Lesson 6: Stronger visualizations with ggplot2 and lattice

Randall Julian

12/28/2017

# Plotting Data With Default Graphics

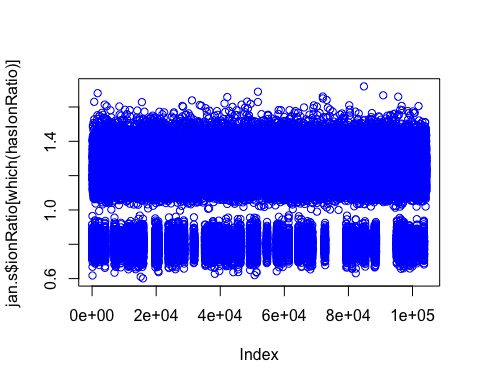
Default **R** comes with several basic plotting commands – *plot* to draw an X,Y graph, *points* to add X,Y points to the current graph, *barplot* to draw vertical or horizontal bars, *boxplot* to draw box-and-whisker plots, *hist* to build and draw a histogram, and many other plot types or plot-specific additions to plots.

The first major drawback to using these plots is that each requires learning a slightly different syntax to decorate the graph.

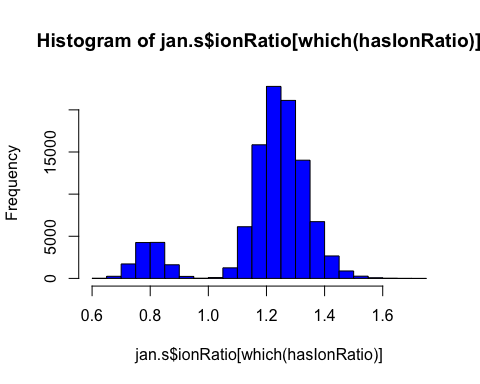
workingDir <- file.path(rootDir,"class\_data")  
jan.s <- read\_csv(file.path(workingDir,"2017-01-06.csv"))

## Parsed with column specification:  
## cols(  
## batchName = col\_character(),  
## sampleName = col\_character(),  
## compoundName = col\_character(),  
## ionRatio = col\_double(),  
## response = col\_double(),  
## concentration = col\_double(),  
## sampleType = col\_character(),  
## expectedConcentration = col\_integer(),  
## usedForCurve = col\_logical(),  
## samplePassed = col\_logical()  
## )

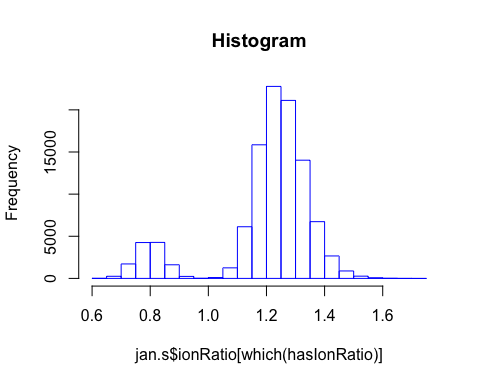
jan.s$idx <- c(1:nrow(jan.s))  
hasIonRatio <- jan.s$ionRatio > 0  
plot(jan.s$ionRatio[which(hasIonRatio)],col='blue')



hist(jan.s$ionRatio[which(hasIonRatio)],col='blue')

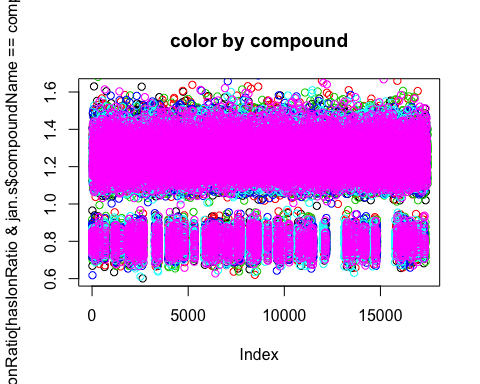


hist(jan.s$ionRatio[which(hasIonRatio)],border='blue',main='Histogram')



The second drawback is that these plots, while drawn quickly, require detailed sort and select mechanisms in order to display complex data on a single graph. Plotting a matrix of graphs (as shown below) is even more difficult and you may spend more time troubleshooting the graph than actually analyzing the data.

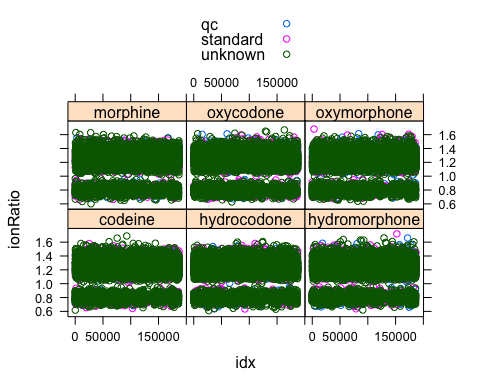
compounds <- unique(jan.s$compoundName)  
for(i in 1:length(compounds)) {  
 if(i==1) {  
 plot(jan.s$ionRatio[hasIonRatio & jan.s$compoundName==compounds[i]],  
 col=i,  
 main="color by compound")  
 } else {  
 points(jan.s$ionRatio[hasIonRatio & jan.s$compoundName==compounds[i]],  
 col=i)  
 }  
}



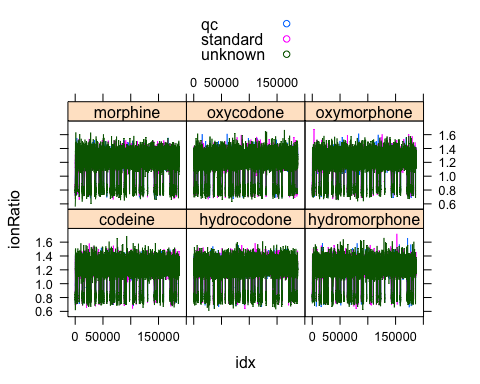
# Plotting Data with *lattice*

Working with large datasets, especially data you want to slice by one or more variables, may require moving to another graphing package. Using **lattice** provides a simplified faceting functionality, with syntax more typical of the default graphics package.

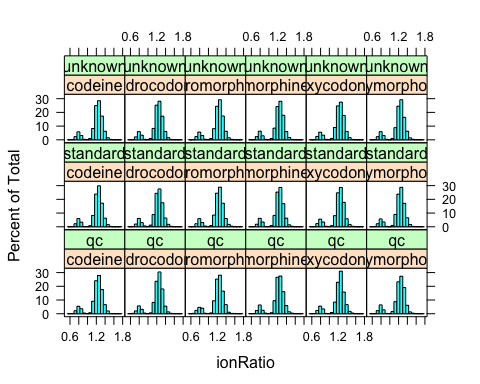
xyplot(ionRatio ~ idx | compoundName,   
 data=jan.s[hasIonRatio,],   
 groups=sampleType,   
 auto.key=TRUE)



xyplot(ionRatio ~ idx | compoundName,   
 data=jan.s[hasIonRatio,],   
 groups=sampleType,   
 auto.key=TRUE,   
 type=c("l","spline"))



histogram( ~ ionRatio | compoundName + sampleType,   
 data=jan.s[hasIonRatio,])



## Benchmarks for running time within **R**

Sometimes there is a need to optimize code for timing as well as readability, and that usually occurs when plotting is involved. Assuming you can’t reduce the complexity with a filter (e.g. plotting only one compound, or across a smaller time scale) the choice to display 100k-10m datapoints will simply take the time it takes. Knowing how long to expect a process to complete is good for future users of your code, and requires benchmarking. Here we run into an awkward difficulty within R and Rstudio, where ‘execute the command’ and ‘render the figure’ are two different tasks, so we need to wrap a ‘modular unit’ of commands in a function so that **R** treats it like a single command. Then, we call *system.time* to report the userTime (the **R** session) and systemTime (the OS kernel).

system.time(print("Anything seen as a single function can be tested in this way"))

## [1] "Anything seen as a single function can be tested in this way"

## user system elapsed   
## 0 0 0

anyUserFunction <- function(someText) {  
 hasNumber <- grepl("[:digit:]",someText)  
 if(hasNumber) {  
 editedText <- "including their pros and cons"  
 unusedVariable <- rnorm(1e6)  
 } else {  
 editedText <- "have you looked ahead yet"  
 unusedVariable <- rnorm(5e6)  
 }  
 return(editedText)  
}  
system.time(anyUserFunction("we will talk about user functions in lesson 9"))

## user system elapsed   
## 0.088 0.004 0.092

system.time(print(anyUserFunction("we will talk about user functions in lesson 9")))

## [1] "including their pros and cons"

## user system elapsed   
## 0.087 0.001 0.090

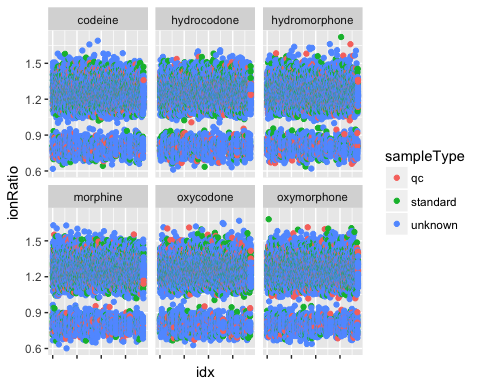
There are purpose-built packages that can make this measurement easier (the libraries **tictoc** and **microbenchmark** are popular) but the above methodology is usually sufficient.

# Plotting Data With *ggplot2*

To maintain the tidy focus of the tidyverse, the **ggplot2** package keeps the same syntax for all graphing schemes, has arguably prettier default graphs, and a frankly intuitive means for layering/faceting of the underlying data. The main drawback is that plotting from a large data.frame is still measured in minutes. The mock data in this course definitely qualifies as a large dataset, so we recommend that plotting be used judiciously if you’re not applying a filter (see below).

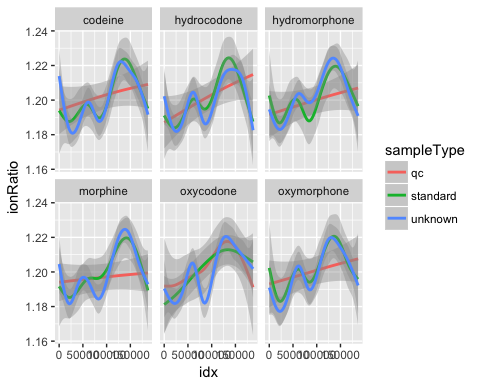
Syntax follows the format of {‘define the data’ {+ ‘describe the visualization’}} where each description is called a *geom* and multiple geoms can be stacked together. Definitions for the aesthetic mappings (e.g. plotTerms, color, iconShape, lineType) can be supplied when defining the data and are applied to the subsequent stack of geoms. Any mappings can be overridden within an individual geom.

g <- jan.s %>%  
 filter(ionRatio > 0) %>%  
 ggplot(aes(x=idx,y=ionRatio,colour=sampleType))  
g + geom\_point() + facet\_wrap(~compoundName) + scale\_x\_continuous(labels=NULL)



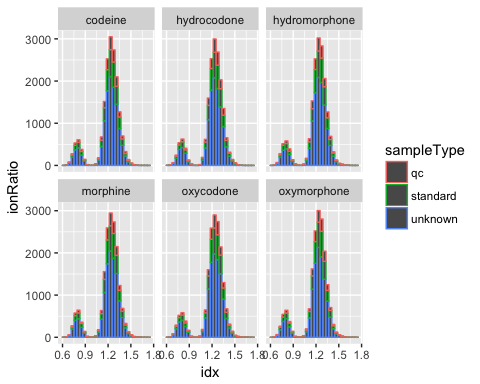
g + geom\_smooth() + facet\_wrap(~compoundName)

## `geom\_smooth()` using method = 'gam'



g + geom\_histogram(mapping=aes(x=ionRatio,colour=sampleType),inherit.aes=FALSE) + facet\_wrap(~compoundName)

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



We could easily spend the whole class session on this package, but the above plots showcase the basic syntax. The cheatsheet downloadable from the link at the end of this lesson provides additional examples of what can be done.

Exercise 1: Draw a better histogram  
The default histogram paramaters for ggplot will stack the sample types in the same bin, making it difficult to determine if the trend for qc and standard samples is the same as the unknowns. The first plot in this exercise makes adjacent bars, but what does the second plot do?

g <- jan.s %>%  
 filter(ionRatio > 0) %>%  
 ggplot(aes(x=ionRatio,colour=sampleType,fill=sampleType))  
#g + geom\_histogram(position='dodge', bins= ) + facet\_wrap(~compoundName)  
#g + geom\_histogram(aes(y=..density..), bins= ) + facet\_grid(sampleType~compoundName)

Exercise 2: Plot timing  
There is a longstanding community opinion that ggplot “takes longer” than the other two plotting mechanisms, but how much longer is it really? Is the time savings from **lattice** worth learning the new syntax?

oneYearSamples <- list.files(workingDir,pattern="csv$") %>%  
 file.path(workingDir,.) %>%  
 map\_dfr(read\_csv)  
oneYearSamples$idx <- 1:nrow(oneYearSamples)  
coreR <- function(oneYearSamples) {  
 sampleTypes <- unique(oneYearSamples$sampleType)  
 for(i in 4:1) {  
 oneType <- which(oneYearSamples$sampleType==sampleTypes[i])  
 if(i==4) {  
 plot(oneYearSamples$idx[oneType],oneYearSamples$concentration[oneType],col=i)  
 } else {  
 points(oneYearSamples$idx[oneType],oneYearSamples$concentration[oneType],col=i)  
 }  
 }  
}  
g <- ggplot(oneYearSamples,aes(x=idx,y=concentration,color=sampleType)) + geom\_point()  
l <- xyplot(concentration ~ idx ,   
 data=oneYearSamples,   
 groups=sampleType,   
 auto.key=TRUE)  
#system.time(coreR(oneYearSamples))  
#dev.off()  
#system.time(print(g))  
#dev.off()  
#system.time(print(l))

# Summary

* [ggplot2 cheatsheat](https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf)
* [lattice overview](https://www.statmethods.net/advgraphs/trellis.html)
* [download a PDF comparison of both packages](http://www.londonr.org/download/?id=69)
* [download an older PDF showing the system.time comparison](https://learnr.files.wordpress.com/2009/08/latbook_time1.pdf)