Scaling Intensive Computational Education: The R Bootcamp
Michael Levy
Environmental Science & Policy

Who?

What?

Who?

Why?
Science increasingly requires computational competence, but education has not kept pace with demand.
The non-profits Software Carpentry and Data Carpentry work to fill this gap with intensive, two-day introductions to computing tools and best practices, but they are labor intensive with instructor-to-learner ratios of 8:1, are superficial on account of introducing multiple tools in just two days, and often fail to build community because instructors are not usually from the university.

Here, I sought to take the best of the Software/Data Carpentry model and provide a deeper introduction to one key computational tool, over five days, with hundreds of students, and build community in the process.

What worked well

Motivation precedes detail
Hook the goals first and learners will push through the details. I started with having students play with visualizations like the one in the flyer to the left.

Live coding
Throughout, I wrote code live, in front of the class. This normalizes mistake-making and demonstrates debugging. It also slows me down and builds in flexibility. I piped the code to students’ browsers in realtime, so they could catch up if they fell behind, and they have a permanent reference of the material as they learned it.

In-class exercises
Learning comes from doing, so have the students do, often.

Stickies & good assistants
This is a Software/Data Carpentry tactic that works wonders. Give each student one red and one green sticky note, and they can signal when they need help or are done with an exercise without interrupting. I had 1-3 assistants at any time, and that was massively helpful.

Feedback
At the end of each day, students filled out a comprehension survey and we started each day looking at yesterday’s results. This shows the students I care and allows me to adjust.

Advanced exercises
Handling variance in learners is always a challenge, and in a course for novices, variance increases with time. Having engaging optional exercises goes a long way to solving this.

Keeping it flexible
Writing each day’s curriculum after the preceding day made for a challenging week but the added responsiveness was well worth it.

Reinforcement
Once is never enough. Circle back on concepts to reinforce and connect lessons.

Improvements for next time

Start with the standards
The R ecosystem is full of handy extensions. dplyr and RMarkdown are great, but non-standard evaluation and dynamic documents have to come after base-R syntax and scripts.

More-frequent exercises
An afternoon of exercises is too much. Interspersing shorter exercises with solutions and lectures at a 60 - 90 minute rhythm worked well.

Keep the statistics out of statistical computing
R is fundamentally a statistical programming language; however, students knowledge of and comfort with statistics varies tremendously. It is often hard to know where data analysis ends and statistics begins (e.g. the rating x budget “interaction effect” in the flyer plot), but I would err away from statistics in the future.

Post-course evaluations

Thanks!
• GradPathways & Professors for the Future for the opportunity.
• Software Carpentry and especially Greg Wilson for the excellent computational pedagogy training.
• The volunteer assistants for their excellent, tireless work, especially Myfanwy Rowlands, Mike Koontz, and Nick Ulle.

@ucdlevy | malevy@ucdavis.edu | www.michaellevy.name
Professors for the Future, 2015-2016

PFTF Fellow: Michael Levy
Program: Ecology
Major Professor: Mark Lubell

Project Title: "A Bootcamp for the Statistical Programming Language R."

Project Summary:

Science increasingly requires computational competence, but computing education has not kept up with demand. This week-long intensive introduction to the programming language R aimed to help fill that gap.

The bootcamp was built on the Software Carpentry model of short, welcoming, intensive introductions to computational tools. Software carpentry, however, typically has student: instructor ratios below ten and introduces three major tools in two days. Here, I adapted Software Carpentry strategies and tactics to provide a deeper introduction to R over the course of a week with a student: instructor ratio above 50.

The course was designed for students with little-to-no experience with computer programming. By the end of the week, students understood the importance and mechanics of computational workflows, were able to manipulate data efficiently and reproducibly, and could make publication-quality graphics.

Learning a programming language is like learning any other language -- you have to do it to learn it. You make mistakes, and you learn by fixing them. Unfortunately, "getting unstuck" is a learned skill and can be endlessly frustrating for beginners. This course had students work on increasingly open-ended challenges throughout the week, with the instructor and assistants available to help and to demonstrate how to help themselves. Mornings were lectures mixed with short exercises; afternoons had students work on short analysis projects to solidify understanding. The exercises and projects progressed from highly structured to open-ended so that students not only learned R syntax, but also computational problem solving.

University of California, Davis