**Step 5**. Expand your horizons. If you do this (“this” writ large, meaning anything you might need programming skills for) long enough, someday you’ll probably run into something that would be a whole lot easier in a different programming language. And you’ll have a choice to make. You can A) suffer through doing it the hard way (if possible) using the language you already know; B) collaborate with someone who knows the programming language that can do it better; C) learn a little of the other programming language, scaffolding from your knowledge of the language you know better.

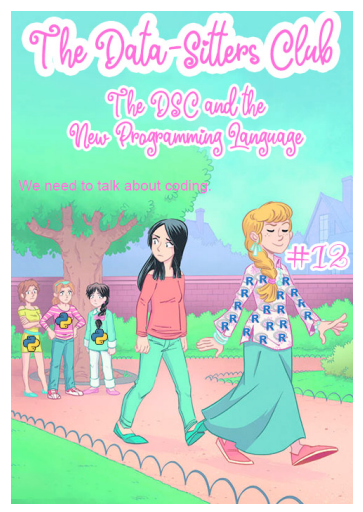
Me, I’m in step 5. It’s been three years since I started learning Python for real, and I’d rank my Python proficiency as a 3-4 (depending on the day) if I were to take my own DH coding survey. But I’m also part of the Stanford Literary Lab, and our Associate Data-Sitter Mark Algee-Hewitt is an R person. So if I want to build on some of the other Lab projects (like we did in DSC #10: Heather Likes PCA), I need to be able to work with R, even if I’d never choose to write it from scratch. And sometimes there are R packages that are worth us trying to understand – like Syuzhet in DSC #11: Katia and the Sentiment Snobs – and I need to be able to do that.

I’ve still got a long way to go. But just a couple chapters of an R textbook have helped a lot in dispelling my confusion about R’s syntax – and I was grateful to learn from the survey that I’m not the only one who struggles with that. I don’t know how much more I’m going to invest in R; I might be like one of those scholars content with a reading knowledge of German or French, with no particular need or desire to ever write or speak anything in those languages.

And that’s totally okay. DH coding skills aren’t like Pokémon: you don’t have to catch ‘em all. You don’t even have to catch any of them, to be honest. As for me, I’ve chosen Python as my Pikachu. Maybe someday R can be Togepi . And maybe, like in DSC #11, they can even work together sometimes. But it’s going to take a lot more work (and maybe some gritting my teeth) before I get there.

# DSC #12: The DSC and the New Programming Language

by Katherine Bowers, Quinn Dombrowski, and Roopika Risam
November 2, 2021 https://doi.org/10.25740/wr764jc8892




We need to talk about coding.

Ever since DSC 7: The DSC and Mean Copyright Law, we’ve been putting an awful lot of code in these books. Text comparison algorithms, machine-generated text, principal component analysis, sentiment analysis.

But we’ve never had The Talk about Coding and DH and honestly, it’s overdue, because we don’t want you guys to get the idea that we’re assuming that *of course you, dear reader, are comfortable with code*. (What’s next, providing a properly LaTeX-formatted PDF download[1] so the Data-Sitters Club might fit in with the computer scientist?)

It’s hard to do DH for long without running into code and programming languages that you don’t know. This is true whether you’re new to DH, or built a whole career on it. Turns out it’s even hard to watch the new Netflix “Baby-Sitters Club” series for long without a casual mention that Claudia’s sister Janine is learning Python. Janine is a genius, but does it take a tech genius to learn Python? Most of the Data-Sitters Club books have been in Python, but we’ve recently started branching out into R.

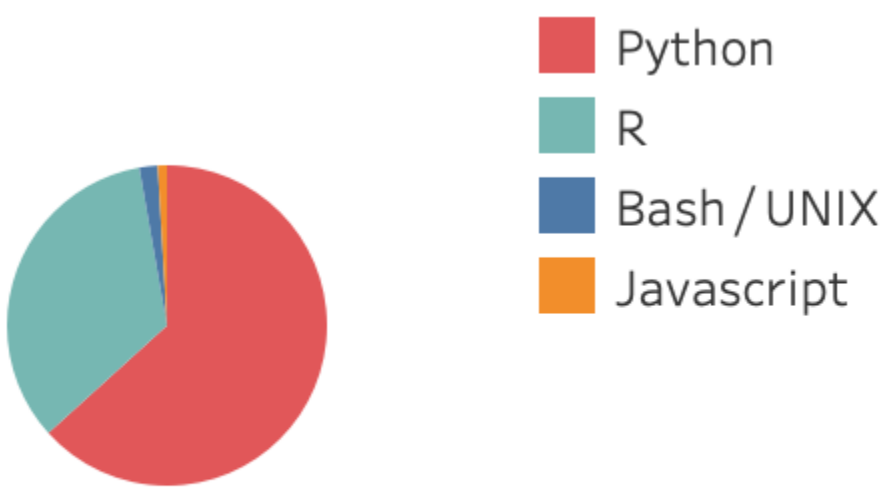
What’s the difference between a programming language, a library / package, and a model? If you’re going to dive into learning to code, what language should you choose? What’s it like to work with more than one? “The DSC and the New Coding Language” has got you covered – with some help from DH Twitter folks who answered our survey about all this.

## Surveying DH Twitter

We're not shy about the Data-Sitters Club being an *opinionated* project. We pour our emotions, our struggles – and, yes, our opinions – into these books. But when it comes to coding languages for computational text analysis, we don't want you to just take our word for it. So in July 2021, we threw together a short survey about programming languages for DH computational text analysis, and shared it (and re-shared it) on Twitter.

Here's what we found out from the 64 responses.

**Preferred programming language**




2/3 of the respondents preferred to use Python for computational text analysis. Other than one very R-happy afternoon when the responses were 50/50 (and yes, watching the responses come in was how I kept myself entertained in early July), this was more or less the ratio throughout the survey.

Going into the survey, I assumed that there would be a connection between the languages people worked with and the programming language they preferred. At least, that's a big part of my personal rationale for why I do Python.

**Choosing a programming language**

If you do reach the point where you've concluded that learning to code is the best path for being able to do what you want to be able to do, how do you pick which language to learn?

**Step 0**. Don't stress. Once you've learned one programming language well, it's easier to learn other ones if it turns out you didn't pick the "ideal" one the first time around.

**Step 1**. Look around in the communities that you're part of. Ask around in your department, your library, your DH center. What programming languages– if any– do people use? Don't forget your virtual communities! Who's in your circles on DH Twitter (if you use it), or what mailing lists are you part of?

**Step 2**. Think, specifically, about what you want to be able to do. If you have lots of people around you doing computational text analysis with Python and R, but what you want to make are critical digital editions with TEI and the XXX-toolkit, it might be better to look towards less-close connections in your community who are doing more-similar things to shape your decision. All things being equal, having access to people with expertise counts for a lot. But given the amount of work that you'll need to put into learning any programming language, it helps a lot if the programming language is a good fit for what you want to do with it.

**Step 3**. Compare the options you've found. Some things have pretty unambiguous paths. If you want to do text encoding and critical digital editions, you're probably going to need TEI and the XXX-toolkit. If you want to do web-based visualizations, you're going to need to learn Javascript. If you want to do the kinds of stuff we've been doing in the Data-Sitters Club, though, Python and R are both reasonable choices. And maybe someday, your answer will be "both, to different extents". Just pick one. Whichever one gives you the best access to a support community. Choose it, and ask your support community for recommendations for resources to get started. But if it really truly is a toss-up between R and Python, take this advice from Mark: "If you want it to be really easy to run statistical analyses at the cost of being harder to work with strings [e.g. text objects], choose R. If you want strings to be easy and can put up with struggling to do statistics, choose Python."

**Step 4**. Work steadily and hard. Becoming proficient in your first programming language – whatever it is – lays the foundation for anything else you'll need to learn about different programming languages. I won't downplay this: you can't just work your way through an introductory book and think you're set. You have to use it, apply it to your own questions, even the ones that aren't covered by a book. Train your brain to translate your ideas into code using that language. Learn how to effectively search for answers online. Resist the urge to copy and paste chunks of code you find online, or chunks of code that you've written before. Type them from scratch. Every time. EVERY. TIME. Until you can do it off the top of your head. And sometimes life works out that a couple months pass without you writing code, and when you get back to it, you've forgotten things and the skills you put so much effort into developing are rusty. It's okay. Do it again. It'll come back easier the second time.

prefer R (especially the tidyverse) tend to dislike Python's pandas package (which has some features similar to core R and the tidyverse, but not quite the same).

People are dissatisfied with package installation and managing dependencies (like when you want to install two packages, and they each require a different version of another package). Respondents also complained about the lack of backwards compatibility between Python 3 and Python 2. And Python can be a real pain to install on Windows.

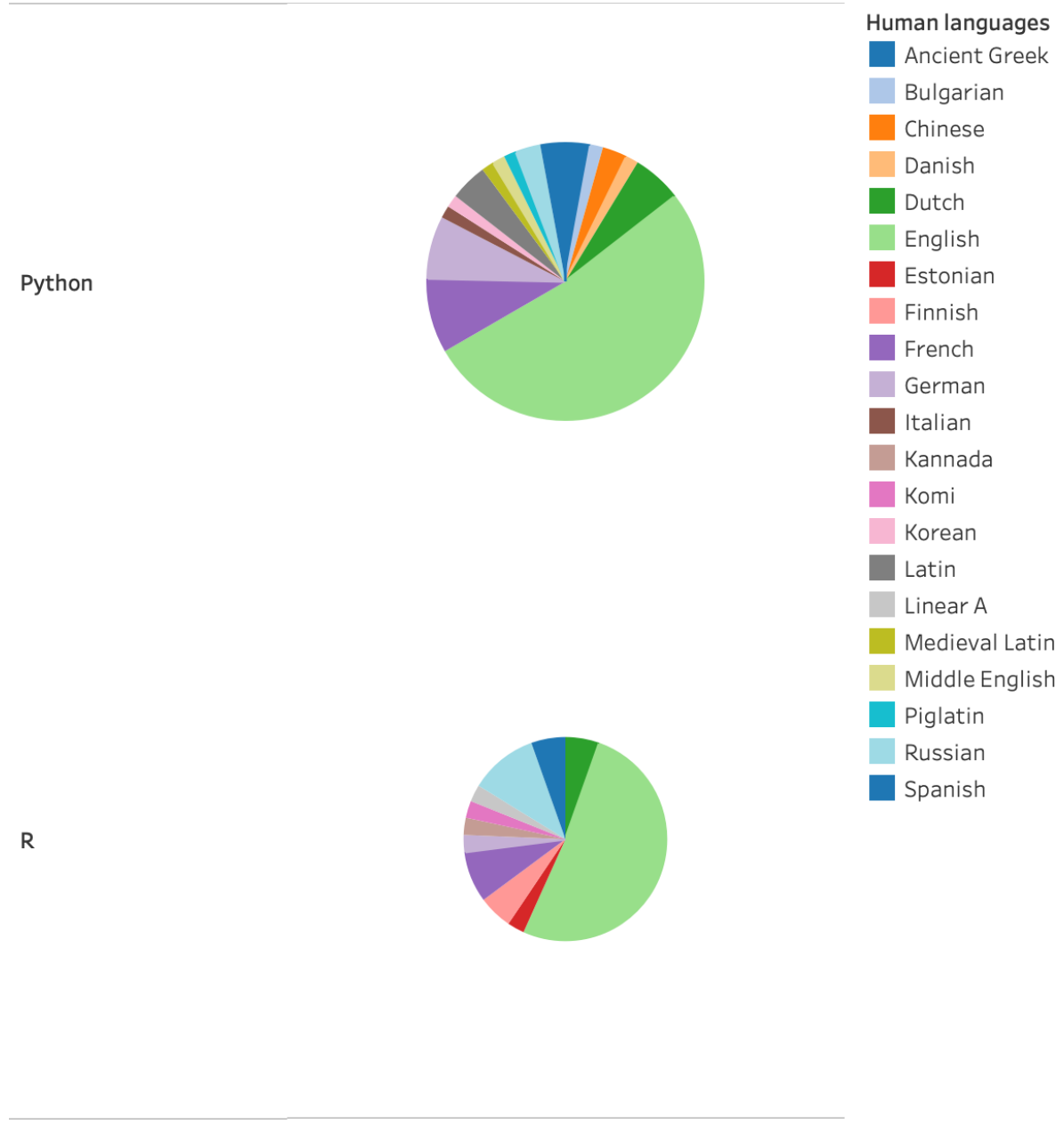
**R's weaknesses**

Similarly, the weaknesses identified for R are the inverse of Python's strength. Multiple respondents complained about R being slow with large data sets. There were also lots of complaints about the syntax, which I found vindicating: "Syntax is too much black magic and doesn't port well to learning concepts for other languages imho", "The syntax and programming paradigm is just unintuitive and too much a mix of everything", "Messy, confusing syntax; idiosyncratic approach to data types and structures", "From my few encounters, it not only feels syntactically quite different from Python, Java, and other popular languages, but it feels grammatically different as well. In other words, I feel like I need to think completely differently when using it, beyond superficial differences in syntax."

## Conclusion

Do you need to learn to code? I dunno, you tell me! Do you actually have a problem where there's no other practical way to solve it? Or can you get the answers you need with a little bit of Microsoft Excel? If you want to level up, maybe what you need is a browser plug-in for web scraping and OpenRefine for data wrangling. Just because you could do those things using a programming language doesn't mean you have to… or even that you should. Data-wrangling skills are valuable, in and out of academia – but learning to code is not a prerequisite. And honestly? Don't let the drum beat of "YOU SHOULD LEARN TO CODE" get into your head. Because if you're a humanities scholar, odds are that you don't actually want to "learn to code" as such. Not really. What you might actually want is to be able to do things you couldn't otherwise do – and that doesn't necessarily involve coding. The process of actually learning to code is stressful, frustrating, and time-consuming. It can be worth it if you're getting something you want out of it (being able to do a concrete thing that you weren't able to do before)… but quite another if you're doing it out of a sense of pressure, guilt, or insecurity. "Learning to code" is a straight-up miserable path to walk down if you're doing it mostly because you feel like you have to in order to be a "real digital humanist". There are so many other things you could put your time and effort towards, like improving your project management, people management, project budgeting, or even paleography skills!

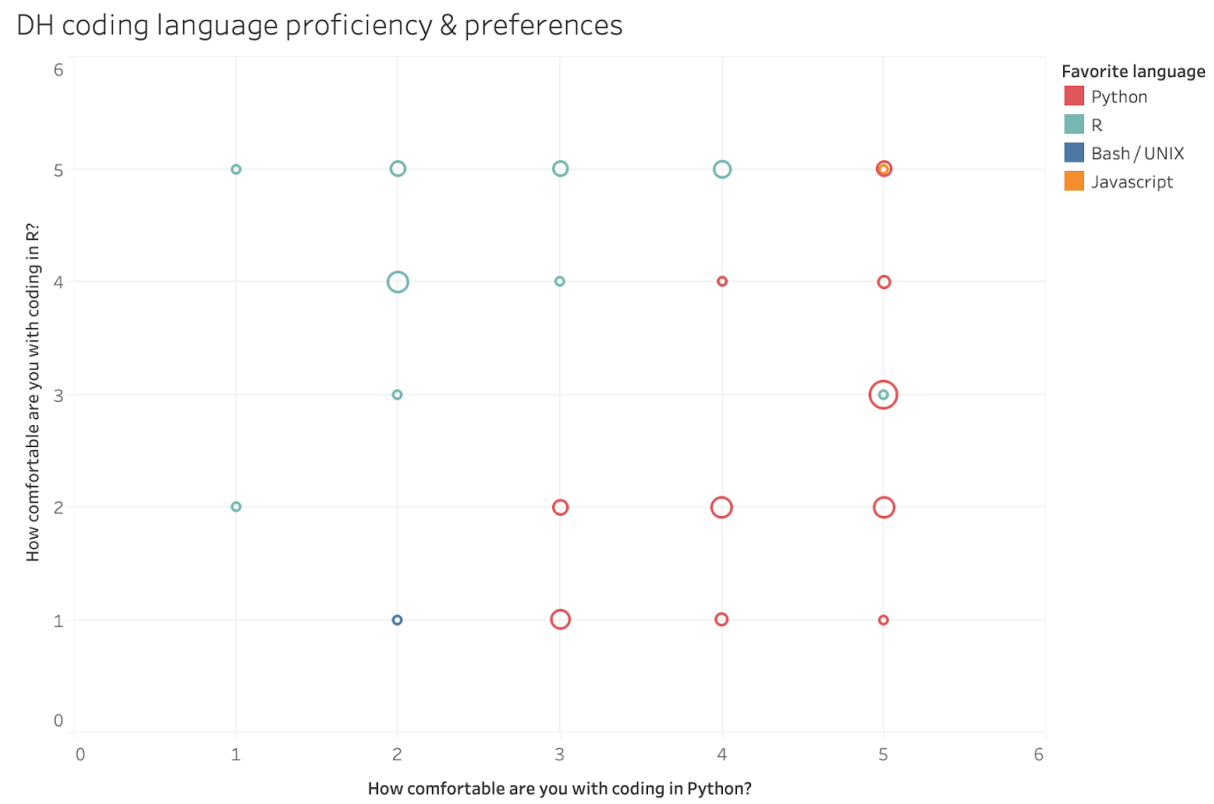
**Programming vs. human languages**




But that theory didn't really work out. For both R and Python, half the respondents said they work with English (sometimes along with other languages). But I'm pretty sure the distribution beyond that is pretty random; I don't think there's anything inherently Estonian-friendly about R or Italian-friendly about Python. I know for a fact that there's a lot of useful Python packages for Russian (like this one for Russian poetry analysis) that don't exist for R, and yet there's twice as many respondents who say they work on Russian who prefer R over Python. 🤷

**We prefer what we know**

So why **do** people prefer one language over the other? It turns out that the adage "familiarity breeds contempt" doesn't apply at all to programming languages. For most people, there's a discrepancy between how well they know Python and how well they know R. People who know R better prefer it, and the same for Python. And that makes sense: after all, people learn programming languages to use them, and a language you know better lets you do more things.




Of the people who said that Python was their preferred language, 58% said they were "extremely comfortable with Python", vs. the 50% of people who preferred R who claimed the same level of proficiency.

**Likes and dislikes**

The survey also asked what people liked and disliked about each language, regardless of what their preference was. While I was a little disappointed at the shortage of snarky responses (how is it that, out of 64 people, not a single one mentioned the – especially pirate-themed – pun affordances of R!?), the picture these responses painted gestures towards a state of affairs where Python and R are far from distinct, entrenched camps (like TEI vs computational text analysis can feel like sometimes– see the discussion of TEI as a cult in DSC #5) but something more like an interoperable continuum, with people tending to favor one or the other, but venturing between them as needed. And there's not much difference between what the people who prefer Python and the people who prefer R see as the strengths and weaknesses of each language.

(If you don't know Python or R yet, don't worry if the details below don't make a lot of sense. The tl;dr is that both languages have their strengths and weaknesses.)

**Python's strengths**

Across the programming language preference spectrum, people appreciate the community, libraries, and tutorials available for Python, especially NLP libraries where "new stuff [is] implemented there first". (We talked about this a little in DSC #11: The DSC and the Sentiment Snobs.) I'm not the only one to feel like you can easily find the answers to your Python problems online; someone else noted "basically all my problems have been solved on Stack Overflow".

Multiple people also mentioned Python syntax as an upside; one respondent explained, "It's very close to how one would describe a routine in natural language," which was echoed by comments like "The code feels 'readable' so it's not too bad to pick up an old program," and "The proper way to do something is usually exactly what you expect; it's hard to mess up."

Many respondents referred to Python as "fast" compared to R, and a few packages that got specific shout-outs were Beautiful Soup (for dealing with XML and HTML), machine learning and deep learning libraries, and NLTK (Natural Language Toolkit, which we've used parts of for various DSC books).

**R's strengths**

Across the board, people agreed that R's strengths lay in visualization (especially the ggplot package) and statistical methods – which makes a lot of sense, since R's origins are in another statistical programming language, S. (Does anyone else ever feel like they've fallen into an episode of *Sesame Street*, through a lens darkly, when talking about R?)

Lots of people also mention the "Tidyverse" (a set of R packages with similar syntax and philosophy that are designed for data wrangling and data science methods). Pipes get multiple mentions, and lots of people like R's native vectorization.

The people who prefer R tend to say that there's a huge community, but since Python people say that too, I suspect it's mostly a matter of whether you know your way around the places where you can find the information you need.

**Python's weaknesses**

Reading through the survey results, one gets the sense that Python's weaknesses are most evident in the context of R's strengths. One respondent who cited R's ggplot visualization package as their favorite thing about R also referenced Python's (roughly equivalent, but not exactly) matplotlib visualization package as their least favorite thing about Python. People who