PS239T: Introduction to Computational Tools for Social Science

Jae Yeon Kim

10/9/2020

Overview

This course will help social science graduate students to think computationally and develop proficiency with computational tools and techniques, necessary to conduct research in computational social science. Mastering these tools and techniques not only enables students to collect, wrangle, analyze, and interpret data with less pain and more fun, but it also let students to work on research projects that would previously seem impossible.

Textbook

The online book version of the course materials is currently work in progress. I am loosely aiming the completion of this book in 2020.

Objectives

- The goal of this course is to help students become a more efficient and innovative researcher by leveraging the power of automation.
- The course is currently divided into two main subjects (fundamentals and applications) and six main sessions.

Part I Fundamentals

- In the first section, students learn best practices in data and code management using Git and Bash.
- In the second, students learn how to wrangle, model, and visualize data easier and faster.
- In the third, students learn how to use functions to automate repeated things and develop their own data tools (e.g., packages).

Part II Applications

- In the fourth, students learn how to collect and parse semi-structured data at scale (e.g., using APIs and webscraping).
- In the fifth, students learn how to analyze high-dimensional data (e.g., text) using machine learning.
- In the final, students learn how to access, query, and manage big data using SQL.

We will learn how to do all of the above mostly in **R**, and sometimes in **bash** and **Python**.

- Why R? R is free, easy to learn (thanks to `tidyverse` and RStudio), fast (thanks to `rcpp`), runs everywhere, **open** (16,000+ packages; counting only ones available at CRAN), and has a growing massive and inclusive community (`#rstats`).
- Why R + Python + bash?

> "For R and Python, Python is first and foremost a programming language. And that has a lot of good features, but it tends to mean, that if you are going to do data science in Python, you have to first learn how to program in Python. Whereas I think you are going to get up and running faster with R, than with Python because there's just a bunch more stuff built in and you don't have to learn as many programming concepts. You can focus on being a great political scientist or whatever you do and learning enough R that you don't have to become an expert programmer as well to get stuff done." - Hadley Wickham

- However, this feature of the R community also raises a challenge.

  > Compared to other programming languages, the R community tends to be more focused on results instead of processes. Knowledge of software engineering best practices is patchy: for instance, not enough R programmers use source code control or automated testing. Inconsistency is rife across contributed packages, even within base R. You are confronted with over 20 years of evolution every time you use R. R is not a particularly fast programming language, and poorly written R code can be terribly slow. R is also a profligate user of memory. - Hadley Wickham

- RStudio, especially the tidyverse team, has made heroic efforts to amend the problems listed above. In this course, you will learn these recent advances in the R ecosystem and how to complement R with Python and Bash.

Logistics

Contributors

- Instructor and content developer: Jae Yeon Kim (jaeyeonkim@berkeley.edu)

Time and location

- Lecture: TBD (Zoom)
- Section: TBD (Zoom)

Office hours

By appointment with ...

Slack & GitHub

- Slack for communication (announcements and questions). You should ask questions about class material and assignments through the Slack channels so that everyone can benefit from the discussion. We encourage you to respond to each other's questions as well.

- GitHub for everything else, including turning in assignments (except final project proposals, which will be submitted to Slack). Students are required to use GitHub for their final projects, which will be publicly available, unless they have special considerations (e.g. proprietary data). All course materials will be posted on GitHub at https://github.com/jaeyk/PS239T, including class notes, code demonstrations, sample data, and assignments.

Accessibility

This class is committed to creating an environment in which everyone can participate, regardless of background, discipline, or disability. If you have a particular concern, please come to me as soon as possible so that we can make special arrangements.

Course requirements and grades

This is a graded class based on the following:

- Completion of assigned homework (50%)
- Participation (25%)
- Final project (25%)

Assignments Assignments will be assigned at the end of every session. They will be due at the start of the following class unless otherwise noted. The assignments will be frequent but each of them should be fairly short.

You are encouraged to work in groups, but the work you turn in must be your own. Group submission of homework, or turning in copies of the same code or output, is not acceptable. Remember, the only way you actually learn how to write code is to write code.

Unless otherwise specified, assignments should be turned in as pdf documents via the bCourses site.

Class participation The class participation portion of the grade can be satisfied in one or more of the following ways:

- attending the lecture and section (note that section is non-optional)
- asking and answering questions in class
- contributing to class discussion through the bCourse site, and/or
- collaborating with the campus computing community, either by attending a D-Lab or BIDS workshop, submitting a pull request to a campus github repository (including the class repository), answering a question on StackExchange, or other involvement in the social computing / digital humanities community.

Because we will be using laptops every class, the temptation to attend to other things during slow moments will be high. While you may choose to do so, I do request that you think of your laptop screen as in the public domain for the duration of classtime. Please do not load anything that will distract your classmates or is otherwise inappropriate to a classroom setting.

Final project The final project consists of using the tools we learned in class on your own data of interest. First- and second-year students in the political science department are encouraged to use this as an opportunity to gather data to be used for other courses or the second-year thesis. Students are required to write a short proposal by March (no more than 2 paragraphs) in order to get approval and feedback from the instructor.

During sections in April we will have **lightning talk sessions** where students present their projects in a maximum 5 minute talk, with 5 minutes for class Q&A. Since there is no expectation of a formal paper, you should select a project that is completable by the end of the term. In other words, submitting a research design for your future dissertation that will use skills from the class but collects no data is not acceptable, but completing a viably small portion of a study or thesis is.

- Final project rubric
- Final project template
- Final project examples

Class activities and materials

Lecture Classes will follow a "workshop" style, combining lecture and lab formats. The class is interactive, with students programming every session. During the "skills" parts of the class, we will be learning how to program in R, UNIX (bash), and Python by following course notes and tutorials. During the "applications" sections, we will follow a similar structure, with occasional guest speakers.

Section The "lab" section will generally be a less formal session dedicated to helping students with materials from lecture and homework. It will be mostly student led, so come with questions. If there are no questions, the lab turns into a "hackathon" where groups can work on the assignments together. Section is required unless prior permission to miss it is obtained from both the instructor and one's groupmates. Attending office hours is not a substitute for attending section.

Computer requirements The software needed for the course is as follows:

- Access to the UNIX command line (e.g., a Mac laptop, a Bash wrapper on Windows)
- Git
- R and RStudio (latest versions)
- Anaconda and Python 3 (latest versions)
- Pandoc and LaTeX

This requires a computer that can handle all this software. Almost any Mac will do the job. Most Windows machines are fine too if they have enough space and memory.

You must have all the software downloaded and installed PRIOR to the first day of class. If there are issues with installation on your machine, please contact the section assistant, Julia Christensen, for assistance.

See B_Install.md for more information.

Course schedule

Part I Fundamentals

- Week 1 Computational thinking and setup
- Week 2 Managing data and code
- Week 3 Tidy data and why it matters
- Week 4 Wrangling data
- Week 5 Wrangling data at scale
- Week 6 Modeling and visualizing tidy data
- Week 7 From for loop to functional programming
- Week 8 Developing your own data tools

Part II Applications

- Week 9 HTML/CSS: web scraping
- Week 10 XML/JSON: social media scraping
- Week 11 Supervised machine learning
- Week 12 Unsupervised machine learning
- Week 13 Database, SQL, MongoDB, and Spark
- Week 14 Wrap-up
- Week 15 Final presentation

Questions, comments, or suggestions

Please create issues, if you have questions, comments, or suggestions.

Special thanks

This course is a remix version of the course originally developed by Rochelle Terman then revised by Rachel Bernhard. Other teaching materials draw from the workshops I have created for D-Lab and Data Science Discovery Program at UC Berkeley.