Lesson 6: Stronger visualizations with ggplot2 and lattice

Adam Zabell

12/28/2017

# Plotting Data With Default Graphics

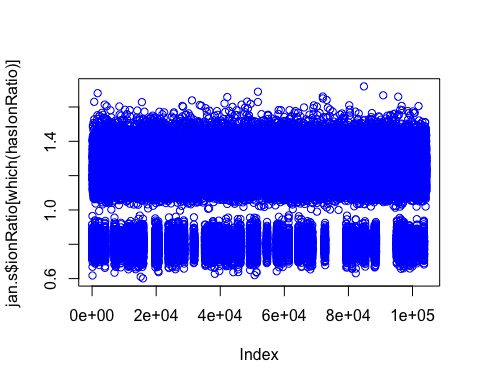
Default **R** comes with several basic plotting commands – *plot* to draw an X,Y graph, *points* to add X,Y points to the current graph, *barplot* to draw vertical or horizontal bars, *boxplot* to draw box-and-whisker plots, *hist* to build and draw a histogram, and many other plot types or plot-specific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

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

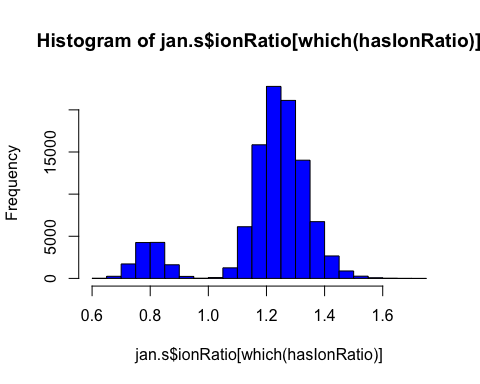
workingDir <- file.path(rootDir,"class\_data")  
jan.s <- read\_csv(file.path(workingDir,"2017-01-06.csv"))

## 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()  
## )

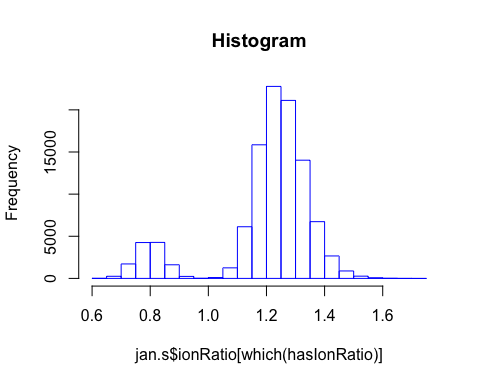
hasIonRatio <- jan.s$ionRatio > 0  
plot(jan.s$ionRatio[which(hasIonRatio)],col='blue')



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

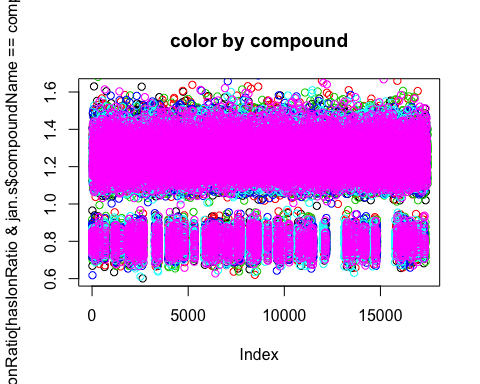


hist(jan.s$ionRatio[which(hasIonRatio)],border='blue',main='Histogram')



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)  
 }  
}

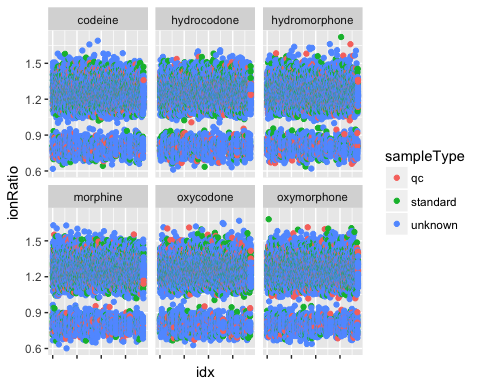


# Plotting Data With *ggplot2*

To maintain the ‘one pattern for one job’ 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 (millions of rows) can be measured in minutes. The mock data in this course definitely qualifies as a large dataset, so we recommend that ggplot2 be used judiciously if you’re not applying a filter (see below).

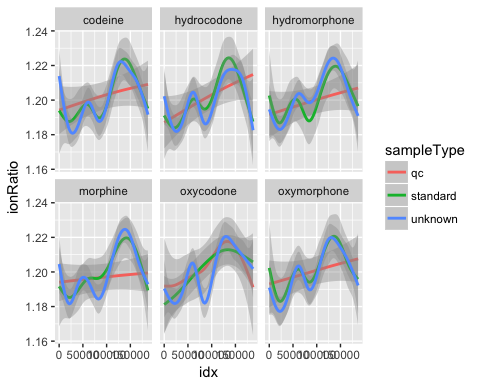
Syntax follows the format of {‘define the data’ {+ ‘describe the visualization’}} where each description is called a *geom* and multiple geoms can be stacked together. Definitions for the aesthetic mappings (e.g. plotTerms, color, iconShape, lineType) can be supplied when defining the data and are applied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

jan.s$idx <- c(1:nrow(jan.s))  
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)



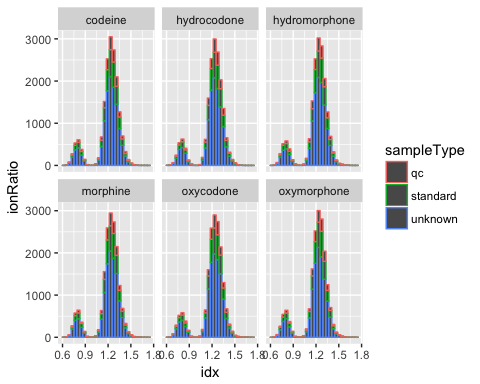
g + geom\_smooth() + facet\_wrap(~compoundName)

## `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.

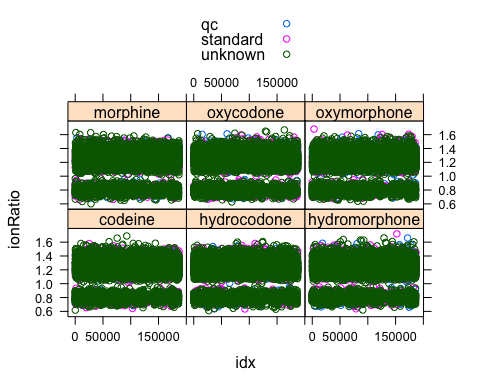
Exercise 1: Draw a better histogram  
The default paramaters stack each of the three sample types for each histogram bin, making it difficult to determine if the trend for qc and standard samples is the same as the unknowns. The first plot in this exercise makes adjacent bars, but what does the second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

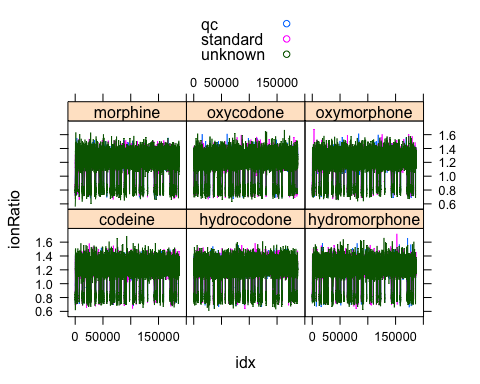
# Plotting Data with *lattice*

When working with large datasets, maintaining the tidyverse ideal of ‘doing useful work quickly’ may require moving to another graphing package. Using **lattice** maintains the simplified faceting functionality, but involves syntax more typical of the default graphics package.

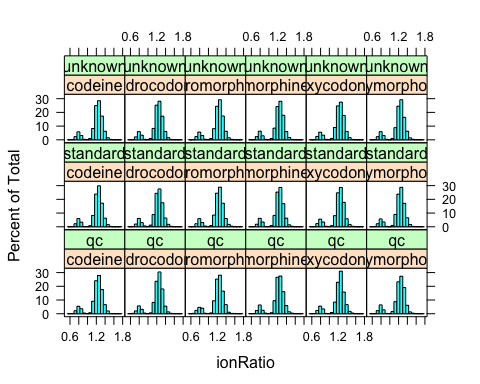
xyplot(ionRatio ~ idx | compoundName,   
 data=jan.s[hasIonRatio,],   
 groups=sampleType,   
 auto.key=TRUE)



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,])



Exercise 2: Plot timing  
We talked about “taking longer” for each of these three plotting mechanisms, but how much longer is it really? 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 each graphing command in a function and call *system.time* to report the userTime (the **R** session) and systemTime (the OS kernel). 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)
* plotting with the lattice package is recommended when working with more than 500k rows