"ggplot2": Understanding the Grammar of Graphics

Aimee Schwab

Wednesday, February 20, 2013

1 Introduction

From the ggplot2 homepage,

"ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics."

ggplot2 was developed to make graphing in R more accessible and intuitive to the novice user. It also provides some powerful options to the experienced R user to create publication-worthy graphics.

2 "qplot()"

In the base R graphics packages, each type of plot has a different function name (i.e. hist, qqplot, etc.). However in ggplot2, all plot types can be called using the same basic command: "qplot()". For new users, this makes learning the basic structure of plotting much easier!

2.1 Basic Structure

The basic structure of qplot() is very similar to existing plotting options in R. A qplot() command (with the most common options) looks like this:

```
qplot(x, y, data=, color=, shape=, size=, alpha=, geom=, method=,
      formula=, facets=, xlim=, ylim=, xlab=, ylab=, main=, sub=)
```

Many of the plotting options look familiar (xlim, xlab, etc.). Others however are new.

- alpha: Sets the transparency for overlapping points. 0 represents complete transparency, 1 represents complete opacity (default is 1).
- color, shape, size, and fill: Associates variable levels with the symbol color, shape, size, or fill. For line plots, levels of a variable are matched with line color. For density or box plots, fills are associated with the variable. Each option can be set to a certain variable, and qplot will do the rest. Basic legends are drawn automatically.
- facets: Creates a trellis graph by specifying conditioning variables. Can specify just one variable or a pair (rowvar ~ colvar).
- geom: Specifies geometric options. Multiple options can be used (although not all sets are valid). Possible values include "point", "smooth", "boxplot", "line", "histogram", "density", "bar", "jitter". We'll see examples of all types soon. The default value depends on the variable inputs!
- main: Specifies main title (just like the base R packages!).
- sub: Specifies sub title (I haven't seen this in the base R packages).
- method: If geom="smooth", method defines which smoothing algorithm is used. Possible methods are "lm" (regression), "gam" (additive models), and "rlm" (robust regression). A Loess smoother is fit by default for samples $< 1{,}000$.
- formula: Specifies the parameters for the smoothed line. If no fitted line is included, then method and formula are not called.

Right away you should notice a big advantage: option names are much more intuitive in ggplot2!

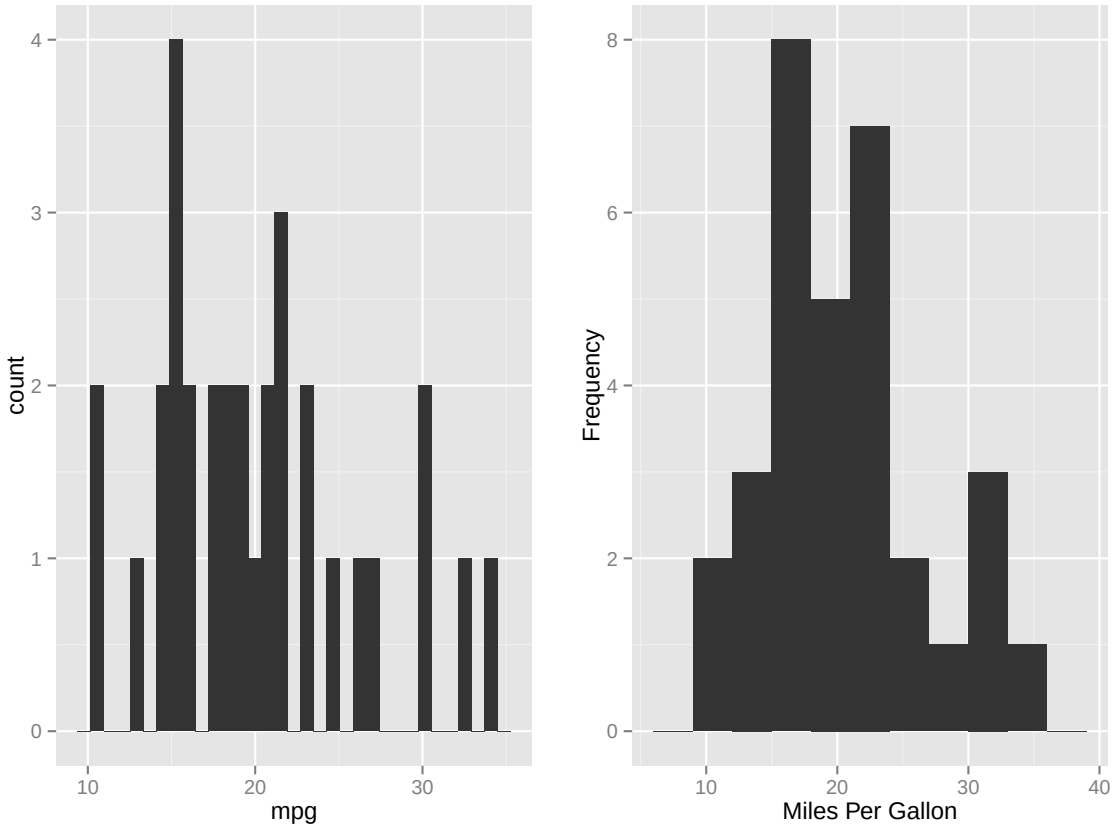
Note: In these notes, I'll discuss the plots separately, with code provided at the end of each example. For the sake of time, I've chosen to focus on exploratory plots, rather than plotting statistical models. There are even more options in ggplot2 for modeling!

2.2 Starter Examples

For some first examples, let's use a data set provided in ggplot2(), "mtcars". mtcars is a data set from a 1974 issue of Motor Trend Magazine which describes fuel consumption, number of cylinders, weight, horsepower, and other automotive benchmarks for 32 different models. Unfortunately the data is a bit old, but this is a good starter data set because of all the possible variable type combinations! :)

Our primary variable of interest will be miles per gallon.

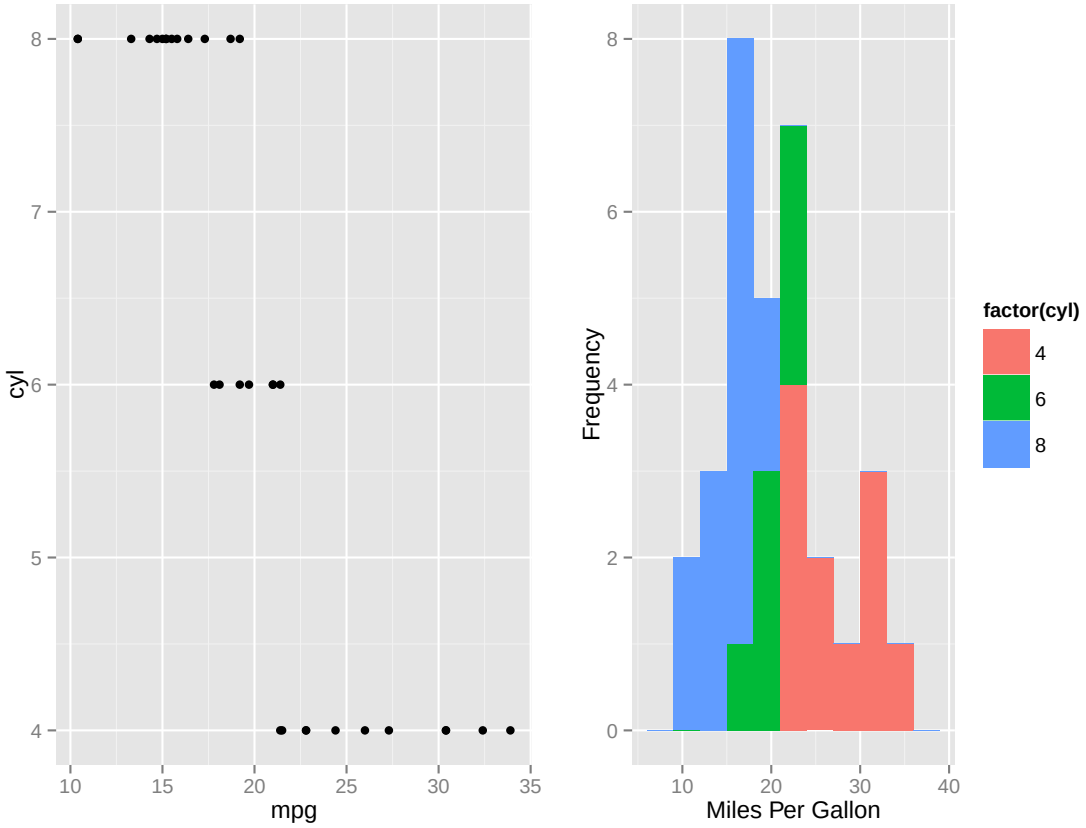
Example 1: Plotting the miles per gallon (mpg). By default, since we are plotting a single quantitative variable, qplot() used a histogram. We can see quite a bit of variation, but the bin size of the histogram leaves something to be desired.




```
> library(ggplot2)
> head(mtcars)
                   mpg   cyl disp  hp drat wt    qsec   vs am gear carb
Mazda RX4         21.0    6  160 110 3.90 2.620 16.46   0  1    4    4
Mazda RX4 Wag     21.0    6  160 110 3.90 2.875 17.02   0  1    4    4
Datsun 710        22.8    4  108  93 3.85 2.320 18.61   1  1    4    1
Hornet 4 Drive    21.4    6  258 110 3.08 3.215 19.44   1  0    3    1
Hornet Sportabout 18.7    8  360 175 3.15 3.440 17.02   0  0    3    2
Valiant           18.1    6  225 105 2.76 3.460 20.22   1  0    3    1
> qplot(mpg, data=mtcars)
> qplot(mpg, data=mtcars, binwidth=3, xlab="Miles Per Gallon",
        ylab="Frequency")
```

Example 2: Plotting miles per gallon against cylinders. What if we want to visualize the relationship between miles per gallon and number of cylinders? Number of cylinders only has three levels (4, 6, and 8), so it would be best to treat this as a category. Unfortunately, qplot() disagrees. By default, two numerical variables will be plotted in a scatterplot. However we can use the "color" option to our advantage in the histogram above. By using the fill option, I can change the color of the histogram bars depending on the number of cylinders. From this plot, we can see that cars with fewer cylinders tended to get better gas mileage. The legend is plotted automatically, using whatever variable name the categories are in. Another option could be to use the facets option to create three separate histograms.

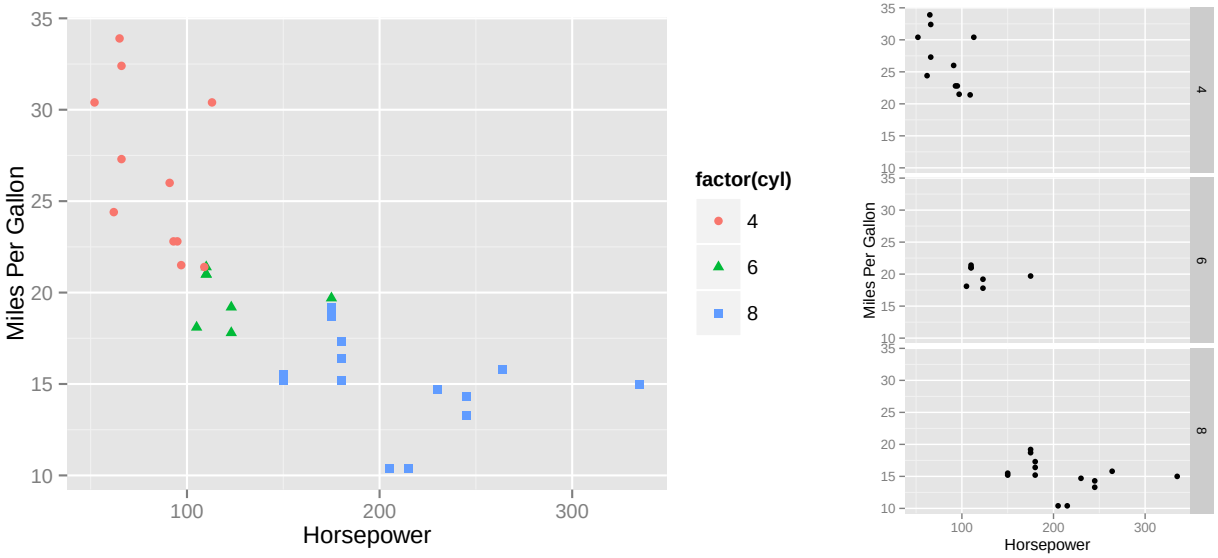
Note: Some ggplot2 examples use British spellings (i.e. colour instead of color). Both will work.



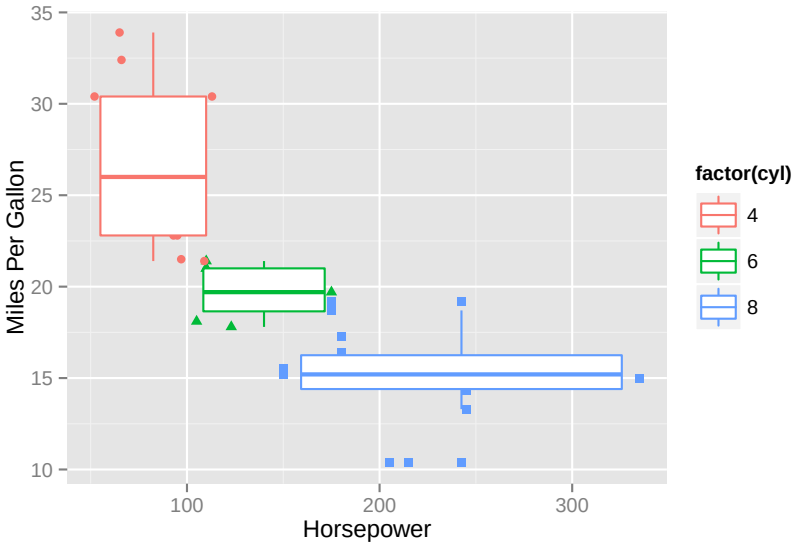

```
qplot(mpg, cyl, data=mtcars)
qplot(mpg, data=mtcars, binwidth=3,
        xlab="Miles Per Gallon", ylab="Frequency", fill=factor(cyl))
```

Example 3: Plotting horsepower against miles per gallon. Suppose we want to know how horsepower affects miles per gallon. In this case, a scatterplot would be appropriate!

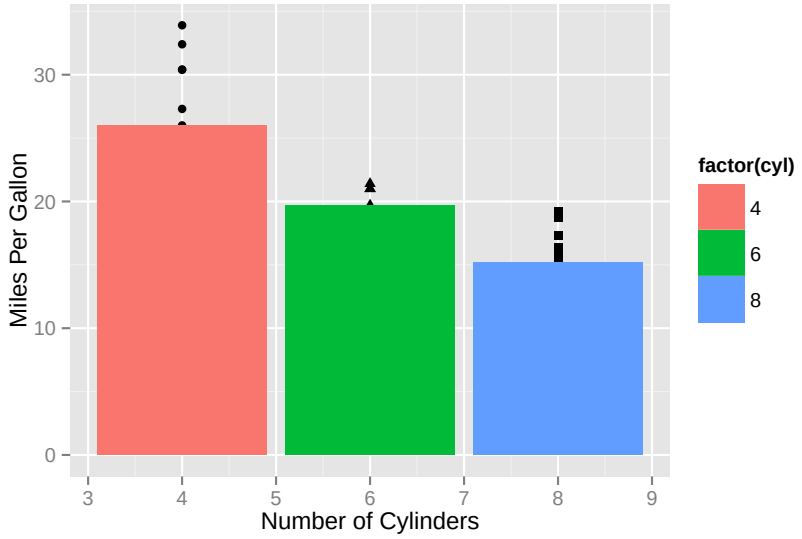
We might also want to separate our observations by number of cylinders. In the graph on the left, we've separated the number of cylinders by using different shapes and colors for our plotting characters. The graph on the right uses the facets option to create scatterplots in a 3 by 1 grid.



We might also want to get an idea of the variability in our data. By using a "+" option after our qplot() statement, we can add various statistics to our plots. For example, running the qplot() statement with "+stat_boxplot()" at the end adds a boxplot for each cylinder level. ggplot2 tries its best to ensure that the boxplots don't overlap.



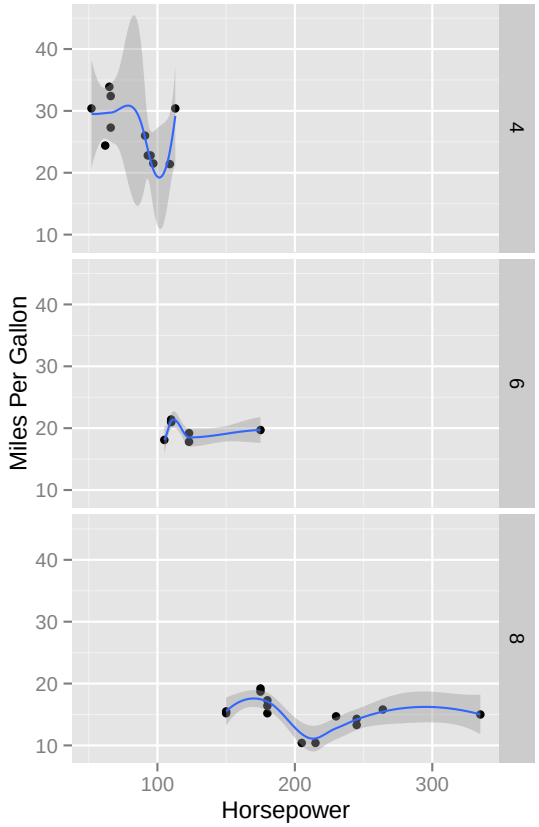
Another option is "+stat_summary". With this we can plot summary statistics such as mean, standard deviation, confidence limits, etc. over an existing graph. Below I've plotted the median miles per gallon for each engine type: 4, 6, or 8 cylinder.




```
qplot(x=hp, y=mpg, data=mtcars, xlab="Horsepower",
        ylab="Miles Per Gallon", shape=factor(cyl), color=factor(cyl))
        +stat_boxplot()
qplot(x=hp, y=mpg, data=mtcars, xlab="Horsepower",
        ylab="Miles Per Gallon", facets=cyl~.)
qplot(x=cyl, y=mpg, data=mtcars, xlab="Number of Cylinders",
        ylab="Miles Per Gallon", shape=factor(cyl), fill=factor(cyl))
        +stat_summary(fun.y = median, geom="bar")
```

Example 4: Adding a smoother. For our example above, we might want to visualize a smoothed line to make some rough predictions. We can specify a smoother by adding it to the geom option. ggplot2 chooses the appropriate smoother based on the number of sample points. The grey shading represents 95% confidence bands. The confidence level can be changed using a level option in a + statement.

Later on, we'll see an example where the + statement must be used. As a rule of thumb, if the base options are okay (for example using a 95% confidence level) you can specify multiple geoms in the original plot statement. However if you want to make changes to the defaults, it's easier to use a + statement.




```
qplot(x=hp, y=mpg, data=mtcars, xlab="Horsepower", ylab="Miles Per Gallon",
      facets=cyl~., geom=c("point", "smooth"))
```

Advantages so far of using ggplot2:

- Using qplot() really makes our lives as statisticians easier. Rather than calling multiple functions for different graphs, qplot() can quickly choose which is the best fit for our data!

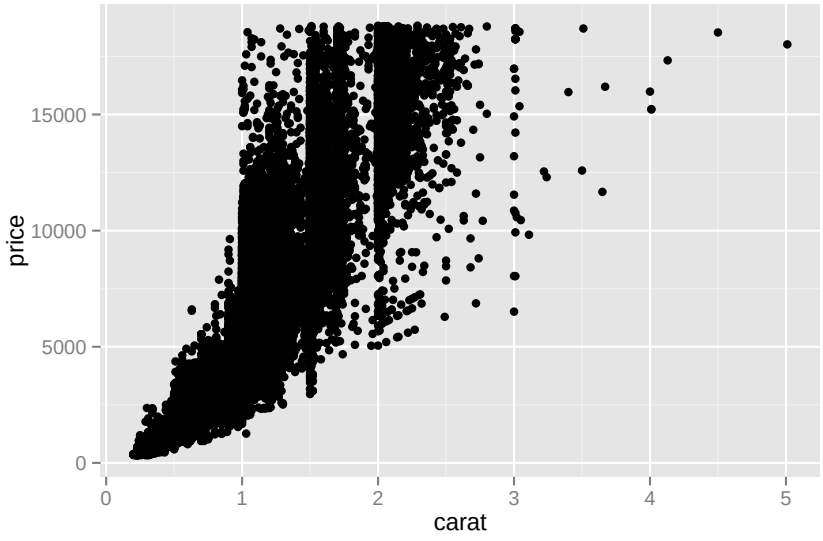
- Changing qplot() decisions is easy with the geom option.
- Adding legends to our graphs no longer requires a separate line of code, and is done automatically!
- Making lattice graphs is much quicker. All it takes is a basic formula specification, which as statisticians we should all be very comfortable doing!

However we're really just getting started with what ggplot2 can do.

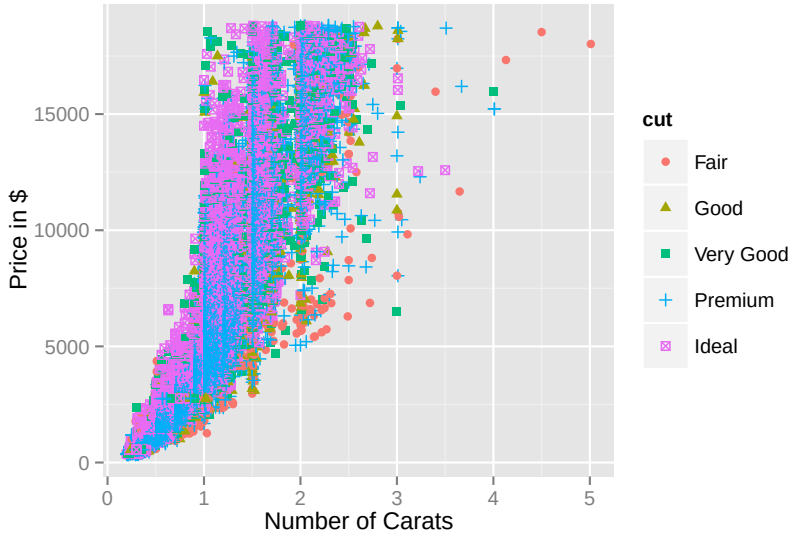
2.3 Getting Fancy

Let's take a look at another data set. "diamonds" is a prepackaged data set that contains about 50,000 observations! This data set includes carats, color, cut, clarity, and price. There aren't a lot of variables, but we have a lot of observations.

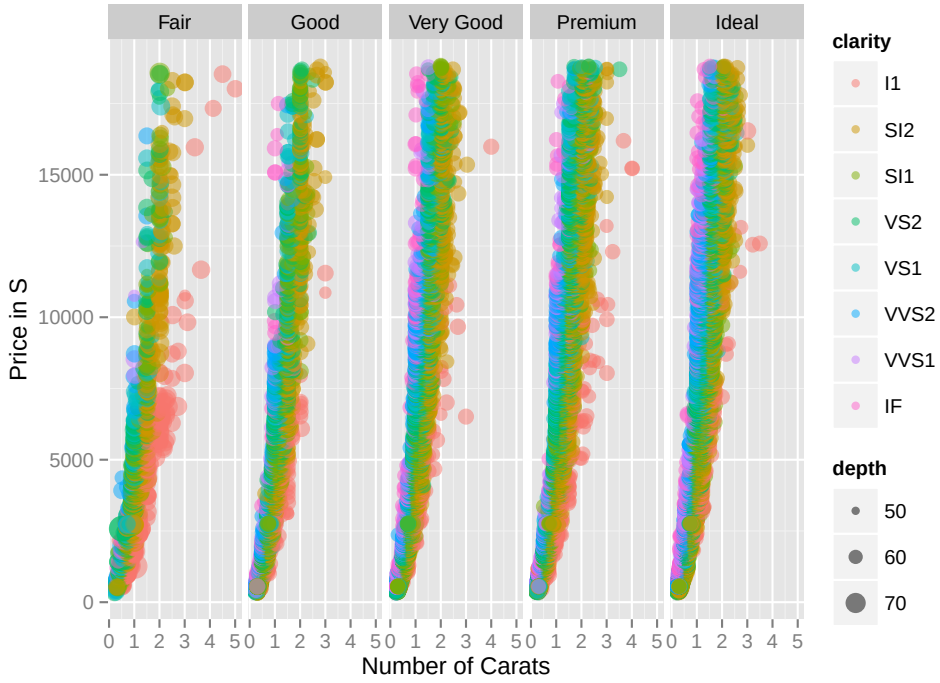
Example 5: Plotting carat against price. Diamonds are typically priced based on carats (size), cut, color, and clarity. Let's try to get a grasp of how carats affect the price.



Overall, as the size of a diamond increases, so does the price. What relationship would we expect the cut of a diamond to have on its price?



Cut has a small impact on price - but what we really see here is an interaction with size! With 50,000 observations, it might be helpful to break the data into subgroups.

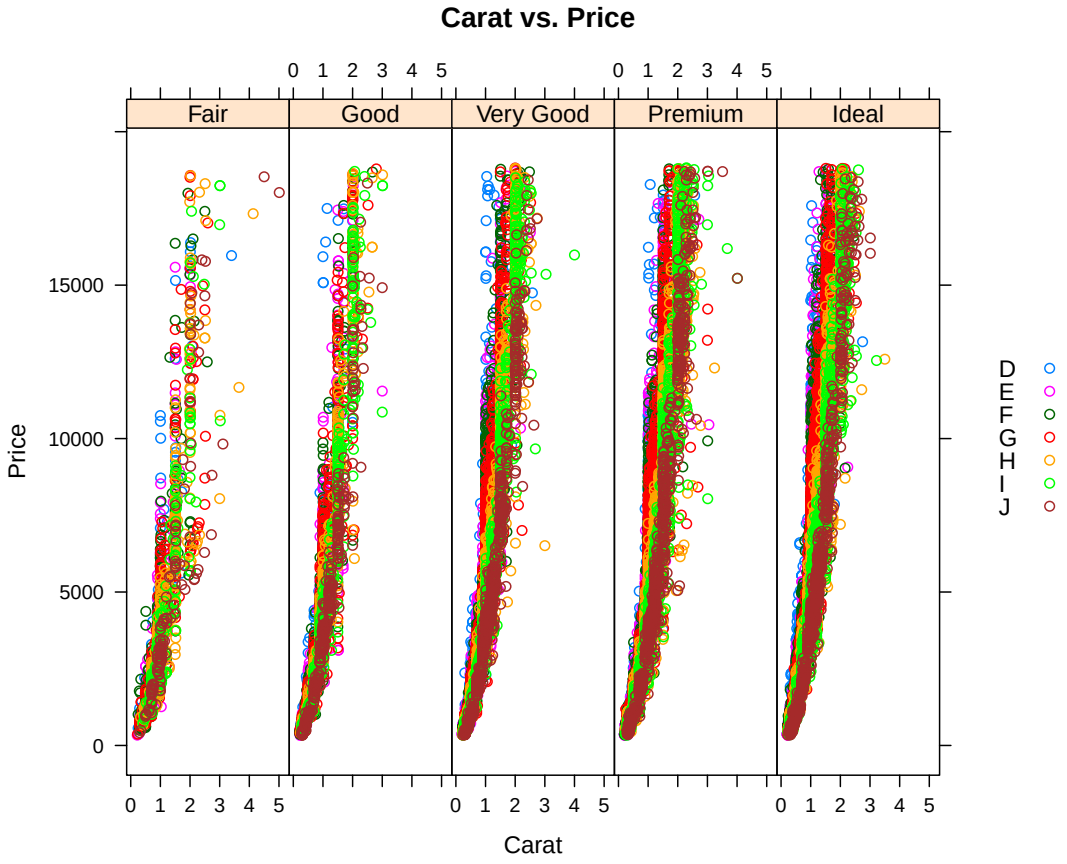


That's a lot of information! Here's what we've added:

- I've used the facets option to break the data up into 5 subsets of diamond cut.
- Each color represents a different level of clarity - IF is the best.
- The point size is proportional to the depth of the diamond.
- The opacity of the graph has been changed to handle overplotting.

We've condensed 5 different variables into one single plot!

Now, compare that graph to a plot using a subset of this data and the "lattice" package (thanks Chris).



The code here is similar in length, but we've left out a few options (scaling plot points and adjusting opacity).

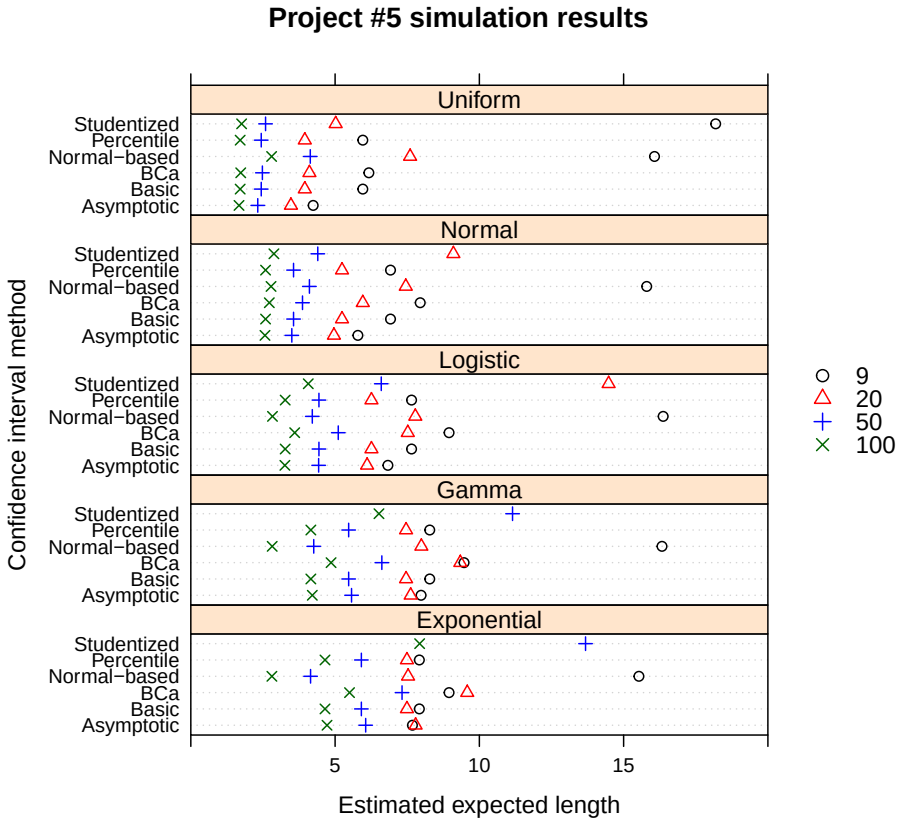
```
qplot(x=carat, y=price, data=diamonds)
qplot(x=carat, y=price, data=diamonds, xlab="Number of Carats",
        ylab="Price in $", color=cut, shape=cut)
qplot(x=carat, y=price, data=diamonds, xlab="Number of Carats",
        ylab="Price in $", color=cut, shape=cut, alpha=0.1)

#Final ggplot2 graph:
qplot(x=carat, y=price, data=diamonds, xlab="Number of Carats",
        ylab="Price in S", facets=.~cut, color=clarity, size=depth,
        alpha=I(0.5))

#Lattice graph:
library(lattice)
```

```
xyplot(x = price ~ carat | cut, data = diamonds,  layout = c(5,1),
        groups  = color, main = "Carat vs. Price", xlab = "Carat",
        ylab = "Price", auto.key = list(points = TRUE, space = "right"))
```

3 Is it really "prettier" than the base graphics?

During his lecture on parallel computing, Chris noted that a graph he used in his bootstrap class last fall might be a good contender to try to replicate in ggplot2. Challenge accepted.

Below is the original plot. The graph shows expected lengths for six different types of confidence intervals under five different distributions and four different sample sizes.

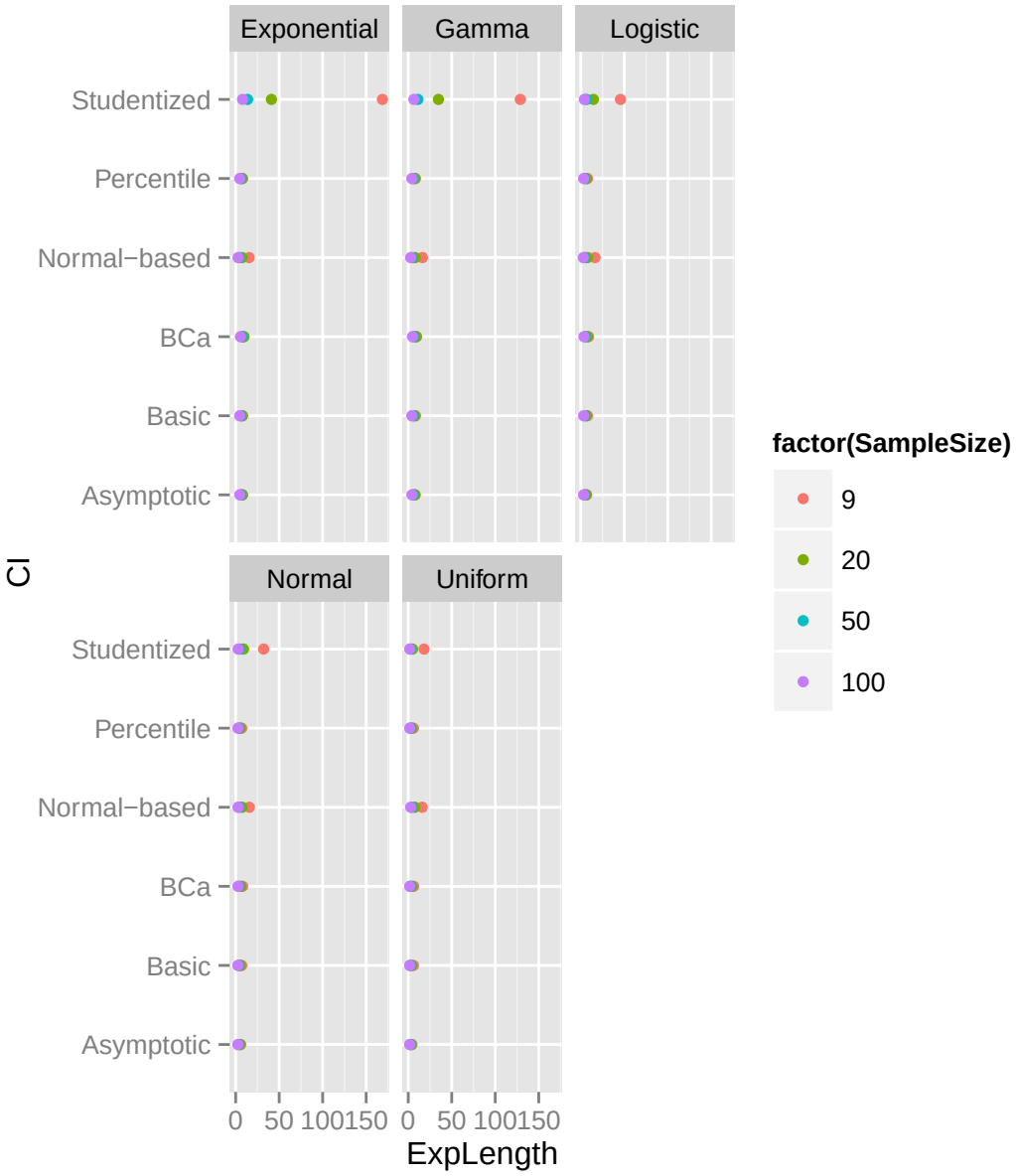


Here's the code you'd need to use to create this graph using the "lattice" package:

```
set1<-read.csv("sim_results.csv", head=TRUE)
library(package = lattice)
plot.levels<-levels(factor(set1$SampleSize))

#Expected length - restrict x-axis
dotplot(CI ~ ExpLength | Distribution, data = set1, groups = SampleSize,
        main = "Project #5 simulation results",
```

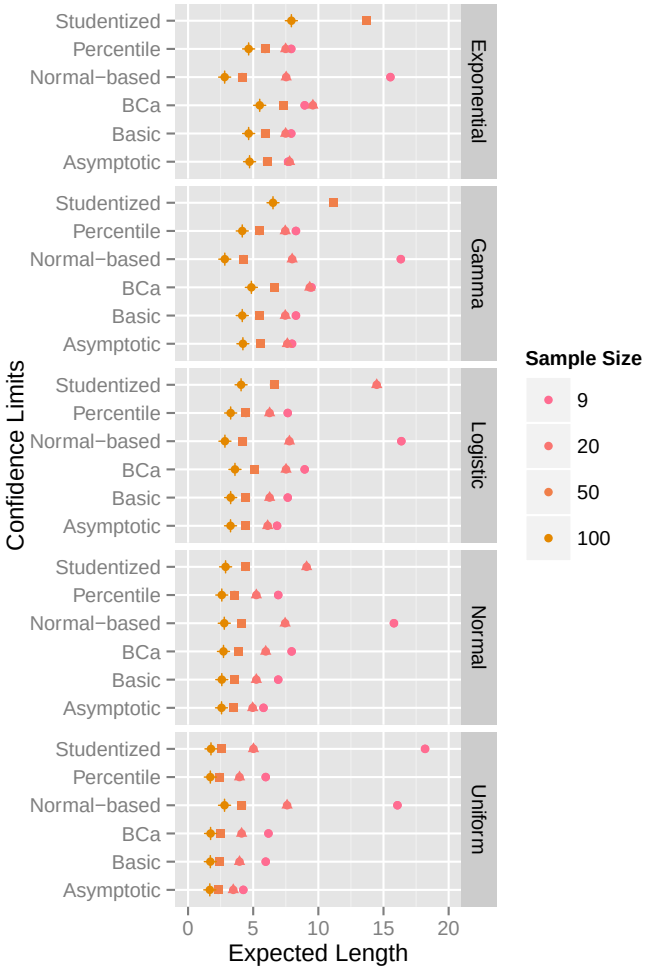
```
key = list(space = "right", points = list(pch = 1:4,
col = c("black", "red", "blue", "darkgreen")),
text = list(lab = plot.levels)), xlim = c(0, 20),
panel = function(x, y) {panel.grid(h = -1, v = 0, lty = "dotted",
        lwd = 1, col="lightgray"), panel.xyplot(x = x, y = y,
        col = c(rep("black", times = 6), rep("red", times = 6),
        rep("blue", times = 6), rep("darkgreen", times = 6)),
        pch = c(rep(1,6), rep(2,6), rep(3, 6), rep(4, 6)))},
xlab = "Estimated expected length", layout = c(1,5),
ylab = "Confidence interval method")
```

It's long. Now let's give ggplot2 a try. Using qplot() with only five inputs, I created the graph below. The code took one line!




```
qplot(y=CI, x=ExpLength, data=set1, color=factor(SampleSize),
      facets=~Distribution)
```

The "easiest" graph still might not be up to par. But there are lots of options available to spice it up! After playing around with some options, I ended with the graph below.



The code for my final graph is below.

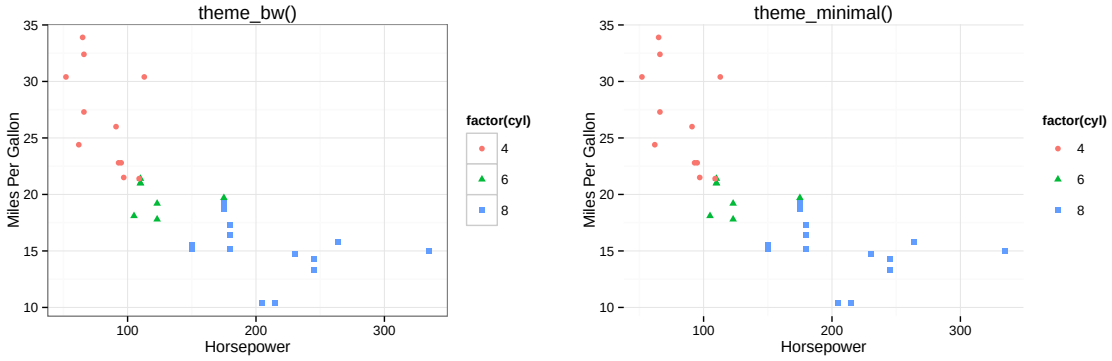
```
qplot(y=CI, x=ExpLength, data=set1, color=factor(SampleSize),
        facets=Distribution~., xlim=c(0, 20), ylab="Confidence Limits",
        xlab="Expected Length")
        +scale_color_hue("Sample Size", h=c(0, 40))
        +geom_point(aes(shape=factor(set1$SampleSize)), legend="none")
```

A few notes about the code:

- The colors I chose have no meaning, except that I like the color red. I specified the colors according to their hue (for more information on the HSL color scheme, check out Wikipedia). You could also specify colors by name or hex. Color palettes can be defined outside of ggplot2 using the RColorBrewer package.
- I specified the plotting shape in a "+" statement to avoid having multiple legends. Multiple legends don't become a problem until you start making changes. If I had listed the shape inside the qplot() function I would have had two legends.

- My final code is about half the length of the base R code. If I hadn't wanted to make changes to the colors/legend, it would have been even shorter. :)

One thing you may have noticed about plots in ggplot2 is that they default to a grey and white background. If that's not your cup of tea, you can also change the "theme" of the plots! Background colors can be specified by hand of course, but changing the theme is much quicker and often enough. Here are a few examples of preset themes. The default is "theme_grey()".



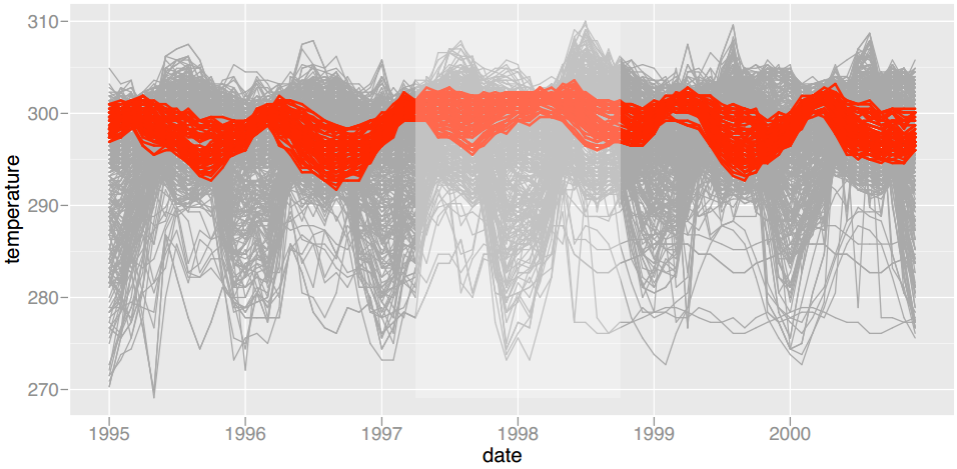

```
qplot(x=hp, y=mpg, data=mtcars, xlab="Horsepower", ylab="Miles Per Gallon",
      shape=factor(cyl), color=factor(cyl), main="theme_bw()")
      +theme_bw()
      +theme_minimal()
```

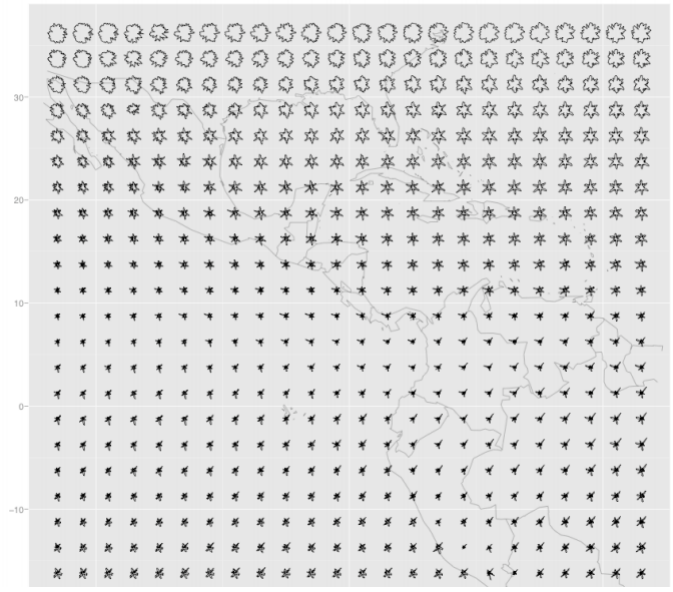
4 My Final Thoughts

ggplot2 is great. That's about it.

But seriously, as an experienced R user it did take some getting used to. I think that a new R user would have a much easier time picking up ggplot2. As I mentioned earlier, the graphs you've seen are just limited to exploratory data analysis. ggplot2 does a great job plotting more complicated statistical analyses, however it takes some work. More development is planned to advance graphing models to make it as user friendly as data graphing.

Some more advanced examples that I found particularly neat included a time series plot of temperature, clustered by ozone characteristics, and a star plot of ozone. Both graphs were made by Dr. Hadley Wickham, creator of ggplot2. Code is available on his website.





5 Available Resources

There are lots of available resources for working with ggplot2.

- ggplot2: Elegant Graphics for Data Analysis by Hadley Wickham. Data sets, sample chapters, and R code used in the book is available here, however some of the R code appears to be out of date. Chapter 2 of the book provides a detailed description of qplot(), this is a great place to start!
- ggplot2 documentation. The online documentation is full of examples, and sorted by desired plot type (with pictures!).
- Dr. Wickham gave a short (~45 minute) talk on using ggplot2 at the World Health

Organization in early 2010. The video covers similar material to what's in the documentation, but if you prefer the online video approach you can find it here.

- The Basel Institute for Clinical Epidemiology and Biostatistics has a nice PDF tutorial in ggplot2 available. There aren't a lot of explanations, just lots of code and lots of pictures. Perfect for an experienced R user!