulting chart and how labels are added.

```
# To produce a scatterplot

plot(rnorm(50), rnorm(50))

plot(rnorm(50), rnorm(50), main="My Title", font.main=4, xlab="X Label",
   ylab="Y Label")

plot(rnorm(50), rnorm(50), main="My Title", font.main=3, xlab="X Label",
   ylab="Y Label")

#To produce a barplot:

barplot(c(2, 4, 5, -1))


dotchart(runif(26), letters)

dotchart(seq(0, 100, length=26), LETTERS)

hist(rnorm(1000))

hist(rnorm(1000), probability=TRUE)

plot(density(rnorm(1000)))
```

The `plot` function has an argument `type` that produces different types of plots including plots with just points, lines, or both. The argument `type=''n''` produces no plot. This can be helpful in constructing more customized plots.

```
plot(rnorm(50), type="l")
plot(rnorm(50), type="b")
plot(rnorm(50), type="n")
plot(rnorm(50), type="o")
```

Step 3: Outputting your figures

For many, R's interactive figure production will suffice for copy and paste into a Word document. There are numerous output formats available via drivers created for R. Some of these include `pdf`, postscript (`ps`), windows metafile format (`win.metafile`), `png`, `jpeg`, bitmap (`bmp`), `tiff`. Some of the formats are vector (resolution independent) and some are bitmap. I highly recommend using a vector format if possible because the quality is far superior (pdf, win.metafile are vector formats).

It should be noted that different drivers lead to subtilely different looks to figures beyond just vector/bitmap issues. Issues associated with embedded fonts and other factors can subtly change the look of figures. Choose the driver that best suits your needs. My personal preference is `pdf` because the vector quality is excellent and I embed them in LaTeXdocuments. If you use Word, pdf doesn't embed (easily), so you might use Windows formats instead.

For more information on available drivers see `?device`

```
# To produce a pdf figure
```

```
pdf(file="my_first_pdf.pdf")
plot(rnorm(50), rnorm(50), main="My First PDF", font.main=4, xlab="X Label",
    ylab="Y Label")
dev.off()

# To produce a windows metafile

win.metafile(file="my_first_wmf.wmf")
plot(rnorm(50), rnorm(50), main="My First PDF", font.main=4, xlab="X Label",
    ylab="Y Label")
dev.off()
```

Each output device takes additional arguments specifying, for example, height and width. The following example modifies some of the arguments for the `pdf` device, changing the default width, height, and background (`bg`) color.

```
# Let's play with some pdf features

pdf(file="my_second_pdf.pdf", width=11, height=8, bg="grey80")
plot(1:20, 1:20, xlim=c(0,25), ylim=c(0,30),
     main="My Second PDF", font.main=4, cex.main=2,
     xlab="X Label", ylab="Y Label", cex=seq(1,5,length=20), bg=rainbow(20),
         pch=21, lwd=seq(1,7,length=20))
dev.off()
```

Step 4: Color

Despite the anachronistic black and white printed page imposed on researchers by academic journals, we are not limited to producing figures in black and white. Color doesn't just add an aethestically pleasing touch to the work you produce (the impact of which should NOT be dismissed), but also can be used to tell more complicated stories with data—using the eye's ability to distinguish color as a critical feature of the data representation.

Rhas a number of features allowing you to add color to your pictures. One can specify colors for various features in various ways. As we saw, for example, the background color of the plot can be changed. Similarly one can specify the color of points that are plotted. Defining the colors of these features can be done in numerous way including specifying colors by name. To see a list of all named colors type `colors()`. One can also specify colors using the red-green-blue color space (`rgb`), hue-saturation-value (`hsv`), hue-chroma-luminance (`hcl`). Also, there's are a number of functions and packages that help the user select colors appropriate for the visualization and data they are construct (see, for example, RColorBrewer).

An incredibly powerful part of color with R(and seen in dozens of websites) is the implementation of alpha transparency. That is, the colors that are defined can be given different levels of transparency. This can be very valuable, particularly when there are issues with overlapping data points and one wants to give a sense of depth. It should be noted that not all devices support alpha transparency.

```
# Colors and alpha transparency

x <- rnorm(5000, mean=50, sd=10)
y <- x + rnorm(5000, mean=0, sd=5)

pdf(file="my_third_pdf.pdf", width=11, height=8)
plot(x, y, xlim=c(20, 80), ylim=c(20, 80),
     main="Bivariate Normal Data", xlab="X Label", ylab="Y Label", pch=21,
        col=rgb(0,0,1,0.1), bg=rgb(0,0,1,0.1), cex=1.5)
dev.off()
```

Step 5: Deconstructing a Chart

Thus far we have basically used R's charts without much modification. The best part of Ris that it allows limitless customizations. In the base graphics, this means building a chart up in more pieces. As you add individual pieces (e.g., the axes) to a chart, you can have finer grained control over the result. Many of the features that allow one to override base graphical parameter settings are done by specifying `par`. Type `par()` to see all of the different graphical parameters that can be altered.

The following example alters some of the graphical parameters:

```
# Deconstructing this chart

pdf(file="my_fourth_pdf.pdf", width=11, height=8, bg=rgb(.48, .48, .52))
par(omi=c(0.5,0.5,0.75,0.5))
par(font.main=4)
par(cex.main=1.4)
par(col.main="white")
par(cex.lab=1.25)
par(font.axis=2)
par(col.axis="white")
par(font.lab=2)
par(col.lab="white")


plot(50, 50, type="n", xlab="", ylab="", xlim=c(10,90), ylim=c(10,90),
   axes=FALSE)
polygon(c(10, 10, 90, 90), c(10, 90, 90, 10), col="white")
points(x, y, pch=21, col=rgb(1,0,0,0.1), bg=rgb(1,0,0,0.1), cex=1.5)

axis(1, seq(20,80, by=10), lwd=1.5)
axis(2, seq(10,90, by=20), lwd=2.5)
axis(3, c(40,50,60) , labels=c("Forty","Fifty","Sixty"), lwd=1.5)
axis(4, seq(15,75, by=10), lwd=3.5)

mtext("Side 1 Label", side=1, line=4, col="blue")
mtext("Side 2 Label", side=2, line=4, col="blue")
mtext("Side 3 Label", side=3, line=3, col="blue")
mtext("Main title", side=3, line=6, cex=2, font=3)
mtext("Side 4 Label", side=4, line=2.5, col="blue")
dev.off()
```

Step 6: Trellis Plots

Rhas tremendous facility to put together plots that fill multiple panels on a page. There are a number of functions that implement this automatically (e.g, `pairs`). To produce such plots yourself, the basic `par` settings that need to be specified are either `mfrow` or `mfcol`.

```
library(quantreg) # load quantile regression package
```

```
pdf(file="my_fifth_pdf.pdf", width=11, height=8)
par(mfrow=c(2,2))
plot(trees$Girth, trees$Height)
abline(v=mean(trees$Girth), lty=3, col='grey80')
abline(h=mean(trees$Height), lty=3, col='grey80')
plot(trees$Girth, trees$Volume)
abline(lm(trees$Volume ~ trees$Girth))
abline(rq(trees$Volume ~ trees$Girth), col=rgb(0,0,1), lty=2)
plot(trees$Height, trees$Volume)
dev.off()
```

Step 7: Bulk Plot Production

The easy to use looping facilities in Rmake it easy to create programs that mass produce customized output. I've used these facilities to produce student reports for the state of Colorado (460,000 student reports) as well as much smaller runs producing a single report for each district in a state. The key to doing such an operation is to have a data set containing all the data to be shown across all the reports, create a loop that runs through each of the groups for which you want to make a printout, subset the appropriate data for that group, create the figure for that data, and repeat.

The following is a toy example but contains all the elements necessary to do this at scale

```
my_data <- data.frame(GROUP=rep(1:10, each=10), X_COOR=rnorm(100),
    Y_COOR=rnorm(100))

for (i in unique(my_data$GROUP)) {
tmp.data <- subset(my_data, GROUP==i)

attach(tmp.data)

pdf(file=paste('batch_plot_', i, '.pdf', sep=''), width=11, height=8.5)
plot(X_COOR, Y_COOR, main=paste('Title for Batch Plot', i), xlab='X Label',
    ylab='Y Label')
dev.off()

detach(tmp.data)

}
```

Next Steps

This introduction barely scratches the surface of what is possible with Rgraphics. The availability of high quality examples with source code is invaluable in learning and producing high quality figures for yourself. High quality data visualizations are in great demand and developing the capacity to produce such representations is a valuable skill to have.

References

R Development Core Team. (2016). R*: A language and environment for statistical computing.* Vienna, Austria. (3-900051-07-0)