An introduction to ggplot2

Sean C. Anderson

August 26, 2013

The ggplot philosophy: rapid data exploration

`ggplot2` is an R package that implements Wilkinson's Grammar of Graphics.[1] Hadley Wickham wrote the package as a chapter of his PhD thesis. Many people now participate in developing the package.

[1] Wilkinson, L. (2005). *The Grammar of Graphics*. Springer, 2[2nd] edition.

The emphasis of `ggplot` is on rapid exploration of data, and especially high-dimensional data. Think of base graphics functions as drawing with data. You have complete control over every pixel in a plot (once you learn the arcane world of `par`) but it can take a lot of time and code to produce a complex plot.

Although `ggplot` can be fully customized, I find it reaches a point of diminishing returns. I tend to use `ggplot` and base graphics for what they excel at: `ggplot` for rapid data exploration and base graphics for polished and fully-customized plots for publication.

The idea is simple: good graphical displays data require rapid iteration and lots of exploration. If it takes you hours to code a plot in base graphics, you're unlikely to throw it out and try something else. If it takes you hours to code a plot in base graphics, you're unlikely to explore other ways of visualizing the data or all the dimensions of the data.

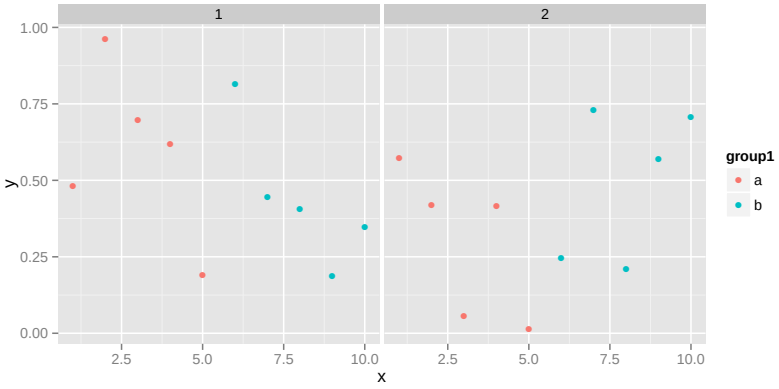
qplot vs. ggplot

There are two main plotting functions in the `ggplot2` package: `qplot` and `ggplot`. `qplot` is short for "quick plot" and is made to mimic the format of `plot` from base R. `qplot` requires less syntax for many common tasks, but has limitations — it's essentially a wrapper for `ggplot`. The `ggplot` function itself isn't complicated and will work in all cases. I prefer to work with just the `ggplot` syntax and will focus on it here.

Basics of the grammar

Let's look at some illustrative `ggplot` code:

```
>  d <- data.frame(x = c(1:10, 1:10), y = runif(20),
+                  group1 = rep(gl(2, 5, labels = c("a", "b")), 2),
+                  group2 = gl(2, 10))
>  ggplot(d) + geom_point(aes(x, y, colour = group1)) + facet_grid(~group2)
```



The basic format in this example is:

1. `ggplot()`: start a `ggplot` object and specify the data
2. `geom_point()`: we want a scatter plot; this is called a `geom`
3. `aes()`: specifies the "aesthetic" elements; a legend is automatically created
4. `facet_grid()`: specifies the panel layout

There are also statistics, scales, and annotation options, among others. At a minimum, you must specify the data, some aesthetics, and a geom. I will elaborate on these below. Yes, `ggplot` combines elements with `+` symbols![2]

[2] This may seem non-standard, although it has the advantage of allowing `ggplot` plots to be proper R objects, which can modified, inspected, and re-used.

Geoms

`geom` refers to a geometric object. It determines the "shape" of the plot elements. Some common geoms:

`geom`	description
`geom_point`	Points, e.g. a scatterplot
`geom_line`	Lines
`geom_ribbon`	Ribbons, y range with continuous x values
`geom_polygon`	Polygon, a filled path
`geom_pointrange`	Vertical line with a point in the middle
`geom_linerange`	An interval represented by a vertical line
`geom_path`	Connect observations in original order
`geom_histogram`	Histograms
`geom_text`	Textual annotations
`geom_violin`	Violin plot
`geom_map`	Polygons from a map

Aesthetics

Aesthetics refer to the attributes of the data you want to display. They map the data to an attribute (such as the size or shape of a symbol) and generate an appropriate legend. Aesthetics are specified with the `aes` function.

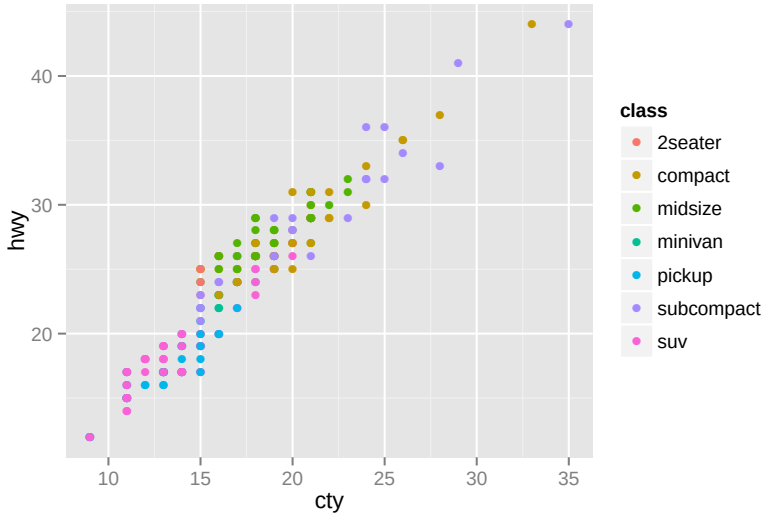
As an example, the aesthetics available for `geom_point` are: `x`, `y`, `alpha`, `colour`, `fill`, `shape`, and `size`.[3] Read the help files to see the aesthetic options for the geom you're using. They're generally self explanatory.

[3] Note that `ggplot` tries to accommodate the user who's never "suffered" through base graphics before by using intuitive terms like `colour`, `size`, and `linetype`, but `ggplot` will also accept terms such as `col`, `cex`, and `lty`.

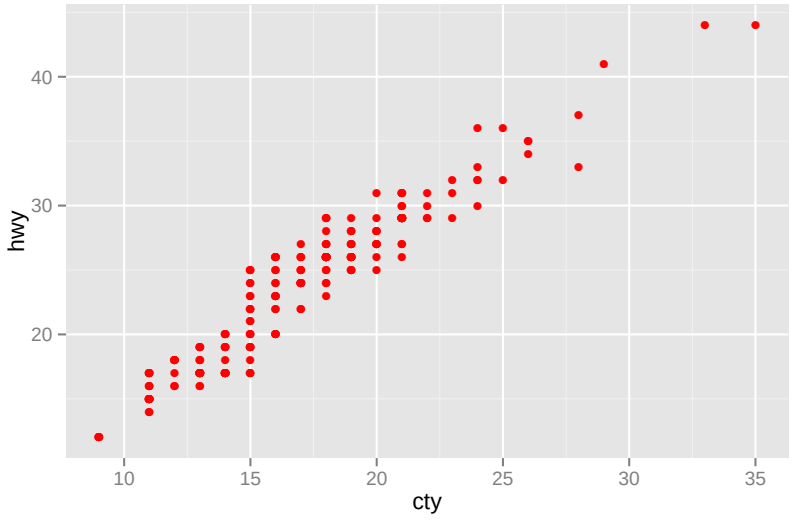
Aesthetics can be specified within the data function or within a geom. If they're specified within the data function then they apply to all geoms you specify.

Note the important difference between specifying characteristics like colour and shape inside or outside the `aes` function — those inside the `aes` function are assigned the colour or shape automatically based on the data. If characteristics like colour or shape are defined outside the `aes` function, then the characteristic is not mapped to data. Example:

```
> library(ggplot2)
> ggplot(mpg, aes(cty, hwy)) + geom_point(aes(colour = class))
```



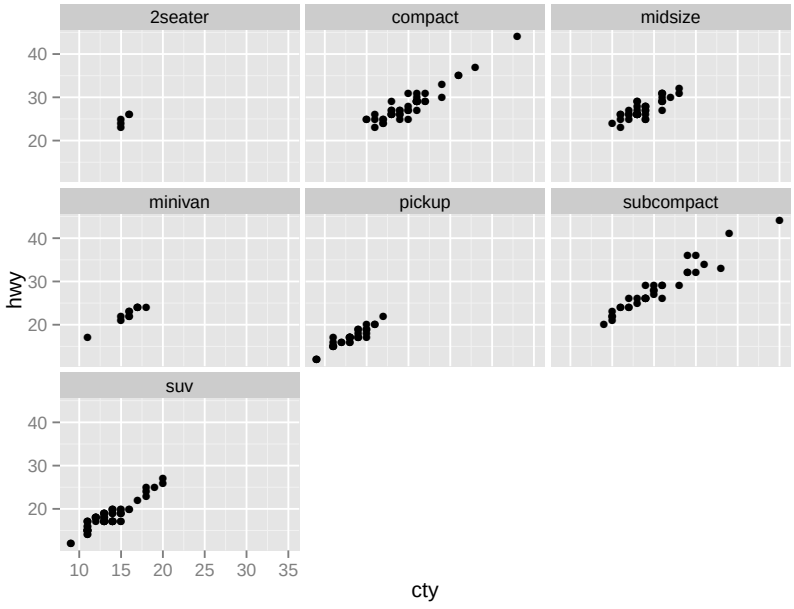

```
> ggplot(mpg, aes(cty, hwy)) + geom_point(colour = "red")
```



Small multiples

In `ggplot` parlance, small multiples are referred to as facets. There are two kinds: `facet_wrap` and `facet_grid`. This is where `ggplot` really shines.

```
> ggplot(mpg, aes(cty, hwy)) + geom_point() + facet_wrap(~class)
```



```
> ggplot(mpg, aes(cty, hwy)) + geom_point() + facet_grid(year~class)
```

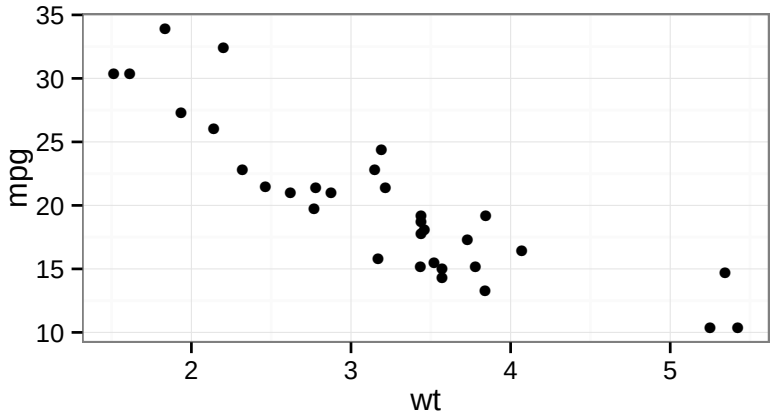
`face_wrap` plots the panels in the order of the factor levels. When it gets to the end of a column it wraps to the next column. You can specify the number of columns and rows with `nrow` and `ncol`. `facet_grid` lays out the panels in a grid with an explicit x and y position.

By default all x and y axes will be shared among panels. You could, for example, specify "free" y axes with `face_wrap(scales = "free_y")`.

Themes

A useful theme built into `ggplot` is `theme_bw`:

```
> dsamp <- diamonds[sample(nrow(diamonds), 1000), ]
> ggplot(mtcars, aes(wt, mpg)) + geom_point() + theme_bw()
```

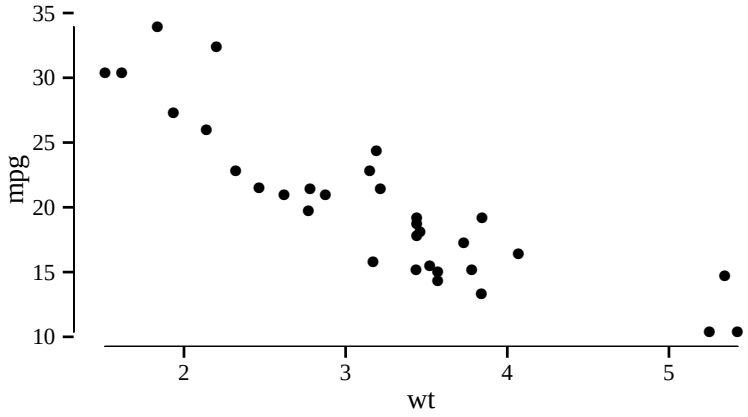


A powerful aspect of `ggplot` is that you can write your own themes. This feature of `ggplot` was recently expanded substantially, and I imagine we'll see more themes developed and shared in the future. See the `ggthemes` package for some examples.[4]

[4] Install the R package from: https://github.com/jrnold/ggthemes

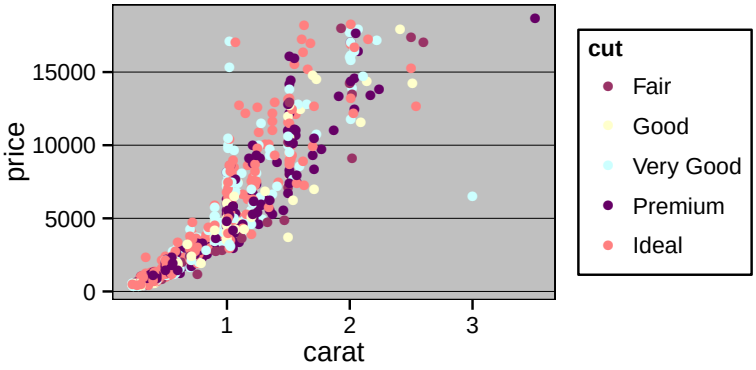
A Tufte-like theme:

```
> library("ggthemes")
> ggplot(mtcars, aes(wt, mpg)) + geom_point() + geom_rangeframe() +
+  theme_tufte()
```



Just what you wanted:

```
> qplot(carat, price, data = dsamp, colour = cut) + theme_excel() +
+  scale_colour_excel()
```



You can customize just about every aspect of a `ggplot` plot. We won't get into that today.

ggplot's dirty little secret

1. `ggplot` is easy to learn, however ...
2. you need to be a data-manipulation ninja to effectively use it

With base plot, you can work with almost any data structure you'd like providing you can write the code to work with it. `ggplot` requires you to think carefully about the data structure and then write one line of code.

`ggplot` works with "long" format data with each aesthetic or facet variable in its own column. So, for example, if we wanted a panel for each level of a factor called "fishstock" then we'd need a column named "fishstock" with all the different values of "fishstock" in that column.

With the `reshape` and `plyr` packages you can get almost any dataset into shape for `ggplot` in a few lines. Sometimes this will take some serious thought.

Help

The R help files were notoriously bad but are rapidly improving.

The best (and nearly essential) help source is the website.[5] There's also an active `ggplot` discussion group.[6] `ggplot` is heavily featured on stackoverflow.[7]

Hadley wrote a book on `ggplot`. It's quite thorough but doesn't feature some of the newer additions to `ggplot`.

[5] http://docs.ggplot2.org/

[6] http://groups.google.com/group/ggplot2

[7] http://stackoverflow.com/questions/tagged/ggplot2

Random tips

Jittering and statistics

`ggplot2` has lots of built in niceties like automatic position jittering and the addition of basic statistical models to your plots. Have a look through the website.

Axis labels

```
xlab("Your x-axis label")
```

Suppressing the legend

```
theme(legend.position = "none")
```

Exploiting the object-oriented nature of `ggplot2`

Save the basic plot to an object and then experiment with different aesthetics, geoms, and theme adjustments.

```
> p <- ggplot(d) + geom_point(aes(x, y, colour = group1))
> p <- p + facet_grid(~group2)
```

Horizontal error bars

Say you want to make a coefficient plot with the coefficients down the y-axis. You can either build the plot piece by piece with points and segments, or you can use `point_range()` and then rotate the axes with `+ coord_flip()`.

Axis limits and zooming

`ggplot2` has two ways of adjusting the axis limits: `+ xlim(2, 5)` will "cut" the data at `2` and `5` and plot the data. `coord_cartesian(xlim`

`= c(2, 5))` will zoom in on the plot while retaining the original data. This will, for example, affect colour scales.

Displaying and saving ggplot2 plots

If `ggplot2` plots are generated in a function, they will be need to be wrapped in `print()` to display. There is a convenience function `ggsave("filename.pdf")`, which will save the last plot to a pdf and guess at reasonable dimensions. You can, of course specify which plot to save and the dimensions of the pdf. You can also use all the same methods of saving `ggplot2` plots that you can use for base graphics.