Teaching computational reproducibility for neuroimaging

K. Jarrod Millman [1,2,*]**, Matthew Brett** [3]**, Ross Barnowski** [4]**, and Jean-Baptiste Poline** [5]

[1]*Division of Biostatistics, UC Berkeley, Berkeley, CA, USA*
[2]*Berkeley Institute for Data Science, UC Berkeley, Berkeley, CA, USA*
[3]*College of Life and Environmental Sciences, Birmingham University, Birmingham, England*
[4]*Applied Nuclear Physics Program, Lawrence Berkeley National Laboratory, Berkeley, CA, USA*
[5]*Neurology and Neurosurgery, McGill University, Montreal, Quebec, Canada*

Correspondence*:
K. Jarrod Millman
millman@berkeley.edu

Word count: 9711
Figure and table count: 4

ABSTRACT

We describe a project-based introduction to reproducible and collaborative neuroimaging analysis. Traditional teaching on neuroimaging usually consists of a series of lectures that emphasize the big picture rather than the foundations on which the techniques are based. The lectures are often paired with practical workshops in which students run imaging analyses using the graphical interface of specific neuroimaging software packages. Our experience suggests that this combination leaves the student with a superficial understanding of the underlying ideas, and an informal, inefficient, and inaccurate approach to analysis. To address these problems, we based our course around a substantial open-ended group project. This allowed us to teach: (a) computational tools to ensure computationally reproducible work, such as the Unix command line, structured code, version control, automated testing, and code review and (b) a clear understanding of the statistical techniques used for a basic analysis of a single run in an MRI scanner. The emphasis we put on the group project showed the importance of standard computational tools for accuracy, efficiency, and collaboration. The projects were broadly successful in engaging students in working reproducibly on real scientific questions. We propose that a course on this model should be the foundation for future programs in neuroimaging. We believe it will also serve as a model for teaching efficient and reproducible research in other fields of computational science.

Keywords: neuroimaging, FMRI, computational reproducibility, scientific computing, statistics, education, Python language, data science

1   INTRODUCTION

Few neuroimaging analyses are computationally reproducible,[1] even between researchers in a single lab. The process of analysis is typically ad-hoc and informal; the final result is often the fruit of considerable trial and error, based on scripts and data structures that the author inherited from other members of the lab. This process is (a) confusing, leading to unclear hypotheses and conclusions, (b) error prone, leading to incorrect conclusions and greater confusion, and (c) an impractical foundation on which to build reproducible analyses.

Confusion, error, and lack of reproducibility are ubiquitous problems that programmers have been fighting since before "programmer" became a job title. There are widely accepted tools and processes to reduce these problems, including the Unix command line, version control, high-level readable programming languages, structured code, code review, writing tests for new code, and continuous automatic test execution.

Many researchers accept that learning these techniques is desirable, but believe that teaching them is too difficult, or would cost too much in class time that should be spent on topics specific to neuroimaging.

In the course we describe here, we tested our hypothesis that we could effectively teach *both* the tools for efficient and reproducible work *and* the principles of neuroimaging, by building the course around a substantial collaborative project, and putting the tools into immediate practice.

At intake, our students had little or no prior exposure to neuroimaging, or to the computational tools and process we listed above. We set them the open-ended task of designing and executing a project, which was either a replication or an extension of a published neuroimaging analysis, built from code they had written themselves. We required the analysis to be computationally reproducible; each project had to provide a short text file that gave a simple set of commands with which the grading instructor could fetch the data, run the analysis, and generate the final report, including figures.

2   BACKGROUND AND RATIONALE

Between us, we have many decades experience of teaching neuroimaging analysis. Like most other teachers of imaging, we have taught traditional courses with a mixture of lectures covering the general ideas of the analysis, combined with practical workshops using one of the standard imaging software packages.

We also have many years of experience giving practical support for imaging analysis to graduate students and post docs.

Over these years of teaching and support, we have come to realize that the traditional form of teaching does a poor job of preparing students for a productive career in imaging research. It fails in two fundamental ways:

- Computation with large complex datasets is difficult and distracting; without training in standard practice to reduce this distraction, we condemn our students to a life-time of inefficient work and slow learning.
- Standard teaching assumes either that the students already understand signal processing and linear algebra, or that they do not need to understand them. In our experience, both of these assumptions are mostly false, with the result that the students do not have the foundation on which they can build further understanding.

---

[1]Following Buckheit and Donoho (1995), we define an analysis as *computationally reproducible* when someone other than the original author of an analysis can produce on their own computer the reported results using the author's data, code, and instructions.

---

60 As a result, imaging researchers usually do not have the vocabulary, understanding, or shared tools to
61 collaborate fluently with researchers from other fields, such as statistics, engineering, or computer science.

62 The essential error in traditional teaching is one of emphasis; it gives priority to the overview at the
63 expense of the intellectual and practical foundations. The assumption is that the student will either know or
64 pick up the mathematical and practical basis, with the big picture as a framework to guide them in choosing
65 what to learn.

66 In fact we find that it is rare for students to go on to learn these foundations. They usually continue as they
67 were when they leave the course: as inefficient practitioners with little ability to reason about their analysis.

68 In contrast to the traditional approach, we emphasize:

69 • Efficient computational practice to decrease confusion, reduce error, and facilitate collaboration.
70 • The fundamental mathematical and statistical bases of the analysis, such as the linear model, using
71   computational tools to build, illustrate, and explain.

72 We were inspired by the famous epithet of Richard Feynman, found written on his blackboard after his
73 death: "What I cannot create, I do not understand."[2] We first taught the students to build code efficiently,
74 and then we taught them how the methods worked, by building them from code. Our aim was that our
75 students should be able to implement their own simple versions of the major algorithms they were using.

76 Our course put great emphasis on a final group project that accounted for more than half of the overall
77 course grade. We did this for two reasons:

78 First, we believe the motivations for computational reproducibility and the difficulties of collaboration
79 are too abstract to be meaningfully understood in the absence of a significant, concrete, group project. If a
80 project lasts a few weeks, it is possible to remember all the steps without carefully recording and explaining
81 them. If you have a small dataset and a handful of tiny functions, it is reasonable to throw them all in a
82 directory and post them online. As datasets get larger and the analysis more complex, it quickly becomes
83 impossible to keep track of what you have done without carefully organizing and recording your work.
84 Large datasets and complex analyses are typical in research.

85 Second, we intended to teach the students efficient reproducible practice with the standard tools that
86 experts use for this purpose (Millman and Pérez, 2014). Our own practice has taught us that the power of
87 these tools only becomes clear when you use them to do substantial work in collaboration with your peers.
88 In contrast, we have found that teaching "easy" tools that need less initial investment has the paradoxical
89 effect of making it harder for students to move on to the more powerful and efficient tools that they will
90 need for their daily work.

3 MATERIAL AND METHODS

91 We taught *Reproducible and Collaborative Statistical Data Science*,[3] during the fall semester of 2015.
92 The course was offered through the department of Statistics at UC Berkeley, and was open to upper-level
93 undergraduates (as STAT 159) as well as graduate students (as STAT 259).

94 The course entry requirements were Berkeley courses STAT 133 (Concepts in Computing with Data),
95 STAT 134 (Concepts of Probability), and STAT 135 (Concepts of Statistics). Together these courses provide

---

[2]http://archives.caltech.edu/pictures/1.10-29.jpg

[3]http://www.jarrodmillman.com/stat159-fall2015/

96 basic undergraduate-level familiarity with probability, statistics, and statistical computing using the R
97 language. Many students were from statistics and/or computer science; other majors included cognitive
98 science, psychology, and architecture. During the 15-week long semester, we had three hours of class time
99 per week in two 90-minute sessions, plus two hours of lab time. Students were expected to work at least
100 eight hours per week outside class. Project reports were due in week 17, two weeks after the last class.

3.1 Course overview

102 This was the entry for our course in the Berkeley course catalog:

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

3.1.1 Tools and process

109 We had three guiding principles in our choice of tools and process: (a) to teach efficient reproducible
110 practice with standard expert tools, (b) to teach fundamental mathematical and statistical principles using
111 explanation with simple code, and (c) students should be able to build their own analysis from basic
112 building blocks.

113 Applying these principles, we taught command line tools, version control, and document machinery
114 using text files. Rather than focusing on a specific neuroimaging analysis package, we taught scientific
115 coding with Python and its associated scientific libraries (Millman and Aivazis, 2011; Pérez et al., 2011),
116 and then showed the students how to run standard statistical procedures on imaging data using these tools
117 (Millman and Brett, 2007).

118 **Command line.** The Unix environment is the computational equivalent of the scientists' workbench
119 (Preeyanon et al., 2014). The Unix command line, and the Bash shell in particular, provides mature,
120 well-documented tools for building and executing sequences of commands that are readable, repeatable,
121 and can be stored in text files as scripts for later inspection and execution. Quoting Wilson et al. (2014)—the
122 Bash shell makes it easier to "make the computer repeat tasks" and "save recent commands in a file for
123 re-use."

124 The graphical user interface of operating systems such as Windows and macOS can obscure the tree
125 structure of the file system, making it harder to think about the organization of data as a hierarchy of
126 directories and files. The command line tools make the file system hierarchy explicit, and so make it easier
127 to reason about data organization.

128 **Version control.** Version control is a fundamental tool for organizing and storing code and other products
129 of data analysis. Distributed version control allows many people to work in parallel on the same set of files.
130 The tools associated with distributed version control make it easier for collaborators to review each other's
131 work and suggest changes.

132 Git is the distributed version control system that has become standard in open source scientific computing.
133 It is widely used in industry, with automated installers for all major operating systems. Web platforms

134 such as GitHub,[4] Bitbucket,[5] and GitLab[6] provide web interfaces to version control that simplify standard
135 methods of collaboration such as code review, automated testing, and issue tracking (see below).

Scientific Python. Python is a general purpose programming language, popular for teaching and prevalent
137 in industry and academia. In science, it has particular strength in astronomy, computational biology, and
138 data science. Its impact in scientific computing rests on a stack of popular scientific libraries including
139 NumPy (computing with arrays), SciPy (scientific algorithms including optimization and image processing),
140 Matplotlib (2D plotting), Scikit-Learn (machine learning), and Pandas (data science). The Nibabel library[7]
141 can load files in standard brain image formats as arrays for manipulation and display by the other packages.

Peer review. Regular peer review is one of the most important ways of learning to be an effective author of
143 correct code and valid data analysis. Git and its various web platforms provide a powerful tool for peer
144 review.

Pull requests are a version control interface feature pioneered by GitHub. A pull request presents proposed
146 changes to the shared repository in a convenient web interface. Collaborators can comment on the changes,
147 ask questions, and suggest improvements. The discussion on this interface forms a record of decisions, and
148 a justification for each accepted set of changes.

GitHub, like other hosting platforms for version control, provides an interface for creating *issues*. These
150 can be reports of errors that need to be tracked, larger scale suggestions for changes, or items in a to-do list.

Functions with tests. Functions are reusable blocks of code used to perform a specific task. They usually
152 manipulate some input argument(s) and return some output result(s). There are several advantages to
153 organizing code into functions rather than writing stream-of-consciousness scripts. Functions encapsulate
154 the details of a discrete element of work. A top-level script may call a well-named function to express a
155 particular step, allowing the user to "hide" the mechanism of this step in the function code. A top-level
156 script that uses functions in this way gives a better overview of the structure of the analysis. Functions
157 make it easier to reuse code at various points in the analysis, instead of rewriting or copying and editing it.
158 Finally, writing functions allows the programmer to test small parts of the program in isolation from the
159 rest.

Testing is an essential discipline to give some assurance to the authors and other users that the code
161 works as expected. Tests should also document how the code is expected to be used. Inexperienced coders
162 typically greatly underestimate how often they make errors; expecting, finding, and fixing errors is one of
163 the foundations of learning from continued practice. There are tools that can measure what proportion of
164 the lines of code are covered by the tests—this is *test coverage*.

Continuous integration. Tests are useless if they are not run, or if the authors do not see the results.
166 Modern code quality control includes *continuous integration testing*, a practice that guarantees that the
167 tests are run frequently and automatically, and the results are reported back to the coding team. Fortunately,
168 this practice has become so ubiquitous in open source, that there are large free services that implement

---

[4]`https://github.com`
[5]`https://bitbucket.org`
[6]`https://about.gitlab.com`
[7]`http://nipy.org/nibabel`

---

169 continuous integration, such as Travis CI,[8] Circle CI,[9] and Appveyor.[10] These services can integrate with
170 hosting sites like GitHub in order to run tests after each change to the code.

171 **Markup languages.** A markup language allows you to write content in plain text, while also maintaining
172 a set of special syntax to annotate the content with structural information. In this way, the plain text or
173 *source* files are human-readable and work well with standard version control systems. To produce the
174 final document, the source files must undergo a *build* or *render* step where the markup syntax (and any
175 associated style files) are passed to a tool that knows how to interpret and render the content.

176 LaTeX and Markdown represent two extremes of markup languages, each with their own usefulness.
177 LaTeX is notation-heavy, but powerful. In contrast, Markdown is notation-light, but limited.

178 Pandoc is a command line document processor that can convert between multiple markup formats. It can
179 also generate rendered output from source files with text markup. For example, it can generate PDF files
180 from text files written with Markdown markup.

181 **Reproducible workflows.** When working with the Unix command line, we frequently generate files by
182 performing a sequence of commands. The venerable Make system was originally written to automate the
183 process of compiling and linking programs, but is now widely used to automate all types of command line
184 workflows.

185 Makefiles are machine-readable text files consisting of rules specifying the sequence of commands
186 necessary for generating certain files and for tracking dependencies between files. Consider the following
187 Makefile rule:

```
188      progress.pdf: progress.md
189              pandoc -t beamer -s progress.md -o progress.pdf
```

190 This rule defines the procedure to generate the PDF slide show file `progress.pdf`. The first line
191 specifies the *target* of the rule, and any *prerequisites*. Here the target is `progress.pdf`; building
192 this target requires the source Markdown text file `progress.md`. The indented line gives the *recipe*,
193 which is the mechanism by which the target should be generated from the prerequisites. In this case,
194 the recipe is to execute the `pandoc` utility on the Markdown file, with various options applied.[11] If
195 you edit `progress.md`, you can regenerate `progress.pdf` from the command line using `make`
196 `progress.pdf`.

197 In the example above, the recipe was a single command, but Makefile recipes often involve several
198 commands. The target of a rule is often a filename (e.g., `progress.pdf`), but can also be an arbitrary
199 name to describe the sequence of commands in the recipe. The dependencies of a rule can be filenames
200 (e.g., `progress.md`), but may also be other rules in the Makefile. These features allow Makefiles to
201 chain together complex sequences of commands necessary for generating multiple target files, which
202 may depend on other files or steps; such chaining is important for building and maintaining reproducible
203 workflows.

---

[8] https://travis-ci.org
[9] https://circleci.com
[10] https://www.appveyor.com
[11] http://pandoc.org/MANUAL.html#producing-slide-shows-with-pandoc

---

3.1.2 Lectures, labs, notes, homeworks, readings, and quizzes

The weekly lectures and labs prepared students with the skills and background needed for the group project. We demonstrated methods and tools during lecture and lab, and expected students to do additional research and reading as they worked on the group project. For example, we only gave one or two lectures on each of Unix, Git, Python, testing, and NumPy. Each weekly lab focused on the software stack from previous lectures. For example, after the Git lecture, we used Git and GitHub to fetch and submit all exercises; after introducing code testing in lecture, we continually reinforced testing in the labs. Course work outside labs also made heavy use of the toolstack, including each new tool as it was introduced in lectures.

For the first three lectures, we introduced students to the Unix environment and version control using Git. The fourth lecture was a high-level introduction to neuroimaging and functional magnetic resonance imaging (or FMRI). We then introduced students to scientific computing with Python. We referred students to our course notes on Bash,[12] Git,[13] Make,[14] Python,[15] scientific Python packages,[16] and LATEX.[17]

By the ninth lecture (week 5), we started to focus on the analysis of FMRI data with Python. Data analysis lectures continued to develop the students' skills with Unix and the scientific Python software stack through practical problems, which arose as we taught FMRI analysis methods.

We assigned two homeworks. Students had two weeks to work on each homework. Homeworks were assigned and submitted using Git and private GitHub repositories during the first half of the semester. Typically, assignments were given as a collection of tests that tested the desired behavior of named functions, and function templates with missing implementations. The functions were documented according to the NumPy documentation standard.[18] Students added implementations to the functions according to the documentation and ran the tests to check that their code returned the correct results for the provided tests. Assignments were graded using extended tests, which were not provided to the students before they submitted their work. The homework reinforced the material we taught during the beginning of the course and focused on scientific programming in Python.

Over the course of the semester, seven readings were assigned on a roughly bi-weekly basis. The readings consisted of articles that emphasized the core concepts of the class, either with respect to scientific computing, or neuroimaging. Students were required to compose a two-paragraph write-up that both summarized the main points of the article, and commented on it.

The labs and quizzes emphasized hands-on experience with the computing tools and process associated with reproducible and collaborative computing (e.g., version control, LATEX, etc.). Multiple choice quizzes were held at the beginning of lab sessions, and emphasized the computing aspects of the course material as opposed to the statistical and neuroimaging components covered in the lectures. The remainder of the lab was devoted to providing hands-on experience via collaborative work on breakout exercises. These exercises were formulated as small projects to be worked on in groups of three to four students.

---

[12] http://www.jarrodmillman.com/rcsds/standard/bash.html

[13] http://www.jarrodmillman.com/rcsds/standard/git-intro.html

[14] https://www.youtube.com/watch?v=-Cp3jBBHQBE

[15] http://www.jarrodmillman.com/rcsds/standard/python.html

[16] http://www.scipy-lectures.org/

[17] https://www.youtube.com/watch?v=8khoelwmMwo

[18] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

239 We also used the labs to help the students practice critical code review. For example, we had the students
240 form small groups and review one another's solutions to the second homework. This involved all the
241 students in each group creating pull requests to a shared GitHub repository with their solutions. Then
242 they had to use the GitHub code review model, which we had been practicing in class, to review three
243 functions in each solution. We asked them to summarize their findings with respect to clarity, brevity, and
244 performance. We used their summaries to inform a subsequent discussion about code review.

3.2 Course project

246 The course centered around a semester-long group project that accounted for 55% of the overall course
247 grade. Students worked on their projects in teams for three months.

248 Early in the semester (week 5) groups of three to five students formed teams. To make sure every aspect
249 of the project work was computationally reproducible and collaborative, we immediately created a publicly
250 visible project repository for each team. The groups defined the scope of their projects iteratively; the teams
251 submitted their proposals in week 6, a third of the way through the semester. We gave each team feedback
252 to clarify project motivation and goals.

253 For their project proposals, we required the teams to use a LATEX template, which we provided, and to
254 add the source file(s) to their project repository (see Figure 1). Each proposal involved (a) identifying a
255 published FMRI paper and the accompanying data, (b) explaining the basic idea of the paper in a paragraph,
256 (c) confirming that they could download the data and that what they downloaded passed basic sanity checks
257 (e.g., correct number of subjects), and (d) explaining what approach they intended to take for exploring the
258 data and paper. All teams chose a paper and accompanying dataset from OpenFMRI,[19] a publicly-available
259 depository of MRI and EEG datasets (Poldrack et al., 2013; Poldrack and Gorgolewski, 2015).

260 [Figure 1 about here.]

261 We reviewed the project proposals and met with each team in week 7 to help them refine their ideas. The
262 feedback process continued throughout the semester, with students regularly submitting drafts according to
263 milestones defined within the project timeline. As much as possible, we used this process to replicate the
264 way we work with collaborators in the lab.

265 Project feedback included peer-review, where each research team was responsible for cloning another
266 team's project, running the tests and analyses, and evaluating the resulting technical document. The
267 peer-review process proved particularly valuable, as the students benefited greatly from the exposure to the
268 coding techniques and organizational principles of their peers.

269 Students used GitHub's pull request mechanism to review all project work. Code was written as a
270 collection of functions (with tests) and short scripts that called these functions to perform the project data
271 analysis. We taught students to test their code thoroughly. We set up each project with a configuration file
272 for the Travis CI[8] continuous integration service, and enabled integration of this service with their GitHub
273 repository. The configuration file specified that their tests would be run and reported on the Travis CI
274 servers each time they changed their code on the GitHub hosting site. We added automated code coverage
275 testing; each change to the code on GitHub resulted in a measure of the proportion of all code lines covered
276 by tests. We expected students to keep measured test coverage above 90%. While each team decided when

---

[19]https://www.openfmri.org/

277 to merge pull requests, we recommended that all pull requests should have at least three participants, test
278 coverage should not decrease, and tests should pass before being merged.

279 Students also used the GitHub issues interface to record project discussions and actions taken by the
280 group members. We told students to create a pull request or issue with code and text and use the GitHub
281 interface to ask one or more of us to respond, if they had any questions about their projects. Periodically,
282 we would review their work and open issues.

283 We covered this workflow in depth during lectures and in lab, and made clear that the project grade
284 would be based on (a) the final report, (b) the analysis code, (c) whether we could run their analysis code to
285 reproduce all their results and figures, and (d) how effectively the team collaborated using the techniques
286 taught in the course (e.g., pull requests, code review, testing). We told the students that we would use the
287 pull request and issue discussions as evidence for their contributions to the project, and as data for their
288 final grades.

289 Rather than have students commit the raw FMRI data to their project repository, we had them commit
290 code to download and validate the raw data used in their project (week 8). Since this was the first code
291 they added to their repositories, we gave them example code to download a file and validate it by checking
292 the file against a stored checksum of the file content. Once they added code to handle downloading and
293 validating the data, we had them commit code for their exploratory data analysis (week 9). After this, the
294 only milestones we provided were for progress presentations and report drafts.

295   [Figure 2 about here.]

296 The students gave their first progress report and presentation in week 12; it consisted of a description of
297 the data they were using, what they had done so far, their plan for completing the project, and a reflection
298 on the overall process (see Figure 2). We required the groups to write their slides in Markdown text
299 format for building into PDF with Pandoc. They committed the Markdown source to the project repository.
300 The students gave their second slide presentation in week 15, two weeks before the projects were due,
301 discussing their current progress and their plan for finishing their project. At the end of the project, in
302 week 17, they had to commit the final PDF of their reports and provide instructions for generating the
303 report PDF from the source LATEX files.

304 Early in the project, we told the students that we would grade their project work on whether it was
305 reproducible. In order to reproduce their work, we told the students we would blindly follow the instructions
306 in a text file named README.md in the root directory of the project repository. When we graded their
307 projects, the README.md file had to explain how to install, configure, and run all aspects of their project.
308 Each README.md had to specify how to rebuild all the components of the project and in what order to do
309 so. For example, it might include a section specifying the make commands to execute, such as:

```
310      make data       # download the data
311      make validate   # ensure the data is not corrupted
312      make eda        # generate figures from exploratory analysis
313      make analysis   # generate figures and results
314      make report     # build final report
```

3.3 Neuroimaging data analysis

In terms of neuroimaging, our aims were for students to (a) understand the basic concepts in neuroimaging, and how they relate to the wider world of statistics, engineering, and computer science; (b) be comfortable working with neuroimaging data and code, so they could write their own basic algorithms, and understand other people's code; (c) continue to use the computational techniques we had taught them to improve efficiency and help their understanding.

For this course we concentrated on the statistical analysis of FMRI data using a linear model. We designed each lecture to teach the next analysis step the students would need for their project work.

All the following teaching used simple Python code to show them how the mathematics works in practice, and to give them sample code that they could edit and run for themselves.[20] We specifically avoided using off-the-shelf imaging analysis software packages, and encouraged the students to build their own analyses from the building blocks we had given them. In lectures, we interleaved teaching with short group exercises where students took the code from class and extended it to solve a new problem.

We covered the following topics:

- The idea of images as visual displays of values in arrays.
- The standard neuroimaging NIfTI image format as a simple image container, combining metadata (such as shape) and image pixel / voxel data stored as a sequence of numbers. We used the Nibabel Python package [7] to load NIfTI data as three-dimensional arrays.
- Four dimensional (4D) images as sequential time-series of 3D image volumes. Slicing arrays to get individual volumes from the 4D time-series. Extracting single-voxel time-courses from 4D images.
- Building regressors for the statistical model. We introduced the idea of the neuronal time-course model as the hypothesized change in neuronal activity caused by the task. For a block design, this model becomes a box-car, with all rest periods having identical low activity, and all activity periods having identical high activity. The FMRI acquisition measures something like blood flow, rather than neuronal activity, but blood flow changes slowly in response to changes in neuronal activity. We introduced the linear-time-invariant assumption in predicting the hemodynamic activity from the hypothesized neuronal activity, and then the hemodynamic response function as the invariant response. Finally we demonstrated the mechanism of convolution as a numerical method to express the effect of the hemodynamic response function acting in a linear-time-invariant way, and the caveats to these assumptions. We now had hemodynamic regressors to fit to our voxel time-course data.
- Correlation as a simple test of linear association between the hemodynamic regressor and voxel signal. Calculating correlation with a single regressor for every brain voxel and the idea of a statistical parametric map.
- The linear model as a matrix formulation of multiple regression. We started with simple (single-regressor) regression as another measure of linear association. We expressed simple regression as a linear model and showed how the model can be expressed as the addition of vectors and vector / scalar multiplication. This leads to the matrix formulation of simple regression, and thence to multiple regression. We introduce dummy indicator variables to express group membership and show how these relate to group means. We showed with code how this mathematics can express statistical methods that they already know, such as regression, t-tests, and ANOVA.

---

[20]For an example lesson showing our use of Python code to illustrate the mathematical and statistical ideas, see `http://www.jarrodmillman.com/rcsds/lectures/glm_intro.html`.

- High-resolution sampling of regressors. The simple cases that we had covered above assumed events or blocks that started at the same time as the scanner started to collect a new brain volume. This is not the case in general. To deal with events that do not start at volume onsets, we need to generate a hemodynamic time-course at higher time resolution than the spacing of the scan onsets, and then sample this regressor (using linear interpolation) at the scan onset times.

- Parametric modulation of event regressors. Some of the OpenFMRI datasets that the students had chosen used regressors to model parametric modulation of events. There is one regressor to model the hemodynamic effect of a particular event type on average, and another to capture the variation of the hemodynamic event activity as a linear function of some characteristic of that event, such as duration or intensity.

- Spatial smoothing with a Gaussian kernel; smoothing as a form of convolution; the `scipy.ndimage` subpackage in SciPy as an implementation of Gaussian and other smoothing.

- The idea of voxel and millimeter coordinates in brain images, and the image affine as a mapping between them. The students needed this in order to relate the coordinates reported in their papers to voxels in their own analyses.

We reinforced the material from class lectures and exercises in the homeworks. Figure 3 shows an excerpt from the second homework, in which we asked to students to fill out functions implementing various algorithms on 4D FMRI data and use these functions to build up a simple diagnostic test for outlier volumes in the time series. Finally, they applied a linear (multiple regression) model to the data to show that removing outlier scans reduced residual variance from the model fit.

[Figure 3 about here.]

We encouraged students to use the basic Python building blocks for their project analyses, but we did not insist. Our rule was that, if they used other software, they had to persuade us that they had a sound understanding of the algorithms that the other software was using.

3.4 Project grading

We based the evaluation of the final projects on criteria that emphasized the underlying principles of reproducibility, collaboration, and technical quality. See Table 1 for the final project grading rubric, which we gave to the students on week 11—the week before the first progress presentations. From the perspective of reproducibility, we evaluated projects on whether the presented results could be generated from their repositories according to the documentation they provided in their `README.md` text file. When we could not reproduce the analysis, we raised one or more GitHub issues to negotiate with the project team. We graded the code tests with respect to code coverage. Grading for the collaborative aspects of the project used the information from the project history provided by the Git version history and GitHub web artifacts, including code contributions, as well as reviewing GitHub pull requests and discussion on GitHub issues. Finally, we assessed the technical quality of the project in terms of the clarity of the final report and with respect to how well the proposed goals of the study were met by the final results of the analysis.

[Table 1 about here.]

4 RESULTS

There were a total of eleven research teams composed of three to five students, each responsible for completing a final project of their own design using datasets available through the OpenFMRI organization. We named groups arbitrarily with Greek letters (e.g., alpha, kappa, zeta). The project repositories are public at `https://github.com/berkeley-stat159`.

Although we allowed students to select any OpenFMRI dataset, in fact all groups selected one of the following:

- `ds000005`:[21] *Mixed-gambles task* (Tom et al., 2007) (groups delta, epsilon, eta, theta);
- `ds000009`:[22] *The generality of self-control* (Cohen and Poldrack, 2014) (group alpha);
- `ds000105`:[23] *Visual object recognition* (Haxby et al., 2001) (groups kappa, zeta);
- `ds000113`:[24] *A high-resolution 7-Tesla FMRI dataset from complex natural stimulation with an audio movie* (Hanke et al., 2014) (groups beta, lambda);
- `ds000115`:[25] *Working memory in healthy and schizophrenic individuals* (Repovs and Barch, 2012) (groups gamma, iota).

During grading, we succeeded in fully replicating all group analyses. The minor problems that arose were largely platform differences between macOS used by the students and the Linux system we were using for grading. Four projects reproduced without issue, six required us to raise one GitHub issue, and one required two issues.

We counted the lines of Python code using the `cloc` utility.[26] The median lines of code across projects was 1,494 (range 624–8,582), of which tests comprised (median; range) 440; 36–592. As this suggests, we were not successful in persuading the students to test a large proportion of their code. We estimated coverage of all lines of Python code at a median of 12.4% (range 2.0–91.8).[27] The code coverage machinery we had put in place did not measure coverage of code outside expected code directories, and the students had put a large proportion of their code outside these directories, leaving the automated code coverage score above 90% in all cases, even though a large proportion of the code was not in fact covered by tests.

No team used SPM, FSL, or AFNI. Two projects imported the `nilearn` package,[28] which is a Python neuroimaging package designed for machine learning, but both projects used Nilearn only for spatial smoothing and basic volume visualization. One project used a single function from the `Dipy` package[29] for calculating a within-brain mask for the functional images. Another used a class from the `NiTime` package[30] for temporal filtering. All projects used the basic NumPy, SciPy, and Matplotlib packages; eight of eleven used the `Scikit-Learn` package[31] for machine learning; six used the `Statsmodels` package[32] for

---

[21]`https://openfmri.org/dataset/ds000005`

[22]`https://openfmri.org/dataset/ds000009`

[23]`https://openfmri.org/dataset/ds000105`

[24]`https://openfmri.org/dataset/ds000113`

[25]`https://openfmri.org/dataset/ds000115`

[26]`http://cloc.sourceforge.net`

[27]See `https://github.com/jarrodmillman/rcsds-paper/tree/master/project_metrics` for details of this estimate.

[28]`https://nilearn.github.io`

[29]Dipy is a package for analysis of diffusion imaging data: `http://nipy.org/dipy`

[30]NiTime is a library for time-series analysis of data from neuroscience experiments: `http://nipy.org/nitime`

[31]`http://scikit-learn.org`

[32]`http://www.statsmodels.org`

---

422 statistical modeling. The uses of Statsmodels were: AutoRegressive Integrated Moving Average (ARIMA)
423 modeling of voxel time-series and autocorrelation plots (two projects); logistic regression (two projects);
424 ordinary least squares modeling (one project); White test for Heteroscedasticity (one project); mixed effects
425 modeling (one project). Of these, we had only shown the implementation of ordinary least squares analysis
426 in class.

427 There was a wide range in the novelty and scope of the projects. Most projects consisted of a serious
428 attempt to replicate one or several statistical findings in the original paper, with the addition of some further
429 extension or exploration. These extensions were typically the application of other statistical techniques. For
430 example, project epsilon was one of the four groups working on the dataset ds000005—the *Mixed-gambles*
431 *task*. They first explored the imaging data using the outlier detection machinery that all students had
432 developed in the second homework. Next they explored various logistic regression models of the behavioral
433 data. For the imaging data, they used the SciPy `ndimage` subpackage to smooth the data, as we had
434 briefly shown in class, and then followed some hints in the lectures to explore different confound models
435 such as linear and quadratic drift, and Principal Components as regressors. They used code from class to
436 implement the general linear model at each voxel, and calculate t- and p-values using contrast vectors.
437 Finally they thresholded their images using Bonferroni correction.

438 Other projects did more substantial technical or intellectual extensions of the original analysis. Project
439 alpha explored an analysis from ds000009—*The generality of self-control*. Although we had not covered
440 this in class, they discovered from data exploration and reflection that the times of slice acquisition
441 would affect their statistical modeling, and developed code to shift their model back and forth in time
442 corresponding to the time of slice acquisition. They tried various confound models on the data, including
443 linear drift, Fourier bases and different numbers of PCA regressors, and explored these models with
444 selection criteria such as the Bayes and Akaike Information Criteria, and adjusted R^2. They ran diagnostics
445 to detect voxels violating assumptions of normality. For whole brain analysis, they implemented the FDR
446 multiple comparison correction and tried other methods for identifying activated brain regions, including
447 hierarchical clustering. Finally, they experimented with ARIMA models of the voxel time course.

448 Project lambda did an heroic effort to replicate the analysis of ds000113—a high-resolution FMRI dataset
449 of subjects listening to a description of the film Forrest Gump. This dataset had many technical challenges;
450 the images are unusually large, at over 1 million voxels per volume, and of unusually long duration, at
451 around 450 volumes per run, and having 8 runs per subject. The analysis requires the correlation of many
452 voxel time-courses in each subject with voxel time-courses in all other subjects in the analysis. The group
453 did a variety of explorations, including finding some artifacts in the original data, and an error in the
454 published paper, and then went on to replicate the original correlation analysis, on a smaller number of
455 subjects. They ran the analysis on Amazon Web Services machines in order to deal with the demands
456 of processing time and memory. As they note in their `README.md` "We strongly encourage running on
457 a machine with 120 GBs of accessible RAM to emulate development environment." They extended the
458 analysis by running a random forest model to detect volumes corresponding to outdoor and indoor scenes
459 in the film. With all this, they achieved around 92% code test coverage.

460 Seventeen out of forty undergraduates and six out of ten graduate students completed anonymous course
461 evaluations. The primary question of interest to us was "Considering both the limitations and possibilities
462 of the subject matter and the course, how would you rate the overall effectiveness of this course?". Ratings
463 were on a 1 through 7 scale with 4 corresponding to "moderately effective" and 7 corresponding to
464 "extremely effective." The average undergraduate and graduate scores were 4.71 and 6.0 respectively,
465 against a department average across all courses of 5.23. Undergraduate, graduate, and department average

ratings for "I enjoyed this class" on a 1 through 7 scale (with 4 corresponding to "somewhat" and 7 to "very") were 4.69, 6.00, and 4.96 respectively.

70.5% of the undergraduate respondents and all the graduate respondents claimed to have worked 10 or more hours a week on average.

5 DISCUSSION

Most neuroimaging researchers agree that computational reproducibility is desirable, but rare. How should we adapt our teaching to make reproducibility more common?

The usual practical answer to this question, is that we should train neuroimaging and other computational researchers as we do now, but add short courses or bootcamps that teach a subset of the tools we taught here but without the substantial practice. That is, reproducibility is an addition on top of current training.

We believe this approach is doomed to failure, because the students are building on an insecure foundation. Once you have learned to work in an informal and undisciplined way, it is difficult to switch to a process that initially demands much more effort and thought. Rigor and clarity are hard to retrofit. To quote the American chemist Frank Westheimer: "A couple of months in the laboratory can frequently save a couple of hours in the library."

For these reasons, our course took the opposite approach. We put a substantial, collaborative, and open-ended project at the center of the course. We then started with the tools and process for working with numerical data, and for building their own analyses. We used this framework as a foundation on which to build their understanding of the underlying ideas. As we taught these tools, we integrated them into their exercises and homework, and made it clear how they related to their project work.

Our claim is that this made our teaching and our students much more efficient. The secure foundation made it easier for them to work with us and with each other. As they started their project work, early in the course, they could already see the value of these tools for clarity and collaboration. Our students graduated from the course with significant experience of using the same tools that experts use for sound and reproducible research.

5.1 Did we really teach neuroimaging analysis?

By design, our course covered tools and process as well as neuroimaging. We used neuroimaging as the example scientific application of these tools. Can we claim to have taught a meaningful amount of neuroimaging in this class?

Class content specific to neuroimaging was a guest lecture in class 4 (of 25), and teaching on the brain images, correlation, and the general linear model from classes 9 through 15. We covered only the standard statistical techniques needed for a basic analysis of a single run of FMRI data. This is a much narrower range than a standard neuroimaging course, but we covered these topics in much greater depth than is typical for a neuroimaging course. This was the basis for final projects that were substantial and well-grounded.

Typical imaging courses do not attempt to teach the fundamental ideas of linear models, but assume this understanding and move onto imaging specifics. This assumption is almost entirely false for neuroscience and psychology students, and largely false for our own students, even though most had training from an undergraduate statistics major. As a result of this incorrect assumption, it is rare for students of neuroimaging to be confident in their understanding of the statistics they are using. We taught the linear model from the first principles of vector algebra, using the tools they had just learned, to build a simple

analysis from basic components. As a result, when the students got to their projects, they had the tools they needed to build their own neuroimaging analysis code, demonstrating and advancing their own understanding.

We note the difficulty of the task that we gave the students, and the extent of their success. We made clear that their project was an open-ended exploration of an FMRI dataset and paper. Few students had any experience or knowledge of FMRI before the course. The only guidance we gave was that they should prefer well-curated datasets from the OpenFMRI depository. In order to design and implement their project, they had to understand at least one published FMRI paper in some depth, with limited assistance from their instructors. We gave no example projects for them to review, or templates for them to follow. The submitted projects were all serious efforts to reproduce and / or extend the published results, and all included analysis code that they had written themselves.

We did not teach the full range of neuroimaging analysis. For example, we did not cover pre-processing of the data, random effects analysis, or inference with multiple comparisons. Our claim is that what we did teach was a sound and sufficient foundation from which they could write code to implement their own analyses. This is a level of competence that few neuroimagers achieve after considerable experience.

5.2 Did we really teach computational reproducibility?

We stressed the importance of reproducibility throughout the course, and made it an explicit feature for grading of the final project, so we were not surprised to find that it was possible to reproduce all of the final projects with little extra clarification or fixes from the group members.

By design, we gave the students a project that was close to a real scientific data analysis. They had to work with a messy and real dataset to explore the data, define their problem and solve it with an analysis that they implemented themselves, to various degrees. We required them to work closely in teams, often remotely, using standard tools for collaboration.

We found this combination of a substantial analysis problem, expert tools, and the requirement for reproducibility, was effective at giving the students a concrete sense of the difficulties in making an analysis reproducible, and how these can be overcome. We speculate, from our own work, that the experience of building a reproducible analysis makes it easier to commit to the tools and practice needed for reproducible work in the future.

5.3 Did we teach the right tools?

We do not believe that the individual tools we chose were controversial. We taught the tools that we use ourselves, and that we would teach to students working with us.

We could have used R instead of Python, but Python is a better language for teaching, and has better libraries for neuroimaging and machine learning.

A more interesting question is whether we went too far in forcing the students to use expert tools. For example, we required them to write their report in LATEX, and their presentation slides and analysis description in plain text with Markdown markup. We asked them to do all project interaction using GitHub tools such as the issue tracker and pull requests. We set and marked code exercises with Git version control and text files, rather than interactive alternatives, such as Jupyter Notebooks.

Of course—at first—some students complained that they would rather use Facebook for code discussions and PowerPoint for their presentation slides. Should we have allowed this? We believe not. Our own

545 experience of using these tools is that their power only becomes apparent when you learn to use them for
546 all the tasks of analysis, including generating reports and presentations. Mixing "easy" but heavy tools
547 like PowerPoint and Facebook with "simple" and light tools like text editors and Markdown causes us
548 to lose concentration as we switch modes from heavy to light and back again. It is easy to be put off
549 by the difficulty of getting used to the light tools, and therefore fail to notice the clarity of thought and
550 transparency of process that they quickly bring. Successfully switching from heavy to light is a process
551 that requires patience and support; it is best done in the context of a class where there are examples of use
552 from coursework and support from experienced instructors who use these tools in their daily work.

5.4   Do students need to be trained in programming?

554 One common justification for not training students in good coding practice is the assertion that scientists
555 do not need to be programmers. However, scientists do have to use, read, and write code, so a more
556 defensible statement would be that scientists do not need to be *good* programmers. It is surely true that
557 successful scientists can be bad programmers, but bad programmers are inefficient and prone to error; they
558 are less likely to detect errors, and improve very slowly over time. We should invest teaching time to help
559 our students work efficiently and continue learning for the rest of their careers.

5.5   Do students need a substantial, collaborative, and open-ended project?

561 We put great emphasis on the final project in this course, and this was clear to most of the students.
562 In response to the evaluation survey question *"What advice would you give to another student who is*
563 *considering taking this course?"* one undergraduate wrote:

564 "[U]nlike most group projects (which last for maybe a few weeks tops or could conceivably be pulled
565 off by one very dedicated person), this one will dominate the entire semester. . . . Try to stay organized
566 for the project and create lots of little goals and checkpoints. You should always be working on
567 something for the project, whether that's coding, reviewing, writing, etc. Ask lots of questions and ask
568 them early!"

569 The size of the project meant that the students had to learn to collaborate with each other efficiently,
570 often remotely, as students had different class schedules. Many of the tools that we taught, such as
571 distributed version control, only become essential when working in collaboration. Conversely, if you are
572 not collaborating with others, it can be difficult to see why it is worth investing the time to understand
573 powerful tools like the Git version control software, or the GitHub interface.

574 Working in collaboration, and working reproducibly, changes the way that we think. If we have to explain
575 our work to others in the group, or to another user of the work, then we develop the expectation that
576 whatever we write will always be something we will demonstrate and explain to others. It becomes part of
577 the work to communicate our ideas and explain what we did.

578 It was important that the project was open-ended. If the student has to solve a small problem with a single
579 correct answer, they can often check whether they have the right answer, and do not need to worry about
580 the quality of the process that generated it. In an open-ended project, it is likely that the group will need to
581 explore different analyses as they progress. The answers are not known, and the group has to proceed with
582 care, to avoid making false conclusions. This is typical of real scientific analysis, and puts a higher burden
583 on rigor and testing, than a typical small classroom problem.

584 We should emphasize how hard it was to get the students to engage with the project early, and work
585 steadily. In the first class, we gave the students a document describing the project, including the various
586 project deadlines. We continued to emphasize the project and project deadlines in announcements.
587 Nevertheless, it was only half way through the course that the students began to realize how open-ended
588 their task was, and how much work they would have to do. For a few weeks, the class was anxious, and our
589 job as instructors was to project faith in the teams' ability to define a tractable and interesting question. We
590 mention this to say that, a large open-ended project has many advantages, but to make it work, it does take
591 courage from the students and the instructors.

5.6 What background do students need?

593 The requirements for our course were previous classes on probability, statistics, and the use of the R
594 programming language.

595 We would be happy to relax the requirement for probability. We did not refer to the ideas of probability
596 in any depth during the course. Authors JBP and MB have taught similar material to neuroscience and
597 psychology students who lack training in probability; the pace of teaching and level of understanding were
598 similar for the statistics students in this course and the other students we have taught.

599 Psychology and neuroscience students do have some statistical training. Our impression was that this
600 background was necessary for us to be able to as start as quickly as we did in the analysis of linear
601 regression.

602 We would also keep the requirement for some programming experience. We assumed familiarity with
603 programming ideas such as for loops, conditionals, and functions. In our psychology / neuroscience courses
604 that used similar material, we required some experience of programming in a language such as Python, R,
605 or MATLAB. It is possible that a brief introduction would be enough to fulfill this requirement, such as a
606 bootcamp.

5.7 Could this material be covered by a bootcamp or hack week?

608 There are several existing programs designed to address the problems of informal and inefficient
609 computational practice. The best known may be Software Carpentry (Wilson, 2014). A Software Carpentry
610 course consists of a two-day bootcamp covering many of the same tools we used, including the Unix
611 command line, distributed version control, and programming with a high-level language such as Python or
612 R. Hack weeks can be a related approach to the same problems. Hack weeks vary in format from a series
613 of tutorials to a week of collaborative work on a project that has already been agreed (Huppenkothen et al.,
614 2017). The tutorials and modeled working practice usually include the same set of tools we used.

615 These approaches have some similarity with the early classes in our course, but there are differences.
616 Bootcamps and hack weeks attract volunteers with interest in or commitment to efficient and reproducible
617 practice. These are typically graduate students or post-docs that already have some technical experience.
618 They are motivated enough to come to campus on a weekend, stay longer at a conference, or travel some
619 distance. Even so, we suspect that it would be possible to show that short introductions in bootcamps will
620 not be effective in changing practice in the medium term, without later reinforcement and support by peers.

621 Our students did choose our course from others they could have taken, but most of them had little
622 background of good practice in computation. The majority were undergraduates. We suspect that many of
623 the students finishing our course did have enough experience of using the tools they had used, to continue

624 using them in their daily work, and teach others to do the same. We base this suspicion on the depth and
625 quality of the project work.

626   Bootcamps and hack weeks can be useful, but their starting point is an attempt to augment an aging
627 curriculum that does not recognize the need for training in accurate, effective, and reproducible computation.
628 We should fix this by changing our curriculum; learning to work this way takes time, practice, and support,
629 and we teach it best with substantial commitment from students and instructors.

630 ## 5.8   Where would such a course fit in a curriculum?

631   Our course would not fully qualify a student for independent research in neuroimaging. As we discuss
632 above, we did not cover important aspects of imaging, including spatial and temporal pre-processing of
633 data, random effects, or control of false positives in the presence of multiple comparisons. Where should
634 the elements of our course sit in relation to a larger program for training imagers and other computational
635 scientists?

636   We think of this course as a foundation. Students graduating from this course will be more effective in
637 learning and analysis. The tools that they have learned allow the instructor to build analyses from basic
638 blocks, to show how algorithms work, and make them simple to extend. We suggest that a course like ours
639 should be first in a sequence, where the following courses would continue to use these tools for exposition
640 and for project work.

641   A full course on brain imaging might start with an introduction to Python and data analysis, possibly as a
642 week-long bootcamp at the start of the semester. Something like our course would follow. There should
643 be follow-up courses using the same tools for more advanced topics such as machine learning, spatial
644 pre-processing, and analysis of other modalities. We suggest that each of these courses should have group
645 project work as a large component, in which the students continue to practice techniques for efficient
646 reproducible research.

647 ## 5.9   What factors would influence the success of such a course?

648   We should note that there were factors in the relative success of this course that may not apply in other
649 institutions.

650   Berkeley has as a strong tradition in statistical computing and reproducibility. Leo Breiman was a
651 professor in the Berkeley statistics department, and an early advocate of what we would now call data
652 science. He put a heavy emphasis on computational teaching to statisticians. Sandrine Dudoit is a professor
653 in the statistics and biostatistics departments, and a founding core developer of the Bioconductor project
654 devoted to reproducible genomics research in R. The then head of the statistics department, Philip Stark,
655 is one of many Berkeley authors (including KJM) to contribute to the recent book "The Practice of
656 Reproducible Research" (Kitzes et al., 2017). Authors KJM, MB, and JBP have worked with Mark
657 D'Esposito, who runs the Berkeley Brain Imaging Center, and was an early advocate for data sharing in
658 FMRI. Other imaging labs on campus take this issue seriously and transmit this to their students. If there
659 had not been such local interest in the problem of reproducibility, students may have been less convinced
660 of the importance of working in a reproducible way, especially given the short-term convenience of a less
661 disciplined working process.

662   Students of statistics and other disciplines at Berkeley are well aware of the importance of Python in
663 scientific computing, and in industry. Tech firms recruit aggressively on campus, and Python is a valuable

664 skill in industry. The cross-discipline introductory course in Fundamentals of Data Science uses Python.
665 The new Berkeley Institute of Data Science has a strong emphasis on Python for scientific computing.

666   In the same way, a booming tech sector nearby made it more obvious to students that they would need to
667 learn tools like Git and GitHub. For example, public activity on GitHub is one feature that companies use
668 to identify candidates for well-paid positions as software engineers.

669   We required students to use LaTeX to write their final reports. This was an easy sell to statistics students,
670 but students outside mathematical fields might be less agreeable; if we were teaching psychology and
671 neuroscience students, we might choose another plain text markup language, such as Markdown, with the
672 text files run through the Pandoc document processor to write publication quality output.

CONCLUSION

673 Our course differs from other imaging courses that we know of, in several respects. We started with correct
674 computational practice, before teaching neuroimaging. We emphasized computation as a way of explaining
675 the underlying ideas and taught the fundamentals of the linear model, rather than assuming students had
676 this background, or would get this background later. We made these ideas concrete with a substantial
677 open-ended project designed to be as close as possible to the experience of graduate research.

678   We were largely successful in teaching the students the tools they can continue to use productively for
679 collaborative and reproducible research.

680   We are sure that most of our readers would agree that, in an ideal world, we should teach students to work
681 in this way, but, given all the other classes our students must take, can we justify the time and energy that
682 this course needs? We believe so, but we know that not all our readers will be convinced. This is true of
683 students as well as instructors. We put such emphasis on the final project, precisely because we know that it
684 is difficult to see how important these tools are, if you have not used them, or used them only in toy projects.
685 We cannot easily teach these tools to our fellow teachers, so instead we offer this sketch of an experiment,
686 to make the discussion more concrete. Imagine we took 200 students, and randomized them into two
687 groups of 100 each. The first group takes something like the course we describe here as an introduction,
688 and then one or more further courses on imaging. The second does not take such a course, but instead has
689 more traditional teaching on imaging that covers a wider range of techniques, using off the shelf imaging
690 software. Part of their later teaching would include some techniques for reproducible research. Two years
691 after such an experiment, we predict that students from the first group will have a greater understanding of
692 what they are doing, will be more effective in analysis, and more likely to experiment with new ideas. We
693 think it much more likely that the first group will be doing reproducible research.

CONFLICT OF INTEREST STATEMENT

694 The authors declare that the research was conducted in the absence of any commercial or financial
695 relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

696 KJM was the lead instructor and was responsible for the syllabus and project timeline; KJM and MB
697 were responsible for lectures and created homework assignments; KJM and RB were responsible for
698 labs, readings, quizzes, and grading; all authors held weekly office hours to assist students with their

projects. KJM and MB wrote the first draft of the manuscript; all authors wrote sections of the manuscript, contributed to manuscript revision, as well as read and approved the submitted version.

FUNDING

ACKNOWLEDGMENTS

REFERENCES

Buckheit, J. B. and Donoho, D. L. (1995). Wavelab and reproducible research. In *Wavelets and statistics* (Springer). 55–81

[Dataset] Cohen, J. R. and Poldrack, R. A. (2014). Materials and methods for openfmri ds009: The generality of self control. `https://openfmri.org/media/ds000009/ds009_methods_0_CchSZHn.pdf`. Accessed March 3, 2018

Hanke, M., Baumgartner, F. J., Ibe, P., Kaule, F. R., Pollmann, S., Speck, O., et al. (2014). A high-resolution 7-tesla fmri dataset from complex natural stimulation with an audio movie. *Scientific data* 1, 140003

Haxby, J. V., Gobbini, M. I., Furey, M. L., Ishai, A., Schouten, J. L., and Pietrini, P. (2001). Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science* 293, 2425–2430

Huppenkothen, D., Arendt, A., Hogg, D. W., Ram, K., VanderPlas, J., and Rokem, A. (2017). Hack Weeks as a model for Data Science Education and Collaboration. *arXiv:1711.00028 [astro-ph, physics:physics]* ArXiv: 1711.00028

Kitzes, J., Turek, D., and Deniz, F. (eds.) (2017). *The practice of reproducible research: case studies and lessons from the data-intensive sciences* (University of California Press)

Millman, K. J. and Aivazis, M. (2011). Python for scientists and engineers. *Computing in Science & Engineering* 13, 9–12

Millman, K. J. and Brett, M. (2007). Analysis of functional Magnetic Resonance Imaging in Python. *Computing in Science & Engineering* 9, 52–55

Millman, K. J. and Pérez, F. (2014). Developing open-source scientific practice. *Implementing Reproducible Research. CRC Press, Boca Raton, FL* , 149–183

Pérez, F., Granger, B. E., and Hunter, J. D. (2011). Python: an ecosystem for scientific computing. *Computing in Science & Engineering* 13, 13–21

Poldrack, R. A., Barch, D. M., Mitchell, J. P., Wager, T. D., Wagner, A. D., Devlin, J. T., et al. (2013). Toward open sharing of task-based fMRI data: the OpenfMRI project. *Frontiers in Neuroinformatics* 7

736 Poldrack, R. A. and Gorgolewski, K. J. (2015). OpenfMRI: open sharing of task fMRI data. *NeuroImage*

737 Preeyanon, L., Pyrkosz, A. B., and Brown, C. T. (2014). Reproducible bioinformatics research for
738   biologists. *Implementing Reproducible Research* , 185–216

739 Repovs, G. and Barch, D. M. (2012). Working memory related brain network connectivity in individuals
740   with schizophrenia and their siblings. *Frontiers in human neuroscience* 6, 137

741 Tom, S. M. et al. (2007). The neural basis of loss aversion in decision-making under risk. *Science* 315,
742   515–518

743 Wilson, G. (2014). Software carpentry: lessons learned. *F1000Research* 3

744 Wilson, G., Aruliah, D., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., et al. (2014). Best practices
745   for scientific computing. *PLoS Biol* 12, e1001745. doi:10.1371/journal.pbio.1001745

```
/
└─ proposal
   └─ Makefile
   └─ proposal.bib
   └─ proposal.tex
└─ code
└─ data
   └─ __init__.py
   └─ data.py
   └─ hashes.json
   └─ test_data.py
└─ report
   └─ Makefile
   └─ report.bib
   └─ report.tex
└─ slides
   └─ Makefile
   └─ final.md
   └─ progress.md
└─ .coveragerc
└─ .travis.yml
└─ requirements.txt
└─ LICENSE
└─ Makefile
└─ README.md
```

```
/
└─ proposal
   └─ Makefile
   └─ proposal.bib
   └─ proposal.tex
└─ LICENSE
└─ README.md
```

(1a) Initial repository template.
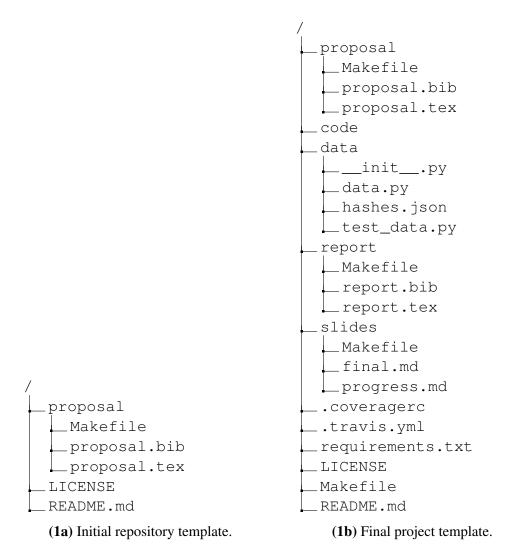
(1b) Final project template.

Figure 1. The initial and final repository directory template for student projects. We gave the students a project copied from the initial template in week 5, from which they would write and build and their initial proposal. As the course progressed, we pushed two updates to their repository. The first gave them a template for building their final report, their top-level `README.md` file and file specifying an open license for their work. The second gave them: an initial set of functions we had shown them in class with associated tests (in the `code` directory, not shown); machinery to trigger automatic execution of tests on Travis-CI servers; automated code coverage reporting; and a template for writing slides for their first progress report.

```
Progress reports and presentations
----------------------------------

1. Briefly describe data
   - Paper
   - dsnum from openfmri.org
   - What kind of data is it?

2. Briefly describe what you've done so far
   - data fetching/preprocessing
   - initial analysis
   - plots? figures?

3. What is your plan?
   - What analysis will you perform?
   - How does it relate to the original data analysis?
     - Will you use all the data?  Why or why not?
     - What model/preprocessing steps will you simplify?
   - What are the problems you face?
     - Try to be explicit about your issues?
     - Suggest potential solutions and/or approaches.
     - What do you need to research more?  Have you found sources?
     - Will you try to make "inferences" from the data?
       - How will you deal with multiple comparisons?
       - How will you attempt to validate your model?

4. Process
   - What was the hardest part of the process so far?
     - Git workflow, Python, fMRI data, all of the above
     - Having an ill-defined assignment?
   - How successful have you been at overcoming these obstacles?
   - What issues have you faced working as a team?  How have you
     been addressing them?
   - What parts of the class have been the most useful?
   - What parts have been the least helpful or most confusing?
   - What do you need to do to successfully complete the project?
   - How difficult are you finding it to make your work reproducible?
     Do you feel confident that you are in a position to make your
     projects reproducible?
   - What would be most helpful for your team in the remaining weeks?
     Additional lectures?  Structured or unstructured group work?
   - What potential topics would be most useful for us to cover?
     - Overview of brain / neuroanatomy?
     - More linear regression (ANOVA)? PCA? The mathematics or the implementation?
     - Machine learning (classification, prediction, cross-validation)?
     - Permutation tests (and maybe bootstrap)?
     - Software tools (Git, Make, Python, statmodels, etc.)
     - Technical writing and scientific visualization?
     - Advanced topics (regularized regression, selective inference)?
```

Figure 2. Class lecture material giving instructions for the first progress report. This is the text we showed them during class, and discussed, to prepare them for their first progress report. As for almost all our lecture material, we posted this text to the class website.

```
""" Script to run diagnostic analysis on FMRI run

The FMRI 'run' is a continuous collection of one or more 3D volumes.
A run is usually stored as a 4D NIfTI image.
Fill in the code necessary under the comments below.
"""


# import standard libraries here


"""
* Load "ds114_sub009.nii" as an image object
* Load the image data from the image
* Drop the first four volumes, as we know these are outliers
"""


"""
Use your vol_std function to get the volume standard deviation values
for the remaining 169 volumes. Save these values to a text file
called 'vol_std_values.txt'.
"""


"""
Use the iqr_outlier detection routine to get indices of outlier volumes.
Save these indices to a text file called 'vol_std_outliers.txt'.
"""


"""
Plot all these on the same plot:
* The volume standard deviation values;
* The outlier points from the std values marked with an 'o' marker;
* A horizontal dashed line at the lower IRQ threshold;
* A horizontal dashed line at the higher IRQ threshold;

Save the figure to the current directory as ''vol_std.png''.
"""


"""
Next calculate and plot the RMS difference values:
* Calculate the RMS difference values for the image data;
* Use the ''iqr_outlier'' function to return indices of possible
  outliers in this RMS difference vector;

On the same plot, plot the following:
* The RMS vector;
* The identified outlier points marked with an 'o' marker;
* A horizontal dashed line at the lower IRQ threshold;
* A horizontal dashed line at the higher IRQ threshold;

Save this plot as ''vol_rms_outliers.png''
"""
```

Figure 3. Above we list the first few lines of `diagnosis_script.py`. This file was part of the second homework, which focused on detecting outlier 3D volumes in a 4D FMRI image. Tasks included: (a) implementing functions on image arrays using NumPy, (b) exploring FMRI data for outliers, (c) running least-squares fits on different models, and (d) making and saving plots with Matplotlib.

	✓-	✓	✓+
Questions	Questions overly simplistic, unrelated, or unmotivated	Questions appropriate, coherent, and motivated	Questions well motivated, interesting, insightful, and novel
Analysis	Choice of analysis overly simplistic or incomplete	Analysis appropriate	Analysis appropriate, complete, advanced, and informative
Results	Conclusions missing, incorrect, or not based on analysis Inappropriate choice of plots; poorly labeled plots; plots missing	Conclusions relevant, but partially correct or partially complete Plots convey information but lack context for interpretation	Relevant conclusions tied to analysis and context Plots convey information correctly with adequate and appropriate reference information
Collaboration	Few members contributed substantial effort or each members worked on only part of project	All members contributed substantial effort and everyone contributed to all aspects of project	All members contributed substantial effort to each project aspect
Tests	Tests incomplete, incorrect, or missing	Tests cover most of the project code	Extensive and comprehensive testing
Code review	Pull requests not adequately used, reviewed, or improved	Pull requests adequately used, reviewed, and improved	Code review substantial and extensive
Documentation	Poorly documented	Adequately documented	Well documented
Readability	Code readability inconsistent or poor	Code readability consistent and good quality	Code readability excellent
Organization	Poorly organized and structured repository	Reasonably organized and clear structure	Elegant and transparent code organization
Presentation	Verbal presentation illogical, incorrect, or incoherent Visual presentation cluttered, disjoint, or illegible Verbal and visual presentation unrelated	Verbal presentation partially correct but incomplete or unconvincing Visual presentation is readable and clear Verbal and visual presentation related	Verbal presentation correct, complete, and convincing Visual presentation appealing, informative, and crisp Verbal and visual presentation clearly related
Writing	Explanation illogical, incorrect, or incoherent	Explanation correct, complete, and convincing	Explanation correct, complete, convincing, and elegant
Reproduciblity	Code didn't run	Makefile recipes fetch data, validates fetched data, generates all results and figures in report	Makefiles generate EDA work and supplementary analysis

Table 1. Project grading rubric. An "A" was roughly two or more check pluses and no check minuses.