An introduction to **ggplot2**

FISH 544: Beautiful Graphics in R

Sean C. Anderson
sean@seananderson.ca

February 14, 2014

1 The ggplot2 philosophy: rapid data exploration

ggplot2 is an R package by Hadley Wickham that implements Wilkinson's Grammar of Graphics.[1] The emphasis of **ggplot2** is on rapid exploration of data, and especially high-dimensional data. Think of base graphic functions as drawing with data[2]. With base graphics, you have complete control over every pixel in a plot but it can take a lot of time and code to produce a complex plot.

Although **ggplot2** can be fully customized, it reaches a point of diminishing returns. I tend to use **ggplot2** and base graphics for what they excel at: **ggplot2** for rapid data exploration and base graphics for polished and fully-customized plots for publication (Figure 1).

Good graphical displays of 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 explore other ways of visualizing the data, and you're unlikely to explore all the dimensions of the data.

2 `qplot()` vs. `ggplot()`

There are two main plotting functions in **ggplot2**: `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 — I find it easier to master one function that can do everything.

[1] Wilkinson, L. (2005). *The Grammar of Graphics*. Springer, 2[2nd] edition.
[2] Examples of base graphic functions are `plot()`, `points()`, and `lines()`.

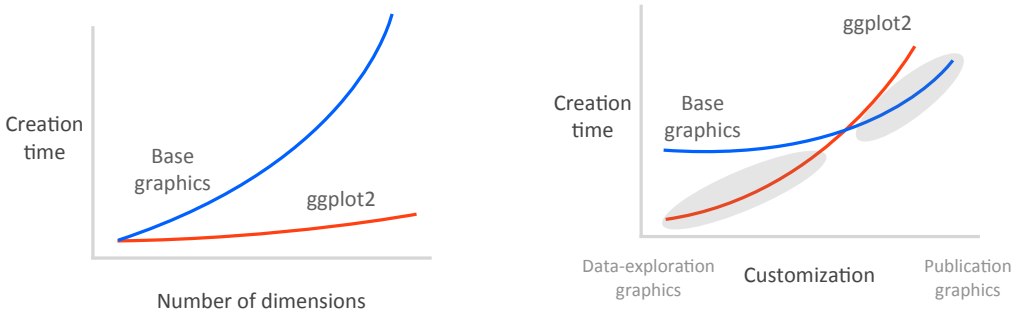


Figure 1: Creation time vs. data dimensions and customization level for base graphics (blue) and **ggplot2** (red). (Left panel) It's remarkably easy to plot high-dimensional data in **ggplot2** with, for example, colours, sizes, shapes, and panels. (Right panel) **ggplot2** excels at rapid visual exploration of data, but has some limitations in how it can be customized. Base graphics are fully customizable but can take longer to set up. I try and exploit the grey shaded areas: I use **ggplot2** for data exploration and once I've decided on a small number of key plots, I'll use base graphics to make fully-customized plots if needed.

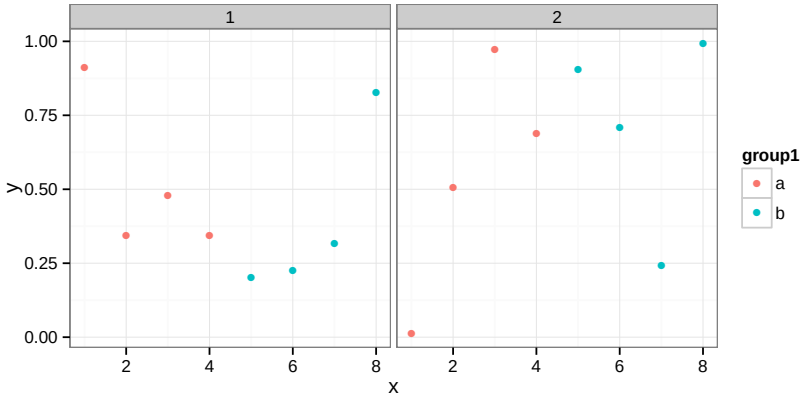
3 Basics of the grammar

Let's look at some illustrative **ggplot2** code:

```
library(ggplot2)
theme_set(theme_bw()) # use the black and white theme throughout
# fake data:
d <- data.frame(x = c(1:8, 1:8), y = runif(16),
  group1 = rep(gl(2, 4, labels = c("a", "b")), 2),
  group2 = gl(2, 8))
head(d)
```

```
##   x      y group1 group2
## 1 1 0.9120      a      1
## 2 2 0.3453      a      1
## 3 3 0.4775      a      1
## 4 4 0.3442      a      1
## 5 5 0.2009      b      1
## 6 6 0.2270      b      1
```

```
ggplot(data = 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 "faceting" or 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, **ggplot2** combines elements with + symbols! This may seem non-standard, although it has the advantage of allowing **ggplot2** plots to be proper R objects, which can modified, inspected, and re-used (I provide some examples at the end).

4 Geoms

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

geom	description
geom_point()	Points
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()	Text annotations
geom_violin()	Violin plot (another name for a beanplot)
geom_map()	Polygons from a map

5 Aesthetics

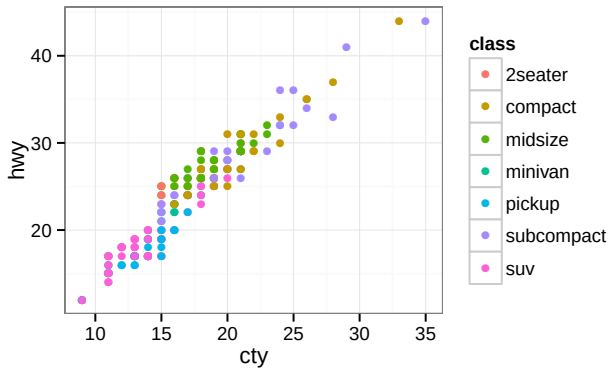
Aesthetics refer to the attributes of the data you want to display. They map the data to an attribute (such as the size or shape of a symbol) and generate an appropriate legend. Aesthetics are specified with the `aes()` function.

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. 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.

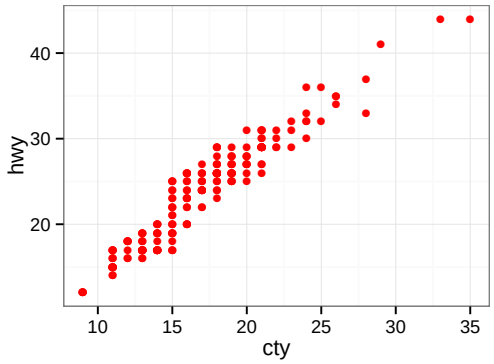
Note the important difference between specifying characteristics like colour and shape inside or outside the `aes()` function: those inside the `aes()` function are assigned the colour or shape automatically based on the data. If characteristics like colour or shape are defined outside the `aes()` function, then the characteristic is not mapped to data. Here's an example:

[3]Note that **ggplot2** tries to accommodate the user who's never "suffered" through base graphics before by using intuitive arguments like `colour`, `size`, and `linetype`, but **ggplot2** will also accept arguments such as `col`, `cex`, and `lty`.

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



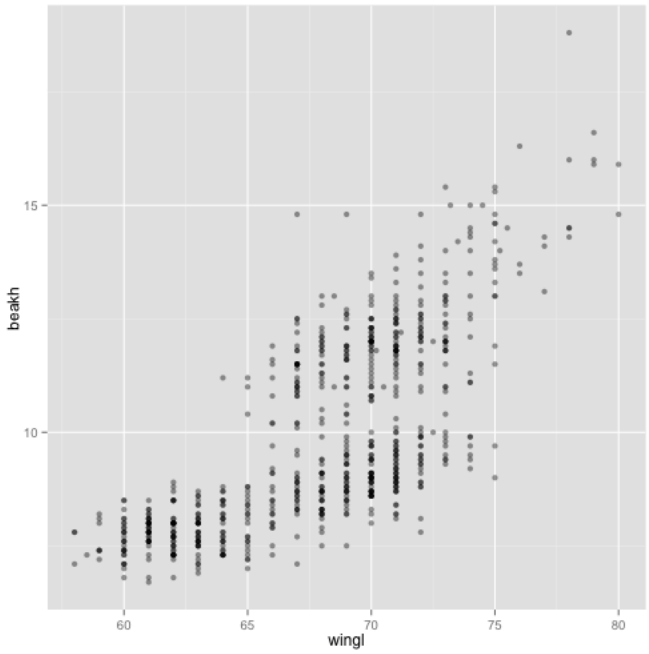

```
ggplot(mpg, aes(cty, hwy)) + geom_point(colour = "red")
```



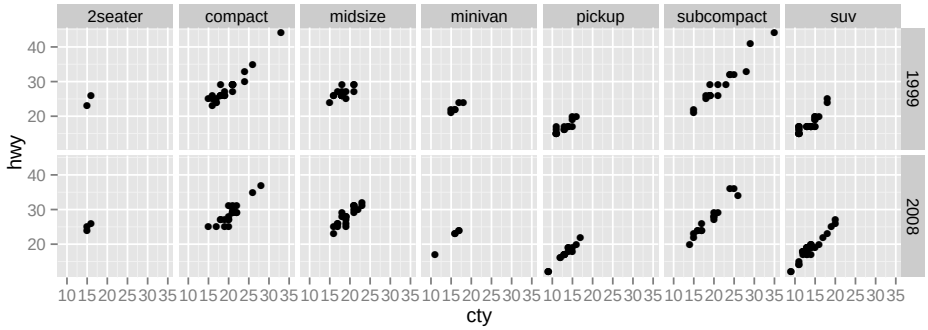
6 Small multiples

In **ggplot2** parlance, small multiples are referred to as "facets". There are two kinds: `facet_wrap()` and `facet_grid()`. `facet_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. However, you could, for example, allow the y axes to vary with `facet_wrap(scales = "free_y")` or allow all axes to vary with `facet_wrap(scales = "free")`.

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



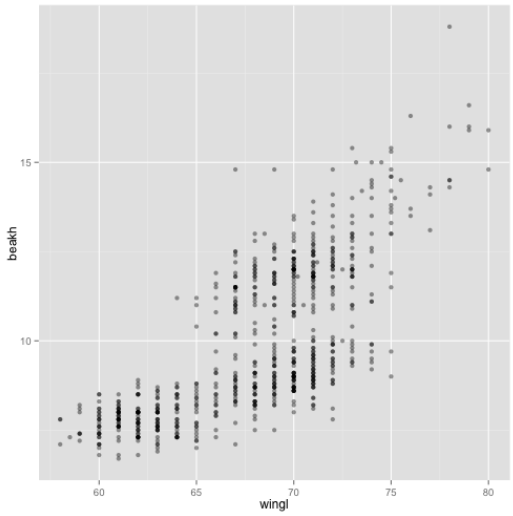

```
ggplot(mpg, aes(cty, hwy)) + geom_point() + facet_grid(year~class)
```



7 Themes

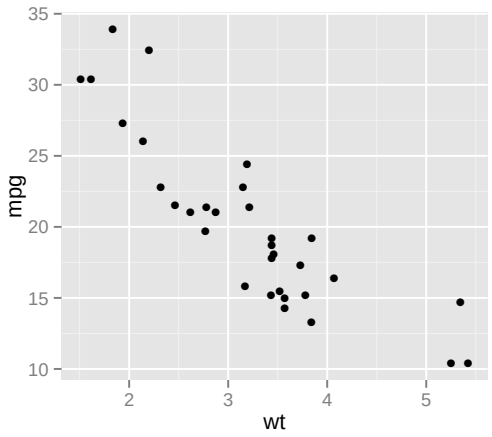
A useful theme built into **ggplot2** is `theme_bw()`. You'll notice that I set it as the default in this document back when I first loaded **ggplot2**. You can specify it for a specific plot like this:

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



Alternatively, the following plot shows the default theme. The grey is designed to match the typical darkness of a page filled with text to keep the plot from drawing too much attention.

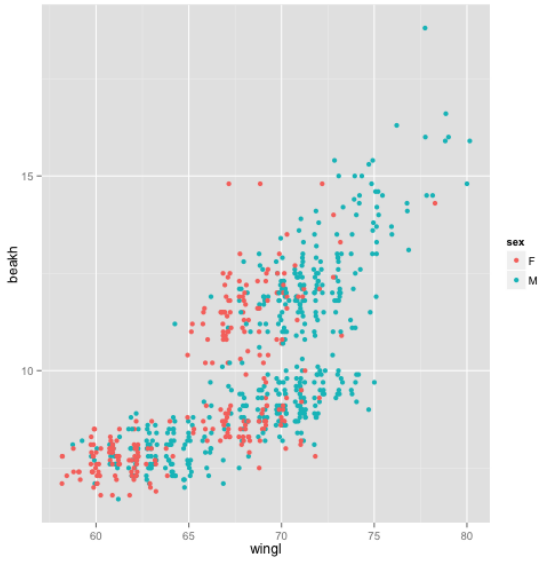
```
ggplot(mtcars, aes(wt, mpg)) + geom_point() + theme_grey()
```



A powerful aspect of **ggplot2** is that you can write your own themes. See the **ggthemes** package for some examples.

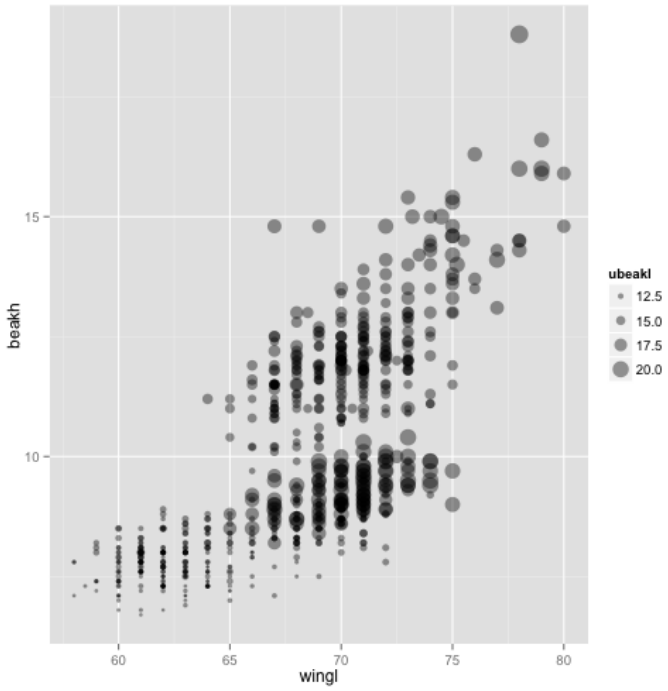
An Edward Tufte-like theme:

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



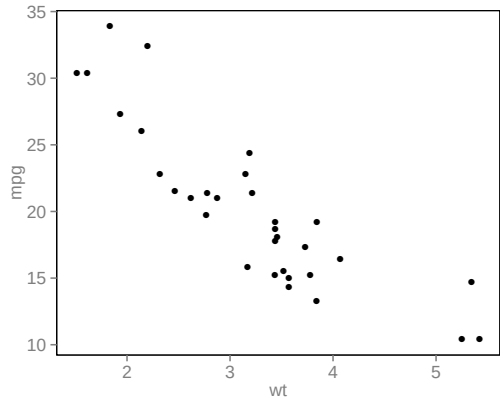
Just what you wanted:

```
ggplot(dsamp, aes(carat, price, colour = cut)) + geom_point() +
  theme_excel() + scale_colour_excel()
```



Here's an example custom theme:

```
theme_example <- function (base_size = 12, base_family = "") {
  theme_gray(base_size = base_size, base_family = base_family) %+replace%
    theme(axis.text = element_text(colour = "grey50"),
          axis.title.x = element_text(colour = "grey50"),
          axis.title.y = element_text(colour = "grey50", angle = 90),
          panel.background = element_rect(fill = "white"),
          panel.grid.minor.y = element_blank(),
          panel.grid.minor.x = element_blank(),
          panel.grid.major = element_blank(),
          plot.background = element_blank()
  )
}
ggplot(mtcars, aes(wt, mpg)) + geom_point() + theme_example()
```



You can customize just about every aspect of a **ggplot2** plot. We won't get into that today, but see the additional help resources at the end of this document.

8 ggplot2's dirty little secret

1. **ggplot2** is easy to learn, but...
2. You need to be a data-manipulation ninja to use it effectively.

With base graphics, you can work with almost any data structure you'd like, providing you can write the code to work with it. For example, you data could be in a list or a data frame. Or maybe it's scattered across multiple objects. **ggplot2**, on the other hand, requires you to think carefully about the data structure and then write one (possibly long) line of code. So, with base plotting you'll tend to spend lots of time writing plotting code. With **ggplot2** you'll tend to spend time thinking beforehand, and then quickly write the code.

`ggplot()` works with "long" format data or "tidy data" with each aesthetic or facet variable in its own column[4]. So, for example, if we wanted a panel or colour for each level of a variable called `fishstock` then we'd need a column named `fishstock` with all the different values of `fishstock` in that column.

With the **reshape2** and **plyr** packages, combined with `merge()` (or `plyr::join()`), you can get almost any dataset into shape for **ggplot2** in a few lines. Sometimes this will take some serious thought along with pen and paper.

[4]See Hadley Wickham's pre-print on tidy data here: `http://vita.had.co.nz/papers/tidy-data.pdf`

9 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 help resources listed at the end of this document.

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.

```
# Build the basic object:
p <- ggplot(d) + geom_point(aes(x, y, colour = group1))
p + facet_grid(~group1) # try one way
p + facet_grid(~group2) # try another way
```

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.

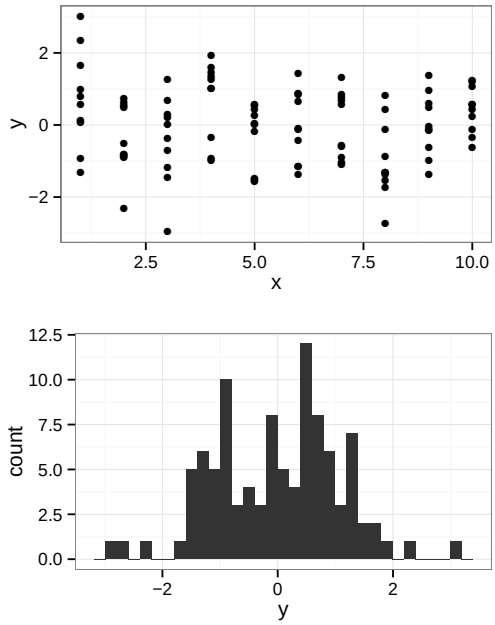
Displaying and saving ggplot2 plots

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.

Combining multiple ggplot2 panels

ggplot2 makes it easy to create multipanel figures with `facet_grid()` and `facet_wrap()`. But, sometimes you need to combine different kinds of plots in one figure or plots that require different data. One easy way to do this is with the `grid.arrange()` function from the **gridExtra** package. For example:

```r
df <- data.frame(x = 1:10, y = rnorm(100))
p1 <- ggplot(df, aes(x, y)) + geom_point()
p2 <- ggplot(df, aes(y)) + geom_histogram()
gridExtra::grid.arrange(p1, p2)
```



ggvis

ggvis is a partial successor to **ggplot2** and is under rapid development: `https://github.com/rstudio/ggvis`. It focuses on implementing a similar grammar of graphics for web-based output. Take a look at the vignettes — start with `vignette("ggvis-basics")` — to see some examples. You'll need to install the **devtools** R package first.

10 Help

The best (and nearly essential) help source is the website, which is largely based off the package help files. However, the website also shows the example plots and is easier to navigate:
`http://docs.ggplot2.org/`
Don't be afraid to keep it open whenever you're using **ggplot2**.

There's also an active **ggplot2** discussion group:
`http://groups.google.com/group/ggplot2`

ggplot2 is heavily featured on stackoverflow:
`http://stackoverflow.com/questions/tagged/ggplot2`

Hadley wrote a book on **ggplot2**: `http://ggplot2.org/book/`. It's quite thorough but doesn't feature some of the newer additions to the package.