Article type: Focus Article

ggplot2 593

Department of Statistics, Rice University

Hadley Wickham

Department of Statistics MS-138
Rice University
P. O. Box 1892
Houston, TX 77251-1892
hadley@rice.edu

Keywords

visualisation, statistical graphics, R

Abstract

ggplot2 is an open source R package that implements the layered grammar of graphics [Wickham, 2010], an extension of Wilkinson's grammar of graphics [Wilkinson, 2005]. This article provides an overview of ggplot2 and the ecosystem that has built up around it. I'll focus on the features that make ggplot2 different from other plot systems (the underlying theory and the programmable nature), as well as some of the important features of the community.
This article begins with a reminder about the motivation for visualisation software, then continues to discuss three particularly special features of ggplot2: the underlying grammar, its programmable nature and the ggplot2 community.

Data analysis

When creating visualisation software, it is useful to think about why we create visualisations: not to create pretty pictures, but to better understand our data. Visualisation is just part of the data analysis process, as shown in Figure1, and it needs to be coupled with transformation and modelling to build understanding. ggplot2 has been designed with this in mind. Because ggplot2 is embedded within R [R Development Core Team, 2010], we can use ggplot2 for visualisation and other R packages can provide tools for transformation and modelling. All that is required is a common data format, and ggplot2 works with data in "long" format, where variables are stored in columns and observations in rows. This means that you don't need to change the format of your data as you iterate between modelling, transforming and visualising.

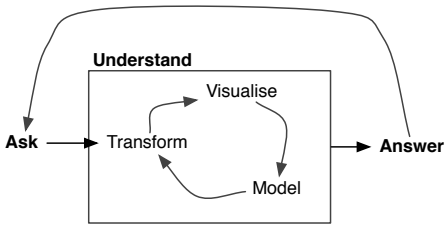

Figure 1: The data analysis cycle

A grammar of graphics

Focussing on just the visualisation component of the cycle, we ask two questions over and over again: what should we plot next and how can we make that plot? ggplot2 focuses on the second question: once you have come up with a plot in your head, how can you render it on screen as quickly as possible? Most graphics packages, like base graphics [Murrell, 2005] and lattice graphics [Sarkar, 2008] in R, start with a posse of named graphics, like scatterplots, pie charts, and histograms, and a handful of primitives, like lines and text. To create a plot, you figure out the closest named graphic and then tweak plot parameters and add primitives to bring your idea to life. For complicated graphics, code is usually imperative: draw a line here, draw text there, do this, do that, and you have to worry about many low-level details.

If you're using a plotting system with an underlying grammar, such as ggplot2 or Wilkinson's GPL, you take a different approach. You think about how your data will be represented visually, then describe that representation using a declarative language. The declarative language, or grammar, provides a set of independent building blocks, similar to nouns and verbs, that allow you to build up a plot piece by piece. You focus on describing what you want, leaving it up to the software to draw the plot.

Transitioning from the first approach to the second is often frustrating, because you have to give up much of the control that you are used to. It's much like learning Latex after learning MS Word: at first you are frustrated by how little control you have, but eventually the restrictions become freeing, leaving you to concentrate on the content, not the appearance. Similarly, learning ggplot2 can be frustrating if you're familiar with other plotting systems, because controlling low-level aspects of plot appearance is considerably more difficult in ggplot2. However, the trade-off is worth it: once you give up this desire for low-level control, you can create richer graphics much more easily.

The following example gives a small flavour of ggplot2 and the grammar, by showing how to create a waterfall chart.

Case study: a waterfall chart

The following example is inspired by an example from the Learn R blog[1]. It shows how to create a waterfall chart, often used in business to display flows of income and expenses over time. Figure 2 shows a typical example, which we'll recreate in two phases, first focussing on the essential structure of the plot, and then tweaking the appearance. This is a similar breakdown to exploratory vs. communication graphics: first you figure out the best plot for the job with rapid iterations and once you have you spend more time polishing it for presentation to others.

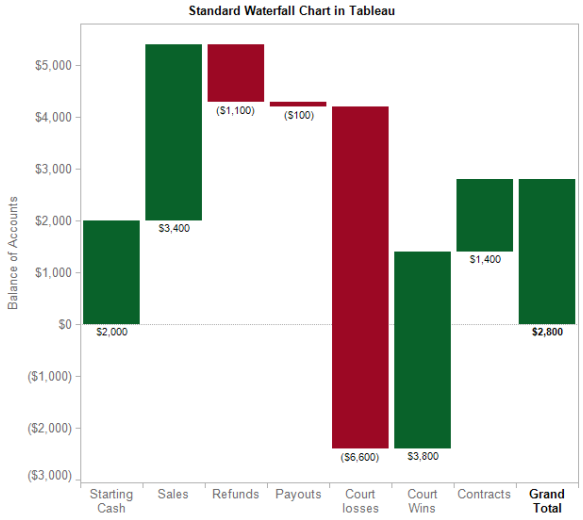

Figure 2: A waterfall chart showing balance changes in a fiction company. Used with permission from Stephen McDaniel at Freakalytics.com.

To recreate this plot with ggplot2 we start by thinking about underlying data and how it is represented. What data does the plot display and how does it display it? The most striking feature are the rectangles that display the change in balance for each event. We could represent that data in R with the following data frame:

```
balance <- data.frame(event = c("Starting\nCash", "Sales", "Refunds",
  "Payouts", "Court\nLosses", "Court\nWins", "Contracts", "End\nCash"),
  change = c(2000, 3400, -1100, -100, -6600, 3800, 1400, -2800))
```

A little thought reveals an additional variable that we need: time. That is what the x axis displays, even though it is labelled with the event. If it's not clear why we need

[1] `http://learnr.wordpress.com/2010/05/10/ggplot2-waterfall-charts`, inspired in turn by `http://www.freakalytics.com/2009/11/17/wc/`

this extra variable, think about a company that received two payouts: we would not want to place them in the same position on the x-axis. It's also useful to add two other variables: the running balance, used to draw the starting point of the bar, and the direction of the flow, used to colour the bars. ggplot2 does provide ways to do simple statistical transformations like this within the plot, but when creating a new plot it's often better to start by doing the manipulations yourself so you know exactly what's going on. The following R code adds those variables to the data.

```
balance$balance <- cumsum(c(0, balance$change[-nrow(balance)]))
balance$time <- 1:nrow(balance)
balance$flow <- factor(sign(balance$change))
```

To make the plot, we'll first recreate the rectangles that show the change in the balance. In ggplot2, the geometric object used to represent the data is called a **geom**, and the geom used to draw rectangles is called `geom_rect`. The following code creates a new plot using the balance data, and then adds on a layer of rectangles, with one rectangle for each observation centred horizontally on time, positioned vertically to run between the current and previous balance. The fill colour of the rectangle is mapped to the flow. This produces Figure 3. You can see that this displays the essence of the data, but is not formatted quite as we desire.

```
ggplot(balance) +
  geom_rect(aes(xmin = time - 0.45, xmax = time + 0.45, ymin = balance, ymax
```

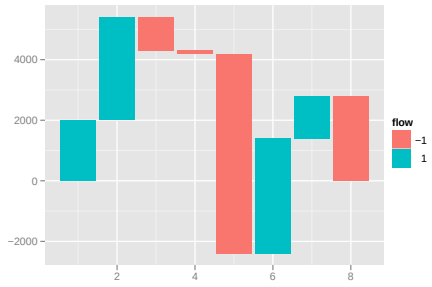

Figure 3: Rectangles display change in balance at each time point.

Before we tweak the formatting, we need to add a layer of text, which is centred about time and placed at the bottom of the balance bar - the minimum of the previous balance and the new balance. We use the convenient dollar formatting function to make nicer labels. The results are shown in Figure 4.

```
ggplot(balance) +
  geom_rect(aes(xmin = time - 0.45, xmax = time + 0.45, ymin = balance, ymax
  geom_text(aes(x = time, y = pmin(balance, balance + change) - 50, label =
```

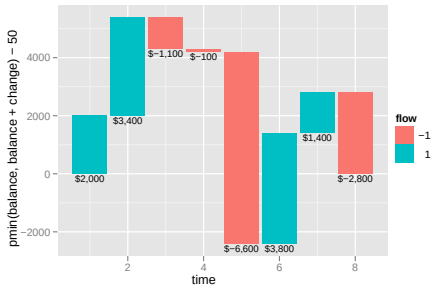

Figure 4: caption

Now we need to focus on the presentation of the plot. During the exploratory phase many of the graphics you create will end up in the trash and you want tools that make iteration as fast as possible. You are already intimately familiar with the data, so you don't need to fuss with nice labels, you just need the bare minimum of legends and axes to accurately decode the plot. As you transition towards the presentation/communication phase of your analysis, you want to spend more time making the plot easy to understand for those who aren't already familiar with the data.

To improve the waterfall chart, we add an additional layer to show zero more clearly, but most of the work is on tweaking the scales. Scales are in charge of the mapping from data to perceptual properties, like position, shape and colour. Scales are also in charge of the inverse operation: creating the guides (legends and axes) which allow us to decode the data. Typical reasons for modifying the scale are to tweak the choice of outputs for the target medium (e.g. black and white printer, large scale display with projector), to meet existing conventions or standards, or to better match the data type with the scale type. Here we modify the scales to meet existing conventions as follows:

- x: put a tick at every time point, and label with the description
- y: give a nicer title and format the values nicely
- fill: make negative flows red and positive flows black, and don't display a legend (we use these colours to match existing financial standards to avoid problems for people with red-green colour blindness)

```
ggplot(balance) +
  geom_hline(yintercept = 0, colour = "white", size = 2) +
  geom_rect(aes(xmin = time - 0.45, xmax = time + 0.45, ymin = balance, ymax = balanc
  geom_text(aes(x = time, y = pmin(balance, balance + change) - 50, label = dollar(ch
    hjust = 0.5, vjust = 1, size = 3) +
  scale_x_continuous("", breaks = balance$time, labels = balance$event) +
  scale_y_continuous("Balance", formatter = dollar) +
  scale_fill_manual(values = c("-1" = "red", "1" = "black"), legend = F)
```

Figure 5 shows the results of these operations. The second graphic shows an alternative representation that uses themes to modify the overall display. Themes control all non-data aspects of the plot, and are largely useful to override the default display properties of ggplot2 for meet publication requirements. The code to produce this graphic is included in the online supplementary materials.

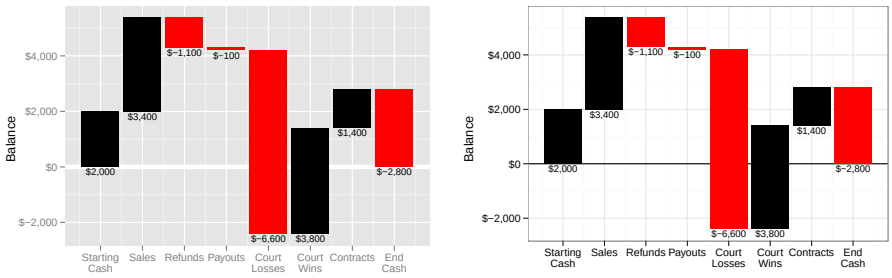

Figure 5: Two examples of polished versions of the waterfall chart. (Left) Modifying the scales makes the plot easier to understand, and (right) modifying the theme gives a traditional white background with black gridlines.

Programming graphics

As well as the underlying grammar, another special feature of ggplot2 is that it is code based and programmable. This is typical for R, but unusual in general: most visualisation tools use a graphical user interface. Programmability is important because it facilitates good science, which needs reproducibility, automation and communication:

- If others can't **reproduce** what you have done, then they are unlikely to believe your results. With R code, it is possible to reproduce an analysis exactly as it was performed previously.
- If reproducibility is about bringing the past into the present, **automation** is about bringing the present into the future. Most data sources are not static, but change over time, whether that's known in advance or as errors are fixed as analysis proceeds. A script that reproduces an analysis can be easily re-run whenever the data changes.
- Code also makes **communication** much easier because it can be copied and pasted; compare that with the difficulties of communicating a work flow in Excel.

One place that this is particularly useful is in graphical inference, a toolkit for checking that what you see in a plot is really there [Wickham et al., (In press, Buja et al., 2009].

Figure 6 shows eight scatterplots that display the position of all three point shots attempted by the Los Angeles lakers in the 2008/09 season. Each attempt is coloured by whether it was successful or not. This type of plot is called a lineup, because of the eight plots, only one is of the real data. The other seven were generated under the null hypothesis that position and success are unrelated. Can you pick out a plot that looks different? If you can, and it's the real data, then we have some evidence ($p = 1/8$) that there is a real difference. (Answer: The real data is in panel 2 — did you spot it?)

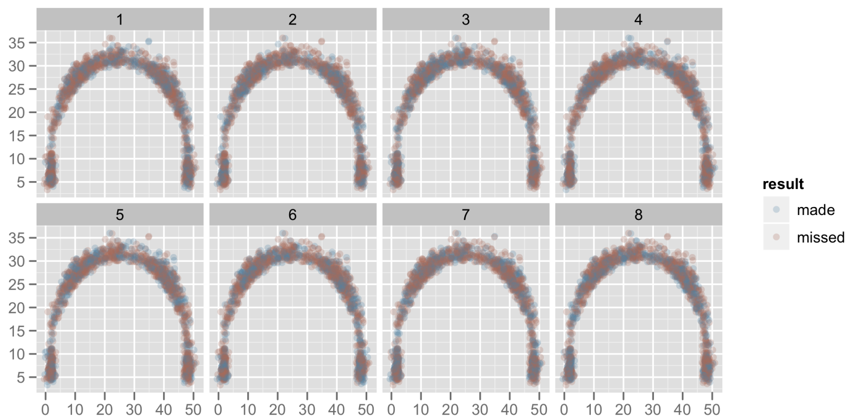

Figure 6: Eight scatterplots show the location of three pointers attempted by the Los Angeles Lakers; the shots are coloured by outcome. One plot shows the real data and seven plots show data from the null hypothesis. Can you spot the real data?

The power of a programmatic approach to visualisation is illustrated by the code used to create this plot, facilitated by the nullabor package[2]. The essence of the code is shown below. Note the second line: it defines the method used to generate samples from the null hypothesis (permuting the result variable), and the total number of panels we want (8) in a natural extension to the grammar. This makes graphical inference easy to plug in to any existing visualisation created with ggplot2.

```
ggplot(threept, aes(x, y)) %+%
  lineup(permute("result"), n = 8) +
  geom_point(aes(colour = result, group = 1), alpha = 1/4) +
  scale_colour_manual(values = c("made" = mnsl("10B 6/6"),
    "missed" = mnsl("10R 6/6"))) +
  xlab(NULL) + ylab(NULL) + coord_equal() +
  facet_wrap(~ .sample, nrow = 2)
```

[2]Available from `https://github.com/ggobi/nullabor`

ggplot2 ecosystem

A vital part of ggplot2 is the community and ecosystem that has built up around it. As well as the ggplot2 website, which provides an online version of the documentation, there are three main components to the ggplot2 ecosystem: the mailing list, the wiki, and code contributions.

Mailing list

The mailing list is a key component of the ggplot2 community, providing a friendly environment to ask questions about ggplot2 and graphical presentation. As of November 2010, the mailing list had nearly 1200 members, and around 7000 messages. It's difficult to know exactly who the members are (most people sign up with gmail accounts), but a 2010 survey of the ggplot2 mailing list revealed that users come from think tanks, non-profits (including the world health organisation), independent consultancies, government agencies, federal labs and companies, both big and small (including ebay, google and facebook), and are using ggplot2 to visualise data from ecology, ecophysiology, evolution, environmental science, toxicology, microbiology, pharmacology, medicine, epidemiology, global health, psychology, psychophysics, linguistics, energy, industrial engineering, technology, chemistry, physics, political science, business, finance, quality control, and market research.

Initially I invested a lot of effort to ensure the that all questions on the list were answered. Recently, the volume messages has become too high to sustain this effort, and the mailing list has become largely self-sustaining with most questions answered by other community members.

As well as the mailing list, ggplot2 also has a budding community on the programming question and answer site stackoverflow[3]. At the time of writing, over 250 questions had been asked and answered on the site. One compelling feature of the stackoverflow is the automatically generated frequently asked questions page[4] which algorithmically determines the most common questions.

Wiki

The chief feature of the ggplot2 wiki[5] is the annual case study competition. This competition provides a venue for ggplot2 users to show off their graphics, and highlights how the richness of the grammar makes it possible to create a very wide range of graphics. Figure 7 shows parts of three entries from the 2010 competition.

The wiki also includes a section where users can list publications where they've used ggplot2, appearances of ggplot2 around the web, links to frequently asked questions and some tips and tricks.

[3] `http://stackoverflow.com/tags/ggplot2`
[4] `http://stackoverflow.com/tags/ggplot2/faq`
[5] `https://github.com/hadley/ggplot2/wiki`

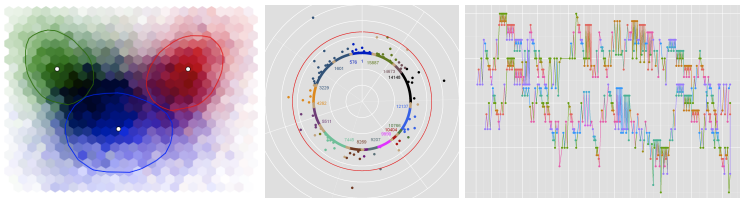

Figure 7: Graphics created for the ggplot2 case study competition for data on (left) the spectroscopic grading of gliomas, (centre) mitochondrial DNA, and (right) the music of Bach.

Code contributions

As the ggplot2 community grows, some users have begun to mature into developers and contribute code back to ggplot2. This is made possible, in part, by ggplot2's development process: everything occurs in the open using the code sharing site github, `http://github.com`. Github provides a number of services that make development easier including:

- Browsing the complete source, and history.
- Forking, which makes it easy for others to branch the main ggplot2 code and try out new things, and pull requests, which make it easy for me to merge these changes back into ggplot2 development.
- Line-by-line comments allow others to point out bugs that I've just introduced, and allow me to comment on other people's contributions.

Conclusions and future work

ggplot2 implements a rich grammar of graphics in R that makes it possible to create new types of plot, each specifically tailored for the problem at hand. ggplot2 is designed to support reproducible research, making it possible to program graphics making visualisation reproducible, automated and easy to communicate. This ease of communication as helped support the development of a strong community and rich ecosystem.

Currently, I am working on turning ggplot2 from the final product of my thesis research to fertile ground for future work. This involves breaking ggplot2 out into multiple simpler packages that each implement a core piece of the framework, like scales and layers, allowing the ggplot2 package to focus on the core grammar. Simpler packages will also make it easier for community members to transition from users to developers, allowing them to contribute new features or bug fixes in clearly defined areas. This work will also be useful for authors of other graphics packages, reducing the amount

of duplicated code and making it easier to explore new ways of describing graphics, without having to devote so much time to fiddly lower-level details.

This work will also provide infrastructure for the next generation of graphics in R, which will combine the best of interactive and dynamic graphics from GGobi with the power and flexibility of a grammar. This is joint work with Michael Lawrence, Heike Hofmann, Di Cook and the Iowa State statistical graphics research group. Initial exploratory work using a OpenGL based graphics device written by Michael Lawrence and Deepayan Sarkar has shown that it is possible to write performant interactive graphics in R.

References

Andreas Buja, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-Kyung Lee, Deborah F. Swayne, and Hadley Wickham. Statistical inference for exploratory data analysis and model diagnostics. *Royal Society Philosophical Transactions A*, 367(1906):4361–4383, 2009.

Paul Murrell. *R Graphics*. Chapman & Hall/CRC, 2005.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, 2008.

Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

Hadley Wickham, Dianne Cook, Heike Hofmann, and Andreas Buja. Graphical inference for infovis. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis'10)*, (In press). [26% acceptance rate. Best paper award.].

Leland Wilkinson. *The Grammar of Graphics*. Statistics and Computing. Springer, 2nd edition, 2005.