USING VISUALISATION TO TEACHING DATA ANALYSIS AND PROGRAMMING

Hadley Wickham
Rice University, United States of America
hadley@rice.edu

Modern data analysis demands computing skills that most potential statisticians lack. This paper discusses my approach to teaching data analysis and programming focused around the potential of visualization to engage students with the data and give them a flexible toolbox with which to attack many potential problems.

INTRODUCTION

This paper discusses my approach to teaching data analysis and programming with visualisation at the centre of the experience. My approach has been shaped by my experiences teaching four statistical computing and graphics classes, two at Iowa State University as a PhD student (http://had.co.nz/stat480) and two at Rice University as a new faculty member (http://had.co.nz/stat405).

My students are heterogenous; a mix of upper-level undergraduates, graduate students in statistics, and graduate students from other fields. Class sizes have ranged between 10 and 30, and has always been taught in a computer lab. Students' computational and mathematical skills are hugely variable, as are their previous experiences with data. The computing environment is also heterogenous. Most students use the lab computers (windows at Iowa State and linux at Rice), but a number work on their own laptops, which are typically windows or mac.

The aim of the course has always been to teach students how to analyse data and how to think computationally. These are both critical skills for the applied statistician, and are not covered in depth elsewhere in the curriculum. Statistical thinking is deeply woven into this course, but I do not explicitly teach specific statistical methods. This was a deliberate choice to allow me to focus on low-level tools that are useful for many (if not all) data analyses. However, I do encourage students to use statistical techniques they have learned in other classes and will provide feedback on whether or not they have been used appropriately.

The remainder of this paper is laid out around the challenges and opportunities of such a class: teaching data analysis, teaching programming and the integration of my teaching and research. Data analysis is a high-order creative skill and is tricky to teach. It requires the mastery of tried and true techniques as well as the ability to synthesise new variations to address the problem at hand. Data analysis is a craft, a combination of science and art, and can not be taught with the same techniques we use for more pure mathematical topics. In Section 2 I discuss how data analysis was integrated into the course, and how I attempted to build strong data analytic skills in my students.

In the most recent versions of the class, I have chosen to only use open source software: R for statistical computing and latex for homeworks and projects. This is ambitious: many students are intimidated by programming, and few are comfortable with text-based command-line-oriented software. Section 3 discusses why the command-line (and computational thinking in general) is so important, and summarises the strategies that I use to help students become productive programmers.

For me, teaching statistical computing has been a fruitful source of research ideas: if a topic is difficult to teach, the implementation or underlying theory may be inadequate. In Section 4 I discuss how addressing these inadequacies has been useful for my research program. I conclude with a summary of my experiences teaching this course, and what I plan to change next time, in Section 5.

DATA ANALYSIS

Data analysis is a hard skill to teach because there is no simple recipe to follow. One can point out the broad brush strokes of an analysis (explore visually to gain the gestalt of the data, create a quantitative model that summarises the key features, then write up in a way that makes the sequence from data to conclusions sensible and obvious), but every dataset requires a slightly

different approach. My technique for teaching data analysis is to provide many opportunities for the students to do data analysis and then provide copious feedback on their efforts. Assessment is particularly important, and in Section 2.1 I discuss how I use assessment to steer students towards better analyses.

In my class, I focus more on visual exploration and less on quantitative modelling. I expect students to work mainly with the raw data and produce graphical summaries. I am interested in the gestalt of the data, not p-values or hypothesis tests or accurate predictions; these can come later and in other classes. Section 2.2 briefly my approach to statistical graphics, based around the layered grammar of graphics.

Assessment

Data analysis skills are evaluated and improved with weekly homeworks and three larger data analysis project. The first few homeworks focus mainly on data analysis, but as the major projects come online the focus moves towards towards practicing programming skills. This last year the class culminated in a poster presentation which was attended by many people outside the class.

I grade data analysis homeworks (and a large component of the group projects) with a rubric of three components: curiosity, scepticism and organisation. These reflect what I believe to be the three key attributes of a statistician: they should be curious about data and able to creatively apply old tools in new ways; they should be sceptical about their findings, always aware that a result may be the result of chance alone, and on the look out for ways to double check their work; and they should be able to present their findings in an organised manner that guides the audience from raw data to results. A copy of the complete rubric is available at the end of the paper.

Teaching statistical graphics

My approach to teaching statistical graphics is based around my research work integrating the grammar of graphics (Wilkinson, 2006) with R. A strong theory of graphics is very useful for teaching because students are not limited to a small palette of named graphics, but can create new visualisations as appropriate for their data.

I teach statistical graphics in the following order:

1. The basics: the scatterplot and histogram. Students are already familiar with these and just need to learn how to create them in R. I also revise reading these plots and emphasise the importance of experimenting with the bin width of the histogram.
2. Aesthetics and facetting. The histogram shows one variable and the scatterplot shows two. What do you do if you want to display more? There are two choices: map additional variables to other perceptual properties (like colour or size or shape), or display small multiples conditioned on another variable.
3. Time and space. Temporal and spatial data is very common and requires new plot types: line plots, choropleth maps and proportional symbol maps.
4. Polishing for presentation. Scales control the mapping from data to things we can perceive and are crucial for turning an exploratory plot in to a plot suitable for communication.
5. Theory for analysis and critique. Finally, I teach the students the complete theory. It is unusual to teach theory last, but I find it works best, because the students have seen how useful the pieces are and are motivated to integrate them into a unified whole.

PROGRAMMING

Why teach programming?

Learning how to program is an important skill for every analyst. While convenient, using a graphical user interface (GUI) is ultimately limiting and hampers reproducibility, communication, and automation:

- *Reproducibility.* If a data analysis is to be a convincing scientific artefact, the trail from raw data to final output must be available. It is very difficult to do this with a GUI, and it is easy for mistakes to creep in (for example, accidentally sorting just a column of data in Excel, not the whole table).

- *Communication.* Code is a vehicle for communication, not just to the computer, but to yourself in the future, and to other professionals in your area. It is difficult to communicate how to use a GUI: click here, then right click here and then choose menu X... Code is easy to communicate because all important information has a text based representation. This makes it trivial to supply code to reproduce a particular problem or solution when teaching.
  Versions of this course taught at Iowa State (Stat480) also used Excel and SAS. Students commented favourably on the use of R and SAS: they could see absolutely everything I was doing and replay it after class, if necessary. When using Excel, it was difficult to tell exactly where I was clicking (and one pixel can make a big difference). Making replays available was time consuming, and required video recordings (I explored doing this but never actually did it because of time constraints.)
- *Automation.* If you've performed an analysis with a GUI, it is difficult to recreate it for new dataset. This happens often in practice, since data are rarely final - during the process of data preparation and exploration you are likely to find problems that can be fixed with reference to the original data. Rerunning a script with a new dataset is trivial.

Some ideas on how to teach programming

Teaching programming is important, but hard. Many students have never programmed before and are intimidated by the command line. I think many computing classes make a fundamental mistake when teaching these students: they start with the basics, the formal structure of the computing language, and the low-level primitives that everything else is built on. My first time teaching I followed this approach, but it took six weeks of basics before students could accomplish anything of interest. This made it hard to keep motivated and on-task. The next time I taught the class, I started with something interesting and useful: graphics.

Now the first day of class teaches students how to open R and create basic graphics with ggplot2. They may have never used a programming language before and don't know anything about how R works, but this doesn't hold them back. They start by using the code I provide as a template, not really understanding what it does, just blindly changing variable names to get different views of the data. As students do this, they start to learn some important things about computer programming: the computer is very fussy and you need to make sure you've typed in everything just right.

I choose to start with graphics because they are visually engaging and can be used to gain insight into any dataset. By the end of the first week students are equipped with the basic tools of statistical graphics and can compare different subsets using conditioning and aesthetics. To get them started with data analysis, the first homework is simple: find three interesting views of a dataset. During class I stress the importance of iteration: the first plot will never be the most revealing and so you need to think of each plot as a single step towards enlightenment.

As the class progresses, I support the transition from blind use of templates to a deeper understanding of the theory that underlies R. I teach how to write functions (and when they are appropriate) and theories of data analysis and visualization. In conjunction with the larger data analysis projects this encourages students to assemble the components that they have learned in class in new and creative ways.

Assessment

As with data analysis, rapid feedback is essential for learning good programming skills. Assessing code is difficult, and I am still struggling to develop good grading criteria; to paraphrase Justice Potter Stewart, I may not be able to describe good code, but I certainly know it when I see it. This is not great for a pedagogical standpoint and I continue to struggle with the best way to grade code to keep students headed towards better quality.

Currently, my assessment centers around on the notion of code as communication, and I assess it on three criteria: planning, execution and clarity. Planning grades evidence of thought before writing the code. Is there a clear strategy, described by an introductory comment? Does the breakdown of the large problem into smaller sub-problems make the problem easier? Execution grades mastery of R vocabulary and use of functions - ideally the code should be concise and free of duplication. Clarity grades how easy it is to read and understand the code.

Coupled with these high level objectives are penalties for poor style. Students need to learn the stylistic conventions for writing code, much as correct punctuation is a necessary skill for the written communication in English. Here points are deducted for errors like incorrect spacing and indenting and overly long lines. This makes the code much easier to read (and thus grade) and helps to establish a common style amongst students so that collaboration in group projects is easier. A copy of the complete rubric is included at the end of the paper.

In some homeworks I focus on lower-level skills. A certainly fluency in the basics (data manipulation, writing functions, identifying errors) is necessary before they can be fluidly combined to solve bigger problems. To practice these skills I assigned programming drills, made up of many simple problems. Each problem only requires a few minutes of thought, and stringing many together helps practice common techniques so that they can be quickly retrieved from memory. These drills are graded based on correctness with the assumption that most students will achieve grades of 90\%+.

RESEARCH

Teaching with data analysis and programming has also been valuable for my own research. Often, parts of R are difficult to teach because they seem to consist of a huge number of special cases; there is no underlying structure which provides a scaffolding for learning. I have found it profitable to explore these areas in more depth, investigating whether there are better ways of solving the same problem in other programming languages, or whether there is an opportunity to develop new theory.

I have developed better tools for text (the stringr package) and dates (the lubridate package, with Garret Grolemund) data, by adapting libraries from other programming languages. This is not novel research, but is a useful service to the community. Other problems have lead to the development of new theory and associated packages: ggplot2 for graphics; plyr for problems where you split up a large, complex data structure, process each piece and recombine; and reshape, for data reshaping (Wickham, 2007).

CONCLUSION

When I first taught this course I was surprised that students who had taken many statistics classes didn't have the first clue how to actually perform a data analysis. As I have revised this course I have focused on data analysis (rather that statistical computing or programming) as the key theme. I have found teaching programming by starting off with visualisation to be very successful, and I would strongly encourage others to try it out too.

ACKNOWLEDGEMENTS

I would like to thank Deborah Nolan, Roger Peng, Duncan Temple Lang, Dianne Cook, Andreas Buja, Heike Hofmann, Luke Tierney and many others for the many conversations that have shaped my understanding of statistical computing and how to teach it. The "reinventing the statistical computing curriculum" workshops have also been invaluable. Finally, I'd like to thank my TA, Garrett Grolemund, who came up with many of the problems used in the drills.

REFERENCES

Wickham, H (2007). Reshaping data with the reshape package. *Journal of Statistical Software, 21*(12). Online: http://www.jstatsoft.org/v21/i12/paper

Wickham, H. (In press). A layered grammar of graphics. *Journal of Computational and Graphical Statistics.*

Wickham. H. (2009). *ggplot2: Elegant graphics for data analysis. useR.* Springer, July 2009.

Wilkinson, L. (2005). *The Grammar of Graphics*. Springer.

RUBRICS

The following page includes my rubrics for grading data analysis homeworks and code.

	Outstanding (A+)	Good (A)	Acceptable (B)	Needs work (C)	Inadequate (F)
	5	4	3	2	1
Curiosity	Intense exploration and evidence of many trials and failures. You have looked at the data in many different ways before coming to your final answer. You have gone beyond what was asked: additional research from other sources used to help understand/explain findings. Your explanation and presentation is creative	Plenty of exploration and investigation. Some additional research helps explain findings, and some of your ideas are creatively presented and explained.	Some exploration, but little evidence that you have selected the best of many ideas. Little or no additional research.	You have done the bare minimum that was asked. There is no evidence to suggest that you tried multiple approaches (tables, graphics, or models) before coming to your final conclusion.	Questions are simple, and there is no evidence of exploration. You have not come up with your own questions of the data, but relied on those we discussed in class
Scepticism	You suggest multiple explanations for a given finding, and use multiple tools to explore surprising results. You present one or two as the most plausible, but have allowed for the possibility that you are wrong. You are self-critical: What did I do well? What did I do poorly? What have I missed? How could I do better next time? You identify flaws in methodology and provide suggestions as to how they could be remedied You don't blindly accept perceived wisdom, but challenge preconceived notions and come up with interesting new ways of testing them.	You are sceptical and self-critical, but not consistently. There is some critical analysis, and some use of multiple techniques to answer the same question.	You haven't blinded accepted findings, but you haven't come up with many ways to check your results either. There is little self-criticism and little evidence to suggest you have thought about how to do better in the future.	Some findings accepted without question. Self-criticism weak.	Findings accepted uncritically. Leaps of logic without justification. You have not thought about how to do better next time.
Organisation	Findings very well organised. Clear headings demarcate separate sections. Excellent flow from one section to the next. The paper is easy to scan. An abstract or summary at the start of the paper briefly summarises your approach and findings. Conclusions at the end present further questions and ways to investigate more. Tables and graphics carefully tuned and placed for desired purpose.	Findings well organised and sections clearly separated, but flow is lacking. Each section has clear purpose. Tables and graphics clear and well chosen	Generally well organised, but some sections muddled. Tables or graphics appropriate, but some are poorly presented - too many decimal places, poorly chosen aspect ratio etc.	Sections unclear and no attempt to flow from one topic to the next. Graphics and tables poorly chosen to support questions. Some have fundamental flaw.s	It is hard to read your paper. There are no headings, figures are far away from where they are referenced in the text. There is no summary or conclusion.

	Outstanding (A+)	Good (A)	Acceptable (A-/B+)	Needs work (B/C)	Inadequate (F)
	5	4	3	2	1
Planning	Introductory comment describes overall strategy and gives evidence of preliminary planning. Thoughtful problem decomposition breaks the problem into independent pieces that can be solved easily.	Evidence of planning before coding, but some flaws in overall strategy.	More planning needed: overall strategy ok, but have missed some obvious ways of making the code simpler.	It all hangs together, but planning was absent or rushed.	No evidence of planning. Strategy deeply flawed.
Execution	Mastery of R vocabulary means that the absolute minimum amount of code is used to get the job done. Code free from duplication. Each function encapsulates a single task, and repeated tasks are performed by functions, not copy and paste	Workable, but not elegant. Common programming idioms used to reduce code. For example: character subsetting instead of complicated if statements; vectorised functions instead of for-loops.			Functions used inappropriately, or existing functions reinvented. Extensive use of copy and paste.
Clarity	Code is a pleasure to read, and easy to understand. Code and comments form part of a seamless whole.	Comments used to discuss the why, and not how of code; to provide insight into complicated algorithms; and to indicate purpose of function (if not obvious from its name). Comment headings used to separate important sections of the code. Variables, functions and arguments named concisely but descriptively.	Generally easy to read, but some comments used inappropriately: either too many, or too few. Some variable names confusing.	Hard to understand. Poor choice of names and comments do not generally aid understanding.	Cannot understand code. If it works, I have no idea why.

One point will be deducted for each of the following style guide violations:
file naming, identifier naming, spacing, curly braces, indentation, line length, assignment