Now You Know How To Go On

Andrew Goldstone

June 12, 2015

> Think how we learn to use the expressions "Now I know how to go on," "Now I can go on" and others; in what family of language-games we learn their use.
> Ludwig Wittgenstein, *Philosophical Investigations*, translated by G.E.M. Anscombe, pt. 1, §179

The course is over, but I thought it important to add some notes about continuing the work you have begun. The main goal of the course was that you would know how to go on—both technically and theoretically—when it comes to analyzing literary data. You have shown, in your reports, your ability to manipulate, visualize, and explore a wide variety of literary data sources analytically. But you have also shown your capacity for figuring out new methods. So here are some suggestions for possible directions in which you might choose to go on—with confidence. I've included bibliographic notes at the end of each section.

MORE WITH R

PRESS ALL THE BUTTONS

Though we covered enough of R that you could in principle do pretty much anything with the language, we certainly didn't discuss everything the language can do. But you now have a framework for building more skills. If you find references to some computational technique that seems interesting to you, see if you can find an R built-in function or a package on CRAN (or elsewhere) that implements the technique, and see whether you can use it. (Follow example code, then if that works, see if you can modify it . . .) You are allowed to just Google things, too! Or to ask questions on Stack Overflow for that matter. The key is to try things out prolifically and to be on the lookout for things to try; strive to be blithely unworried by dead ends and temporary failures.

If you want to meta-practice trying things out, here are some suggestions:

1. Jockers, chap. 11: hierarchical clustering. Most of the chapter is data-wrangling code, all of which could be improved (by you). But at the very end Jockers demonstrates cluster-

ing with R's `hclust` function and visualizing a clustering result as a dendrogram (tree plot). Read `help(hclust)` and try the various clustering methods, not just the default ("complete-linkage" clustering), either on the data Jockers sets up for you or on any other data of similar form (what you need is: a bunch of texts that you think *could* be hierarchically classified; featurize them and then compute a distance matrix).

2. Community detection in networks. There are algorithms to divide up the nodes of a network into relatively strongly-connected groups (this is like finding cliques, but fuzzier and harder). `igraph` implements a pile of community-detection algorithms. Figure out what they are and see if you can get one or three working. `igraph` has a nice built-in function for plotting communities, but see if you can get beyond the visual and probe the detected communities programmatically.

3. The Naïve Bayes classifier. This widely used and relatively simple algorithm starts from a set of documents that are assigned to a small number of classes, and then uses the features of those documents to guess the class membership of another document or documents. Searching around you can find a few tutorials on trying out the Naïve Bayes classifier functions available in two CRAN packages, `e1071` and `klaR`. But a more interesting challenging would be to read Hulden, "Supervised Classification," on the Programming Historian, skimming over all the parts that are in Python. Python, schmython! See if you can replicate her results in R, either using `klaR::NaiveBayes` or even from scratch using only basic mathematical functions (you only need sums and logs . . . and `dplyr` pipelines).

4. Distinctive words. Dunning, "Accurate Methods for the Statistics of Surprise and Coincidence," is a quite accessible essay about the problem of finding which words "stand out" most in a comparison between two documents. Dunning introduces his "log-likelihood" statistic, which is often superior to something like tf*idf scoring for picking out distinctive words. With a little diligent Googling you could find someone else's implementation, but Dunning gives you a formula which you could implement yourself (again, no magic—just sums and logs over vectors).

5. Make a thematic map. I am quite clueless when it comes to geography, but here's how I'd start exploring (rehashing some links from my slides on the last day). Try out the examples from the `tmap` package (getting it installed will take patience) and then see if you can extend them into literary geography using, say, the Three Percent data. You may also need to work out how to "geocode" locations using the `ggmap` package.

Dunning, Ted. "Accurate Methods for the Statistics of Surprise and Coincidence." *Computational Linguistics* 19, no. 1 (March 1993): 61–74.

Hulden, Vilja. "Supervised Classification: The Naive Bayesian Returns to the Old Bailey." In *The Programming Historian*. 2014. http://programminghistorian.org/lessons/naive-bayesian.

Jockers, Matthew L. *Text Analysis with R for Students of Literature*. New York: Springer, 2014. doi:10.1007/978-3-319-03164-4. The parts we didn't work through, that is.

Kolaczyk, Eric D., and Gábor Csárdi. *Statistical Analysis of Network Data with R*. New York: Springer, 2014.

Tennekes, Martijn. "tmap in a Nutshell." 2015. http://cran.r-project.org/web/packages/tmap/vignettes/tmap-nutshell.html. A few other mapping possibilities are noted in the slides from our last class meeting.

MORE ADVANCED PROGRAMMING

We passed over R's more complex data structures, which come in the form of *objects*; R is an *object-oriented* language, which means it provides tools for specifying complex data models and writing functions that encapsulate computations that are particular to them. You already have seen, for example, that `print` in

```
x <- "Loving in truth, and fain in verse my love to show"
print(x)
```

does something very different from `print` in

```
x <- ggplot(laureates, aes(gender)) + geom_bar()
print(x)
```

That is because R knows about many versions of the `print` function, and it knows how to hand control over to the one corresponding to the type of `print`'s argument. (`print` is a so-called "S3 generic function.") For discussions of R's object-oriented programming features, see Matloff, *The Art of R Programming*, or Wickham, *Advanced R*. Object-oriented programming is more confusing in R than in almost any other programming language, however; a better introduction to OOP would be via Python.

We also didn't say much about the challenges of determining whether your programs are *efficient* in their use of time and storage space—and if they aren't, improving them. This is one of the deep subjects of computer science. However, for the R-specific craft of improving slow programs, two texts are of interest: Wickham, *Advanced R* (again), and the grimly entertaining Burns, *The R Inferno*.

R is primarily designed for statistical analysis, but we barely touched on its capacities in this domain or on what the domain *is*. Statistical computing means writing programs that help you understand data. If that sounds awfully like the subject we were purportedly studying all semester—it is! But we didn't get to pursue the more quantitative and formalized methods that are the

stock in trade of social science, which include numerical descriptions, assessments of the validity of inferences about the data, predictions from data, and classification techniques. Computers are good not only for doing the numerical calculations of statistics but for *simulating* the processes that might generate data in order to clarify those processes. R has many specialized tools that help with all of this.

Taking what you know about programming as a starting point, there are two routes into this aspect of computation, depending on whether you treat the "statistical" or the "computation" as primary. For the computational emphasis, a reasonable source is Matloff, *The Art of R Programming*, which explains some sophisticated R programming with a focus on statistical applications. The statistical route, which may be preferable because it leads to more generalizable methods, would be to begin learning statistics in a context where R is the computing environment of choice. You might work through an introductory statistics text or look for a course featuring R. As for elementary introductions to statistics in book form, the options are overwhelmingly many, and I am unqualified to venture a recommendation. I did find Navarro's draft textbook (*Learning Statistics with R*) well-written, with a good review of R fundamentals. Gries—whose chapter on regular expressions from his corpus-linguistics text we read—has a textbook on statistics for linguists using R. Also of interest is James et al., *An Introduction to Statistical Learning*, which tries to use R to get you from first principles to "modern" data analysis methods, hand-waving past all the advanced mathematics.

You can of course study statistical computing in any programming language or software environment, not just R.

Burns, Patrick. *The R Inferno*. 2011. http://www.burns-stat.com/pages/Tutor/R_inferno.pdf. The author's "Abstract" reads: "If you are using R and you think you're in hell, this is a map for you." An amusing guide to programming pitfalls in advanced R usage.

Gries, Stefan Thomas. *Quantitative Corpus Linguistics with R: A Practical Introduction*. New York: Routledge, 2009.

———. *Statistics for Linguistics with R: A Practical Introduction*. 2nd ed. De Gruyter, 2013.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. New York: Springer, 2013. doi:10.1007/978-1-4614-7138-7. An elementary version of a famous machine-learning textbook (Hastie and Tibshirani). Probably exhausting to work through in its entirety, especially since the examples are pretty distant from literary and cultural study, but useful as a handbook, with R code, for machine learning techniques that are in wide use.

Matloff, Norman S. *The Art of R Programming: A Tour of Statistical Software Design*. San Francisco: No Starch Press, 2011. A widely-used R book that goes from basics to advanced topics, including things like object-oriented programming and debugging. It has some good extended example programs, focusing on statistical applications in the sciences.

Navarro, Dan. *Learning Statistics with R*. http://health.adelaide.edu.au/psychology/ccs/teaching/lsr/. I liked this introductory statistics textbook, aimed at psychology undergrads. It has a good review of R basics an accessible elementary presentation of basic descriptive and inferential statics (no calculus or linear algebra). It is a work in progress, but the author shares the manuscript online.

Wickham, Hadley. *Advanced R*. Chapman & Hall, 2014. http://adv-r.had.co.nz/. When he says "Advanced," he means it. You know enough to read this book, but there is challenging material here, elliptically presented.

THE PROGRAMMER'S TOOLCHAIN

The course stayed (mostly) inside R and RStudio in order to reduce the number of things you had to cope with. But in fact if you want to do anything further that requires writing your own programs or writing "code" of any kind, it would be worth developing some facility with more tools of the code-writing trade.

THE SHELL

We glimpsed the shell a few times; this is the command line, from which you can navigate your file system, manipulate files, and run programs. It is similar in spirit to the R console but in fact operates in a language of its own (the language of *shell scripting*).

It's quite useful to acquire some facility with this environment, for three reasons: (1) some very handy programs have no graphical interface and must be operated from the command line; (2) you can learn to automate some of the repetitive tasks that otherwise require lots of clicking and dragging; and (3) you can do things when you are interacting with server computers that are command-line-access only (very common in web development). Unfortunately, the terminal is often a frustrating and obscure place, full of what the computer scientist Philip Guo calls "command-line bullshittery,"[1] the "incidental complexity" that gets in the way of doing the things you care to do. Nonetheless, it's worth learning your way around the shell.

When I say "the shell," I mean one of the standard shells in a Unix, Linux, or MacOS X system: the most commonly used are bash, tcsh, and zsh. If you have a Mac and you open the Terminal, by default you will find yourself using bash. If you want to practice your shell skills on a Windows computer, install the venerable Cygwin.[2]

In addition to manipulating files, you can do a quite a bit of elementary text-processing from the shell, including regular-expression searching and replacing (using `grep` and `sed`), sorting and deduplicating (`sort` and `uniq`), and so on.

1. http://pgbovine.net/command-line-bullshittery.htm

2. http://www.cygwin.com. The Windows Command Window is in the class of shell-type programs but it is not a *nix shell; neither is the Windows PowerShell, though it can do lots of things. If you set up the Vagrant VM you can also access a Linux shell by logging in to your virtual server: try `vagrant up`; `vagrant ssh`. Now you are using bash on a virtual Ubuntu machine.

Once you set out to use command-line programs you quickly discover that you have to install *other* programs to get things working. This is where the bullshittery really kicks in. You'll need to use "package managers" to install software; on the Mac, one uses homebrew[3]; on Debian and Ubuntu (the most common Linux distributions), there's `apt-get` and `aptitude` (see the Ubuntu help pages on package management[4]).

I don't have particular book recommendations on this front, since the books I learned from are all years out of date. Among numberless online tutorials, you could refer to Milligan and Baker, "Introduction to the Bash Command Line."

Milligan, Ian, and James Baker. "Introduction to the Bash Command Line." In *The Programming Historian*. 2014. http://programminghistorian.org/lessons/intro-to-bash.

VERSION CONTROL

As you have learned, as soon as you have a programming task of even moderate complexity—or as soon as you have collaborators—keeping track of the different versions of your programs and reconciling versions can be a challenge. Version control software helps with this task: think of it as an amped-up "Track Changes" feature with elaborate capacities for working with multiple collaborators and reconciling conflicting versions. There are quite a few version control systems, but the system of the hour is *git*, especially in conjunction with the massive open-source repository and "social coding" website Github.[5] It takes a little practice to learn to use git for keeping track of your own work; you can do this not just for programs but for any plain text. For years now I have kept all my notes and drafts, as well as all my programs, in git repositories: a repository stores the *editorial history* of every file, and it even lets you work with multiple alternate versions of your files. When you're writing a program, it's very nice to keep track of when you had your last working version and be able to "rewind" backwards if things go awry. When you're collaborating, the ability to compare and merge divergent editorial histories is indispensable.

Chacon, Scott, and Ben Straub. *Pro Git*. 2nd ed. Apress, 2014. http://git-scm.com/book/en/v2. An extensive tutorial book on the version control system, freely available online.

Code School. "Try Git." http:/try.github.io. If you can hold your nose at the smell of "gamification," this is a friendly introductory tutorial to git and github.

TEXT EDITING

Learn one or more programmer's text editors. These are free-standing editors like the editor built into RStudio but are more general-purpose. It is traditional to cite the Thirty Years' War between

3. http://brew.sh
4. https://help.ubuntu.com//lts/serverguide/package-management.html
5. http://github.com

Vim and Emacs. I use Vim. Neither is easy to learn, though both have good manuals, if you have the patience. On Macs I think the free TextWrangler is a good basic choice; the equivalent on Windows might be Notepad++. Sublime Text is a newer program that has many adherents.

A SECOND LANGUAGE

Learning more programming languages is usually much easier than learning the first one and more straightforward to accomplish on your own. For further applications in text-processing and data analysis, the obvious next choice is Python. Python is more general-purpose than R, and indeed is used in some introductory computer science courses (R is not). Like R, Python has a rich "ecosystem" of software libraries that enhance its capabilities. Some things that R makes tricky, Python makes much simpler, like processing huge files a small piece at a time.

As experienced programmers, you can indeed learn Python from the official Python tutorial.[6] Almost everything will seem familiar from R, yet slightly different. You might also find the Programming Historian's sequence of Python lessons congenial (scroll down to the bottom of the Lessons page for the list of Python lessons). You should install and learn Python 2, not Python 3. On a Mac, use homebrew (`brew install python`) to get an up-to-date version of the software; MacOS X comes with Python, but for various reasons it's a good idea to install a separate copy for your own work.

For algorithmic language processing, the book explaining the widely-used Natural Language Toolkit for python (Bird, Klein, and Loper, *Natural Language Processing with Python*) is a starting point.

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. 1st ed. Sebastopol, CA: O'Reilly, 2009. http://www.nltk.org/book_1ed/. Introduces natural-language-processing fundamentals through the widely-used NLTK library for Python. The full text is online; the website promises that an updated edition of the book is forthcoming.

Guttag, John. *Introduction to Computation and Programming Using Python*. Rev. ed. Cambridge: MIT Press, 2013. A very accessible elementary computer science book that teaches you Python at the same time. I almost used it for our course.

Lutz, Mark. *Learning Python*. 5th ed. Sebastopol, CA: O'Reilly, 2013. A big introductory book on the Python language. I haven't used this one myself.

The Programming Historian. 2013–present. http://programminghistorian.org/. Tutorial lessons by various hands, aimed at historians. There is a sequence of programming lessons in Python. There are also lessons about various web platforms, including two very clear lessons on Omeka by Miriam Posner.

6. https://docs.python.org/2.7/tutorial/index.html

WEB DEVELOPMENT

We gave only the briefest glimpse to HTML and "coding" for the web, but web development is central to what everyone means by DH. Even within the narrower precincts of quantitative methods for literary history, the web has been very important both as a medium of scholarly communication and as a source of data.

Given what we have done so far, the fastest route to learning about making web sites might be to learn about doing interactive data visualization online. There is a very fine introductory book, Murray's *Interactive Data Visualization for the Web*, which gives you a tutorial in using the ubiquitous D3.js (Data-Driven Documents in JavaScript) library. This book will teach you the basics of HTML, CSS, *and* the JavaScript programming language on the way to showing you how to put interactive charts derived from data in CSV, TSV, or similar plain-text formats online. JavaScript is an important language because it is so ubiquitous; if you have a web browser, you have the software you need to run JavaScript programs. It is akin to both R and Python, and again knowing R will help you pick it up on your own if you're motivated to.[7]

But of course you might have other Web-ly interests than data visualization. Indeed, I may have been remiss in my duty to you as a DH teacher by not starting out the class by having you make your own website (a common introductory activity in these courses). Well, I felt that by the end of the course you would be ready to figure out how to do that for yourself. If you'd like to make a site, I recommend one of the following options:

1. activate your MLA Commons[8] Wordpress site
2. deploy a static site on Github Pages[9]
3. deploy a static site on Amazon S3, buy a domain name, and point it there
4. pay for an ad-free hosted Wordpress site (try Reclaim Hosting[10] or wordpress.com[11])

More complex web-based endeavors require learning about database software and the programs that make it possible for websites to access them. Often database, web-software, and web design are wrapped up together in a *framework* or *content-management system* (CMS). Wordpress itself is such a CMS. Another widely used platform in DH circles is Omeka, which makes it relatively simple to create a database of digital objects and "exhibit" it. Then there are general-purpose CMSs; popular examples include Drupal, Ruby on Rails, and Django. These use the PHP, Ruby,

7. There's also an R package for making interactive visualizations for the web, called Shiny (http://shiny.rstudio.com). Its major disadvantage is that you either need to run specialized server software or to use RStudio, inc.'s cloud service, http://www.shinyapps.io (the basic service has no monthly fee). I haven't experimented with it, but if you'd like to stay with R it might be interesting to try out.
8. http://commons.mla.org/
9. https://pages.github.com/
10. http://reclaimhosting.com/
11. http://wordpress.com

and Python programming languages, respectively. I do not advise setting out to learn any of these just because; but it's good to know that if you ever wanted to make a complex, interactive web project, you wouldn't have to invent everything from scratch.

"DH@RU Workshops." 2013–present. http://dh.rutgers.edu/workshops/. Online notes for all the DH workshops held at Rutgers, including one on getting yourself online.

Murray, Scott. *Interactive Data Visualization for the Web*. Sebastopol, CA: O'Reilly, 2013. http://alignedleft.com/work/d3-book/. There is a free online version of this tutorial book.

THEORY AND METHODOLOGY

Finally, here are some further theoretical jumping-off points for the study of literary data that I wish I'd had space for on the syllabus:

Bennett, Tony, Michael Emmison, and John Frow. *Accounting for Tastes: Australian Everyday Cultures*. Cambridge University Press, 1999. A fascinating collaboration between two literary scholars in the Birmingham School tradition and a statistician attempting to assess cultural hierarchy and diversity in Australia.

Boltanski, Luc, and Eve Chiapello. *The New Spirit of Capitalism*. Translated by Gregory Elliott. London: Verso, 2005. If you want to understand DH as an organizational trend, you could do worse than to read this book. In some ways this picks up on the Raymond Williams story and carries the analysis into the post-Fordist epoch. Does some interesting quantitative textual analysis, too.

Bourdieu, Pierre. *Distinction: A Social Critique of the Judgement of Taste*. Translated by Richard Nice. Harvard University Press, 1984. Still a *sine qua non* for quantifying and assessing cultural objects, to be read alongside *The Rules of Art*. Despite many criticisms, not least of the data analysis, this book has remained perhaps the most important touchstone for sociologists studying culture.

English, James F. *The Economy of Prestige: Prizes, Awards, and the Circulation of Cultural Value*. Cambridge: Harvard University Press, 2005. A qualitative study, yet understanding cultural value and its distinctive economies is fundamental to any serious literary data analysis. Considering prizes and the organizations that give them allows us to think of literature *as* data in novel ways.

Frow, John. "On Midlevel Concepts." *New Literary History* 41, no. 2 (Spring 2010): 237–52. A very fine essay on the present state of the sociology of literature, part of an interesting *NLH* special issue.

Griswold, Wendy. *Regionalism and the Reading Class*. Chicago: University of Chicago Press, 2008. This is the most recent book by the leading American sociologist of literature. Remarkable as a cross-nationally comparative study of readers and the state institutions that contribute to cultures of reading.

Leavis, Q. D. *Fiction and the Reading Public*. 1932. London: Chatto and Windus, 1965. A road not taken in literary studies: Leavis was and is singular for her atttention to actual readers and not simply texts, and for her willingness to carry out survey research! About the only thing that was carried forward was her pained awareness of cultural hierarchies.

Martin, John Levi. "What Do Animals Do All Day?: The Division of Labor, Class Bodies, and Totemic Thinking in the Popular Imagination." *Poetics* 27, no. 2–3 (March 2000): 195–231. A truly elaborate quantitative analysis of a single children's book by Richard Scarry. This is how the pros do it. Martin is a sociological theorist.

Radway, Janice A. *Reading the Romance*. Chapel Hill: University of North Carolina Press, 1984. Perhaps the greatest-ever work of literary data analysis: a revolutionary account of what readers *do* with their books.

Thompson, John B. *Merchants of Culture: The Publishing Business in the Twenty-First Century*. 2nd ed. New York: Penguin, 2012. An extraordinary analysis of trade publishing as a field, mostly ethnographic but contextualized by market data and a hundred years of history.

Tuchman, Gaye, and Nina E. Fortin. *Edging Women Out: Victorian Novelists, Publishers, and Social Change*. New Haven: Yale University Press, 1989. A feminist classic in the sociology of literature, and rare for its focus on historical rather than contemporary practice.

Williams, Raymond. *Culture and Society, 1780–1950*. New York: Columbia University Press, 1958. Almost all controversies about literary data are subtended by the developments Williams described in this indispensable book.

More media-studies and book history texts could and should be added here.