Reflections on a decade of teaching statistical computing

DSC 2025

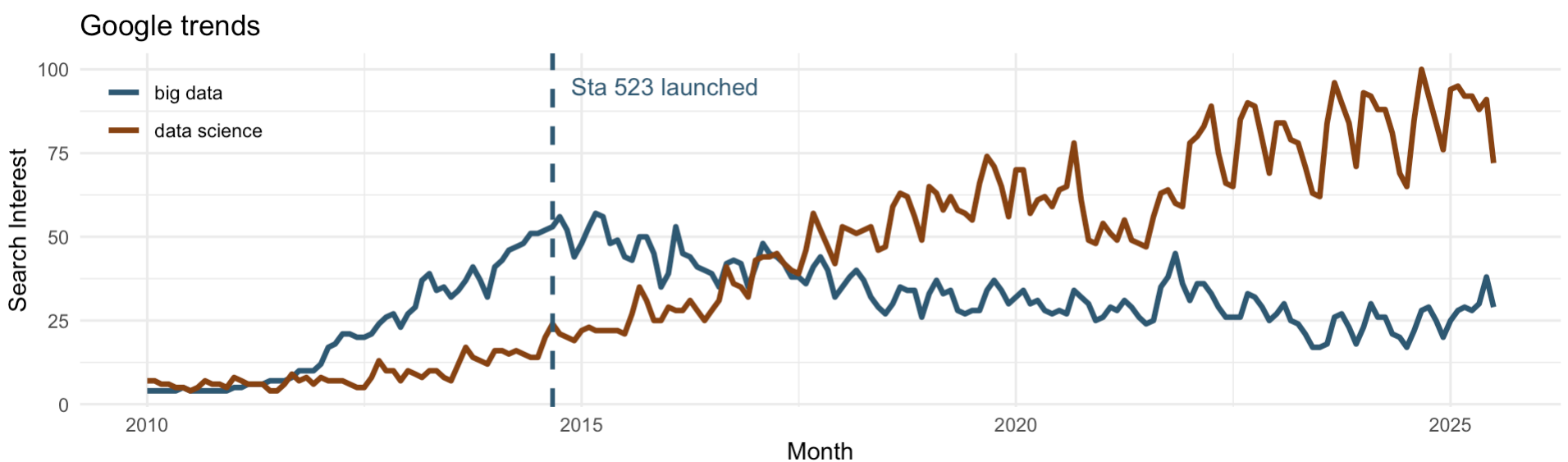
Colin Rundel
Duke University

Who am I

- B.S. in Biology, Caltech - 2003
- Ph.D. in Statistics, UCLA - 2012
- Professor of the Practice, Dept. of Statistical Science, Duke - 2015

Courses timeline

- 2014 Fall - Sta 523 - Programming for Statistical Science first offered (1st year MS required course)
- 2016 Spring - Sta 323 - Statistical Computing first offered (3rd year UG elective)
- 2018 Spring - Sta 199 - Introduction to Data Science first offered (1st year intro course)
- 2023 Spring - Sta 323 added as a major requirement



Things we got right

Curriculum

Core content and learning objectives of the course were and continue to be solid:

- Programming in R
- Data munging
- Visualization
- Text data
- Web scraping & APIs
- Interactive web apps
- Spatial data
- Databases

Reasonably innovative in 2014, now standard for most data science courses

Some drift and evolution over time but mostly in the specifics of the topics not the topics themselves.

Tech Stack & reproducibility

From the beginning the courses have been built to teach/use/require fully reproducible workflows:

- R (scriptability)
- git & GitHub (version control)
- R Markdown / Quarto (literate programming)

All assignments are scaffolded Quarto documents - delivered and collected via GitHub.

Students only turn in the qmd not its rendered output - we run all code to ensure reproducibility.

Aside - Why not Python?

This is a reasonable question and these courses could be successfully taught with Python (now)

For us specifically,

- For our masters program - there is a 2nd course, Sta 663, that does
- For our undergraduate program - there is a CS department that does
- Staffing Python focused courses has been difficult
- Vertical integration of common tools throughout the curriculum is super powerful

Open materials

All of the course materials have been open and publicly shared since the very first iteration of the course.

stat.duke.edu/~cr173/Sta523_Fa14/

⋮

sta523-fa24.github.io

stat.duke.edu/~cr173/Sta323_Sp16/

⋮

sta323-sp25.github.io

Open-source & freely available textbooks are invaluable, e.g.

- R for Data Science
- Advanced R
- R Packages

Things we didn’t get (as) right

Computing Environment

First offering of the course had students BYOD and manage their own software stack - it was a disaster.

The technology stack for these courses is critical but we also need it to get out the way of our students,

- managing a computational environment is not a learning objective
- minimize the work students have to do to do the work they want/need to do
- stable, accessible, and consistent computational environments are critical

Solution - remotely accessible web-based computing interfaces, e.g.

Posit Cloud, Posit Workbench, Open OnDemand

Assessment

Courses have always included small team based assignments, biweekly largish and open-ended coding tasks

- Requires assigning teams (we use random assignment)
- Requires team based assessment (peer evaluations, commit counts, etc.)
- Expect equal effort not equal lines of code
- Good for learning, not so good for individual assessment

We have since added individual “midterms” at the middle and end of the course.

- Same structure as the team based assignments
- Slightly shorter and generally summative
- Open internet closed other people

Pedagogy

Weekly interactions include two 75 minute lectures and one 75 minute lab session.

Early tendency was to lean heavily on traditional lectures,

- Need to teach students to learn how to learn and how to problem solve
- Learning by doing - ungraded in class exercises
- Learning by example - live coding demos in class with extensive narration

The next decade

AI

Not a unique problem for these courses but something that we need to come to terms with,

- In the last year or two - LLMs have gone from being able to complete 30%-50% of an assignment to closer to 80%
  - Nature of the assignment determines this percentage (lack of training data and other limitations)
- Hard to assess exactly but usage is nearly ubiquitous at both UG and MS level
- We have always had to deal with students using Stack Overflow, Google, etc. to find solutions
- Goal is for students to use code to solve a problem - do we care how that code is generated?

Modes of AI interaction

- ChatBots (ChatGPT, Gemini, etc.)
- Code Assistants / IDE autocomplete (GitHub Copilot, Cursor, etc.)
- Agentic (Claude Code, OpenAI Codex, etc.)

I don’t know, but suspect, different approaches have different implications for learning

- How do we model effective and safe usage of these tools?
- How do we handle issues of equity around access to these tools?

Thank You

 rundel.github.io

 rundel

 rundel@gmail.com
colin.rundel@duke.edu

 Course materials
sta523-fa24.github.io
sta323-sp25.github.io
sta663-sp25.github.io