Teaching statistical computing to undergraduates

K. Jarrod Millman

Division of Biostatistics, UC Berkeley

Philip B. Stark

Department of Statistics, UC Berkeley

MS78: Teaching computational thinking and practice

SIAM Conference on Computational Science & Engineering
Sunday, March 15, 2015

What training is offered?

Classes

- Concepts in Computing with Data (133)
- Introduction to Statistical Computing (243/244)
- Reproducible and Collaborative Statistical Data Science (159/259)

Bootcamps

- R, Python, Software carpentry, etc.

Groups

- BIDS, DLab, BITSS, Social computing, Hacker within, Py4science, etc.

Other departments

- EECS, Biostatistics, School of Information, etc.

Concepts in Computing with Data.

UC Berkeley, Statistics 133, Summer 2014

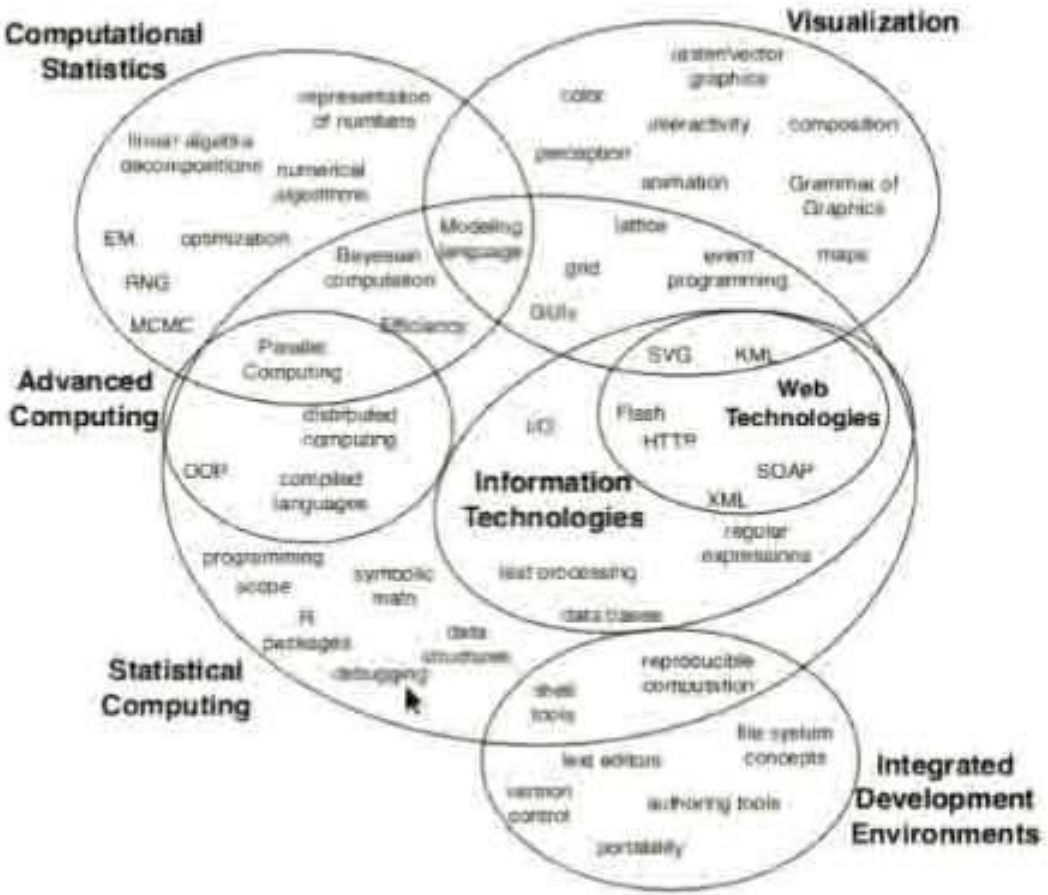
An introduction to computationally intensive applied statistics. Topics may include organization and use of databases, visualization and graphics, statistical learning and data mining, model validation procedures, and the presentation of results

- Instructor
  - Jarrod Millman
  - OH: W 12-2P (Barker 210)
- GSI
  - Karl Kumbier
  - OH: W 10-12 (145 Moffitt)
- Session Dates: 06/09-08/15/14
  - Class meets MTWTF 9-10A in 145 MOFFITT
  - Lab meets TuTh 10-11A or 12-1P in 340 EVANS
  - Midterm on 7/9 at 9-10A in 145 MOFFITT
  - Final on 8/15 at 9-10A in 145 MOFFITT
- CNN: 79690

Prerequisites: Familiarity with basic concepts in probability and statistics is important. Being comfortable with matrices, vectors, basic set theory, functions, and graphs will also help. There is no prerequisite for programming.

http://www.jarrodmillman.com/stat133-summer2014

What to cover?



Nolan, D., & Temple Lang, D. (2010). Computing in the statistics curricula. *The American Statistician*, 64(2).

What did I want to accomplish?

Goal: write *valid, executable R programs* which I could run on my computer and behaved as specified

Challenge: students *lack context* for why this is important and their past experience had *poorly prepared* them

Topics

The goal of this course is to introduce you to a variety of programs and technologies that are useful for organizing, manipulating, and visualizing data with a focus on the R statistical computing environment. Topics may include:

- Working at the (Bash) command line
- Implementing reproducible research
- Version control using Git and GitHub
- Basics of R programming (data structures, control flow, debugging, etc.)
- Simulation and random number generation
- Exploratory data analysis and dimension reduction (PCA)
- Hypothesis testing (t-tests)
- Clustering (k-means, hierarchical clustering) and classification (k-nn, CART, Random Forests)
- Linear and logistic regression

Class setup

- Focus on students solving problems
- Tables and chairs (not stadium seating)
- Everyone works on their own laptop
- Extensive hands-on practice
- Extensive use of Git
- Running case study using Duke/Potti cancer trials

The first week

- Monday: Course overview and motivation
- Tuesday: BASH, Git, and GitHub
  - Lab: Install
- Wednesday: R datastructures
- Thursday: More Git
  - Lab: Git
- Friday: More R
  - 1st homework assigned

Working at the command line

- Foreign to most students
- Slow to embrace
- Lectures were short and focused on live demos
- Daily assignments

Version control using Git and GitHub

Used for

- homework, labs, quizzes, and exams

Required me to

- teach Git
- provide clear instructions

Required students to

- install necessary software
- work at commandline
- work with text editor

Basics of R programming

Focused on

- Datastructures (vectors, matrices, lists, dataframes)
- Vectorized operations and the `apply` family
- Using and writing functions
- Exploratory data analysis (summary statistics, boxplots, PCA)

Introduced

- Simulation and random number generation
- Hypothesis testing
- Clustering
- Regression

Simplified example assignment

```
library(RUnit)

to_celsius = function(temp) {
    # take a vector of Farenheit temperatures
    # return a Celsius vector
}

tryCatch(
    checkEquals(to_celsius(c(32, 100, 210)),
                c(0, 340/9, 890/9)),
    error = function(err) print(err)
)
```

Automating grading

- Students submit work via Git
- Pull from each repo
- Source the submitted R file
- Test variables and functions
- Push scores and logfiles to each repo

Why automate?

I do not trust myself to visually confirm the correctness of my own code. Why should I trust a myself or a TA to evaluate student code?

Moreover, automating the grading enabled me to:

- Assign and grade more student work
- Reduce latency in providing feedback
- Spend more time working directly with students

How did I automate?

- Wrote some Python code
- Used RPy to work with R files
- Test code correctness (did not use automated unit testing framework)
- Allow resubmissions for first several assignments
- Allow students to petition for a regrade (then regraded everyone)

`https://github.com/jarrodmillman/gradebook`

Code quality vs. quantity

- Program correctness baseline
- Structured solutions
- Code reviews
- Office hours

Outline

Lessons learned

- Filesystem and data organization
- Command line interface
- Interpreter confusion
- But it works (even though it doesn't run)
- Google and random trial-and-error

What would I improve

- Start with many mini-assignments
- More focus on code review
- More emphasis on group projects to increase amount of Git
- More practice fixing broken or poorly written code
- More emphasis on concepts with short answer quizzes

Statistics 159/259 — Reproducible and Collaborative Statistical Data Science

A project-based introduction to statistical data analysis. Through case studies, computer laboratories, and a term project, students will learn practical techniques and tools for producing statistically sound and appropriate, reproducible, and verifiable computational answers to scientific questions. Course emphasizes version control, testing, process automation, code review, and collaborative programming. Software tools may include Bash, Git, Python, and LaTeX.

Questions and discussion