Infrastructure and tools for teaching computing throughout the statistical curriculum [*]

Mine Çetinkaya-Rundel [†]
Department of Statistical Science, Duke University and RStudio
and
Colin Rundel [‡]
Department of Statistical Science, Duke University

Abstract

Modern statistics is fundamentally a computational discipline, but too often this fact is not reflected in our statistics curricula. With the rise of big data and data science it has become increasingly clear that students both want, expect, and need explicit training in this area of the discipline. Additionally, recent curricular guidelines clearly state that working with data requires extensive computing skills and that statistics students should be fluent in accessing, manipulating, analyzing, and modeling with professional statistical analysis software. Much has been written in the statistics education literature about pedagogical tools and approaches to provide a practical computational foundation for students. This article discusses the computational infrastructure and toolkit choices to allow for these pedagogical innovations while minimizing frustration and improving adoption for both our students and instructors.

Keywords: R markdown, git / github, reproducibility, data science, workflow, R language, Continuous integration, RStudio, teaching, cirriculum

[*]The authors gratefully acknowledge the constructive feedback from reviewers Nicholas Horton, Jo Hardin, Ricardo Bion, and Jenny Bryan.

[†]mc301@duke.edu

[‡]colin.rundel@stat.duke.edu

1 Introduction and motivation

The 2014 American Statistical Association Curriculum Guidelines for Undergraduate Programs in Statistical Science emphasized the increasing importance of teaching computing and related skills as part of the statistics undergraduate curriculum. Specifically in reference to the increased importance of data science the guidelines state:

"Working with data requires extensive computing skills. To be prepared for statistics and data science careers, students need facility with professional statistical analysis software, the ability to access and manipulate data in various ways, and the ability to perform algorithmic problem-solving. In addition to more traditional mathematical and statistical skills, students should be fluent in higher-level programming languages and facile with database systems." (Workgroup 2014)

Similar arguments have also been made in Finzer (2013) and Nolan & Lang (2010), which argue that computational literacy and programming are as fundamental to statistical practice and research as mathematics. At this time we would like to think that the answer to the question of whether we should be teaching computation as statisticians is clearly yes, and we should now be focusing our efforts on understanding how we can best teach these skills. As with any other foundational topic we believe that students should be exposed early and often to computation, but we also acknowledge that it is not trivial to integrate this additional content into the already saturated statistics curriculum.

We must also recognize that these computational skills are being identified as desirable and valuable by our students and demand is increasing accordingly. Anecdotally, prior to explicitly adding computation to the statistics curriculum at Duke University, exit interviews with undergraduate students graduating with a degree in statistics revealed that the course they reported as being most valuable was a MATLAB programming course offered by the Engineering Department. This was a painful missed opportunity for us as a department as the content diverged from skills and topics that were directly applicable to statistics.

Over the last four years we have revisited our curriculum with this in mind, and with the overarching goal of ensuring that beyond foundational statistics and data analysis knowledge, our majors, minors, or any students in a statistics course acquire fundamental computational data analysis skills. While doing so, we also want to teach best practices for reproducible computing, programming, and collaboration.

In this paper we discuss in detail the technical infrastructure and toolkit choices we have made along the way in revising our curriculum with eye towards minimizing frustration and improving adoption for both our students and instructors. Section 2 discusses our choice of computing environment and technical infrastructure, and provides specific examples of implementation throughout the curriculum. Section 3 details the computational and pedagogical choices made in courses across the Statistical Science curriculum, and Section 4 provides a discussion of our overall course design philosophy and what we hope to be the trickle down effects of choices presented in this paper.

2 Computing environment

Much has been written in the statistics education literature about pedagogical tools and approaches to provide a practical computational foundation for students (e.g. Kaplan 2007, Horton et al. (2014)). This article aims to fill the gap in the literature about how best to set up a computational infrastructure to allow for these pedagogical innovations while keeping student frustration to a minimum.

The most common pain point for getting students started with computation is the very first step – installation and configuration. Regardless of how well detailed and documented instructions may be, there will always be some difficulty at this stage due to differences in operating system, software version(s), and configuration among students' computers. It is entirely possible, and we have experienced first hand, that an entire class period can be lost to troubleshooting individual student's laptops. We have a universal goal for our computational classes to get students to do something interesting with data (e.g., visualization) within the first ten minutes of the first class.

One solution is having students work on the computational aspects of the course exclusively in a computing lab. However this solution is limiting in its own way. First, it's usually the campus IT department, not the instructors, who have administrative access to these computers. As such, it can be a headache to accomplish even basic maintenance tasks like keeping software up-to-date and also leads to generic one-size-fits alls computing environments instead of specializing software and configurations to specific courses. Second, we want students to think of computation as an integral part of the statistics curriculum. Hence we do not want to limit computation to a separate lab session. Our goal is to actively engage students with computation through all facets of our courses which means the computational tools need to be available during lecture as well as during lab.

How, then, can we enable our students to use their own laptops while providing a frictionless onboarding experience? We have opted for a web-based solution, specifically RStudio's server, which only requires a computer with a working web browser (RStudio Team 2016). This offers the best of both worlds since students are able to use the same RStudio integrated development environment (IDE) in both the classroom and the lab, and the centralized server makes it easier to configure and manage the software (R, RStudio) and its dependencies (packages) for all users. The following sections provide more detail on why this particular computing environment was adopted as well as technical details on how we have configured and deployed these software tools.

It is possible to achieve similar learning goals using other tool stacks, for example Python and Jupyter notebooks. Our decision to focus on R and RStudio was based on existing expertise within the department, prepared materials, and benefits of the larger R ecosystem (e.g. the `tidyverse`).

2.1 Background

2.1.1 Why R?

Unlike most other software designed specifically for teaching statistics, R is free and open source, powerful, flexible, and relevant beyond the introductory statistics classroom (R Core Team 2016). Arguments against using and teaching R at especially the introductory statistics level generally cluster around the following two points: teaching programming in addition to statistical concepts is challenging and the command line is more intimidating to beginners than the graphical user interface (GUI) most point-and-click type software offer.

One solution for these concerns is to avoid hands-on data analysis completely. If we do not ask our students to start with raw data and instead always provide them with small, tidy rectangles of data then there is never really a need for statistical software beyond spreadsheet or graphing calculator. This is not what we want in a modern statistics course and is a disservice to students.

Another solution is to use traditional point-and-click software for data analysis. The typical argument is that the GUI is easier for students to pick up and so they can spend more time on statistical concepts. However, this ignores the fact that these software tools also have nontrivial learning curves. In fact, teaching specific data analysis tasks using such software often requires lengthy step-by-step instructions, with annotated screenshots, for navigating menus and other interface elements. Also, it is not uncommon that instructions for one task do not easily extend to another. Replacing such instructions with just a few lines of R code actually makes the instructional materials more concise and less intimidating.

Many in the statistics education community are on board with teaching R (or some other programming language, like Python) in upper level statistics courses, however the value of using R in introductory statistics courses is not as widely accepted. We acknowledge that this addition can be burdensome, however we would argue that learning a tool that is applicable beyond the introductory statistics course and and that enhances students' problem solving skills is a burden worth bearing.

2.1.2 Why RStudio IDE?

The RStudio IDE includes a viewable environment, a file browser, data viewer, and a plotting pane, which makes it less intimidating than the bare R shell. Additionally, since it is a full fledged IDE, it also features integrated help, syntax highlighting, and context-aware tab completion, which are all powerful tools that help flatten the learning curve.

RStudio's direct integration with other critically important tools for teaching computing best practices and reproducible research, some of which we discuss in Section 3.1 and Section 3.2, also influenced our decision for making it central in our toolkit.

2.1.3 Why RStudio server?

A key feature of using a centralized RStudio server instance is that it allows us to provide students with identical computing environments which helps with the introduction of reproducible research and computation, which we discuss in further detail in Section 3.1.

It should also be noted that we do not want to completely dissuade students from downloading and installing R and RStudio locally, we just do not want it to be a prerequisite for getting started with R. We have found that teaching personal setup is best done progressively throughout a semester, usually via one-on-one interactions during office hours or after class. Our goal is that all students will be able to continue using R even if they no longer have access to our departmental resources.

Next we describe details of our two current approaches to implementation. We discuss considerations for how to choose between these approaches, pros and cons of each, as well as where the resources come from (e.g., university vs. departmental).

2.2 Infrastructure

2.2.1 Centralized RStudio server

Our first approach to running RStudio server has been adopted for our higher level courses where we need shared infrastructure and higher end computational resources. To this end our department has dedicated a portion of its yearly computing budget to purchase powerful computer servers (32 cores, 512 GB RAM) that are used specifically for teaching purposes. These servers run free academically licensed RStudio Server Pro, and instructors have direct control over all aspects of the computing environment. More modest configurations are more than adequate (e.g., a mid-to-high end desktop) for the vast majority of use cases, however care should be given for working with larger datasets in a shared enviroment. While we have chosen to run RStudio server on hardware owned and located within the department, this approach would work just as well using virtuaized hardware in the cloud (e.g. EC2, Azure, etc.).

As these servers are a departmental resource, they require a departmental account to login - which works well for our upper division and graduate level courses as most students are directly affiliated with the department. Students taking statistical science courses who are not our majors, minors, or graduate students are issued temporary visitor accounts which expire at the end of the semester.

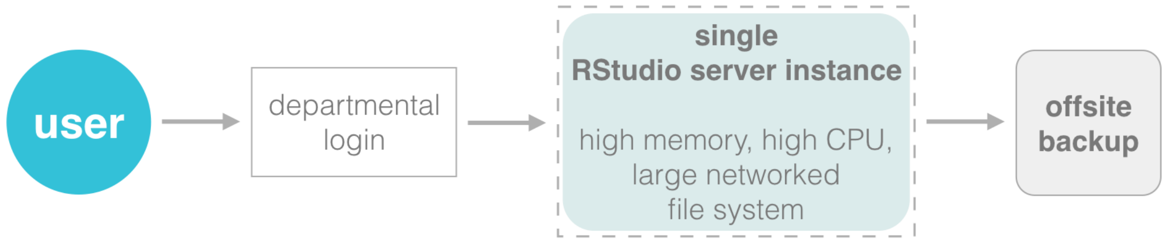

user
departmental login
single RStudio server instance
high memory, high CPU, large networked file system
offsite backup

Figure 1: Monolithic RStudio server architecture

The primary benefit of running and managing the server in-house comes down to control - as needed the instructor(s) are able to install and update software, change configurations, restart or kill sessions, and monitor all aspects of the system. This does increase the demands on the instructor and any involved IT staff, but we have found the benefits to far outweigh the costs. One other unforeseen benefit to a centralized approach is that it makes it possible to present large scale analytical tasks that would not be possible on a traditional desktop or laptop. For example, our advanced courses include a homework assignment where the students need to process a dataset that is on the order of several hundred gigabytes in size, which would not be possible if students were required to use their own system.

2.2.2 Dockerized RStudio server

Our second approach to running RStudio server involves the construction and hosting of a farm of individualized Docker container instances. Docker is a popular and rapidly evolving containerization tool suite that allows users to automate the deployment of software in a repeatable and self-contained way. Each container wraps a portion of the filesystem in such a way that all of the code, runtimes, tools, and libraries needed for a piece of software are available, meaning that software will always run in exactly the same way regardless of the environment in which it is being run. As such, Docker is a powerful tool for reproducible computational research, since every Dockerfile transparently and clearly defines exactly what software and which version is being used for any particular computation task (Boettiger 2015).

One additional advantage of Docker containers is that they are similar to virtual machines in that they are sandboxed from one another. By mapping each student to a single container we are able to keep all student processes segregated and enforce strict CPU, memory, and disk usage quotas to avoid accidental disruption of one another's work.

However Docker containers are generally lighter weight than virtual machines, in terms of system resources used (e.g., memory, CPU, etc.). This makes it feasible to run a large number of containers on a single system at the same time. Since most RStudio usage (particularly by our introductory students) is intermittent, we have found that it is possible to run more than 100 RStudio containers concurrently on a single server. Servers can be run locally or on a cloud-based service. The cost for the latter can be defrayed by the credits many services offer for academic use cases. Currently our setup uses cloud based virtual machines, hosted on Microsoft's Azure, with 4 cores, 28 GB RAM, and 400 GB disk.

A sketch of the architecture of the Docker containers is shown below in Figure 2.

Students use their university login to connect to the website that redirects them to their containers. This is very helpful as it does not require students to create and remember a new ID and password. This approach was made possible by extensive support of from Duke's Office of Information Technology to deploy and support the container servers and create the necessary infrastructure and tools to tie into the university's existing authentication system.

Further details of Duke's containerized RStudio server approach can be found at https://github.com/mccahill/docker-rstudio. This repository contains a README which explains how the large-scale container farm is set up and also contains the Dockerfiles that are used

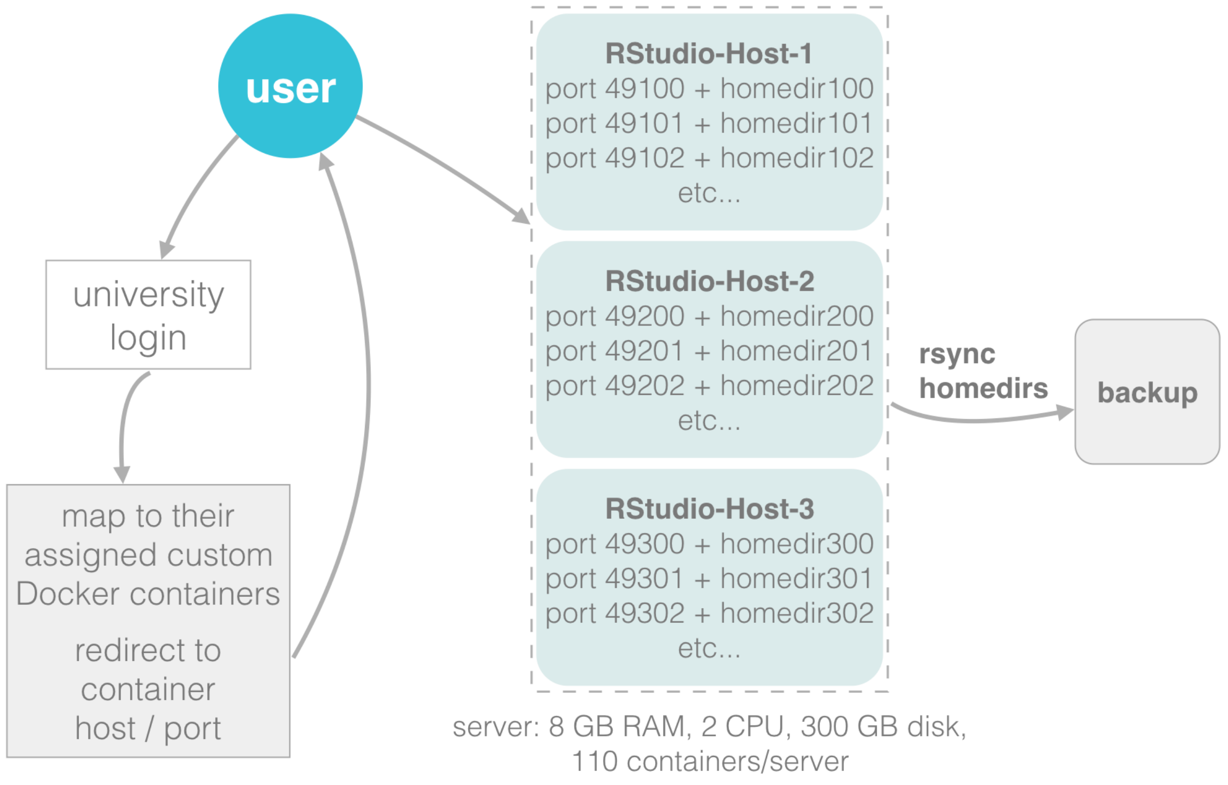


Figure 2: RStudio Docker container farm architecture

to create the individual containers.

Implementing the infrastructure solutions we have discussed above can be overwhelming and time consuming. We encourage faculty interested in adopting these tools to partner with their departmental and/or university IT professionals. Additionally, building these partnerships can lead to collaborations that benefit the entire university. For example, at Duke, the creation of the Docker container systems to support our introductory courses lead to the development of a larger virtual machine / container based infrastructure for RStudio and other scientific computing tools (e.g. Jupyter, MATLAB, and Mathematica).

3 Implementation throughout the curriculum

We have implemented the use of and the emphasis on some or all of the tools and concepts discussed in this article in a variety of courses in the Duke Statistical Science curriculum. Table **??** provides a list of these courses along with their audience and toolkits used.

Table 1: Current courses developed by the authors and the computational infrastructure and tools that are used in each

Course	Name	Audience	RStudio	RMarkdown	git/Github	Unix/Linux	CI
STA 101	Data Analysis and Statistical Inference Spring 2016	Non-major undergrad (social sciences)	X	X			
STA 102	Intro Biostatistics Spring 2016	Non-major undergrad (life sciences)	X	X			
STA 112	Data Science Fall 2014, 2015, 2016, 2017	1st year undergrad (quantitative)	X	X	X		X
STA 323	Statistical Computing Spring 2016, 2017	2nd / 3rd year stat majors and minors	X	X	X	X	X
STA 523	Statistical Programming Fall 2014, 2015, 2016, 2017	1st year MS	X	X	X	X	X

In the following sections we discuss reproducibility with R Markdown as well as version control with git and GitHub. The tools and techniques discussed in this section are what are becoming standard practices in data science teams in industry as well as being more widely adapted by academics. Additionally, in recent years there has been an uptick in courses focusing on data science in many institutions. One of the earlier and inspirational examples of such courses, with fully publicly available resources, is STAT 545 at University of British Columbia (Bryan 2017*b*).

We must acknowledge that the list of technologies and tools presented here might appear somewhat overwhelming, this is understandable but hopefully will not discourage readers from exploring them. These tools reflect just a sampling of a large buffet of options that can be mixed and matched to meet the computational needs of any course.

3.1 Reproducibility with R Markdown

R Markdown provides an easy-to-use authoring framework for combining statistical computing and written analysis in one document (Allaire et al. 2016, Xie (2016)). It builds on the idea of *literate programming*, which emphasizes the use of detailed comments embedded in code to explain exactly what the code was doing (Knuth 1984). Students can use a single R Markdown document to both write, execute, and save code, as well as generate data analysis reports that can be shared with their peers (for teamwork) or instructors (for assessment).

The primary benefit of R Markdown is that it restores the logical connection between the statistical computing and the statistical analysis that was broken by the copy-and-paste paradigm (Baumer et al. 2014). In the copy-and-paste paradigm the statistical software package is used to obtain data analysis results, and then select pieces of these results are copied and pasted into a typesetting program. The author then adds descriptions and interpretations in the typesetting program to generate a complete report (See Figure 3a). The copy-and-paste paradigm is dangerous and disadvantegous (Xie 2015) since

i. synchronization of the two parts being left up to the human compiling them makes it error-prone,

ii. workflow is difficult or impossible to record and reproduce, especially if it involves a graphical user interface based statistical software, and

iii. changes in the data source or input parameters requires going through the same tedious procedure again to re-create the report, which can take just as long time as the original analysis.

As an alternative, the literate programming approach keeps code, output, and narrative all

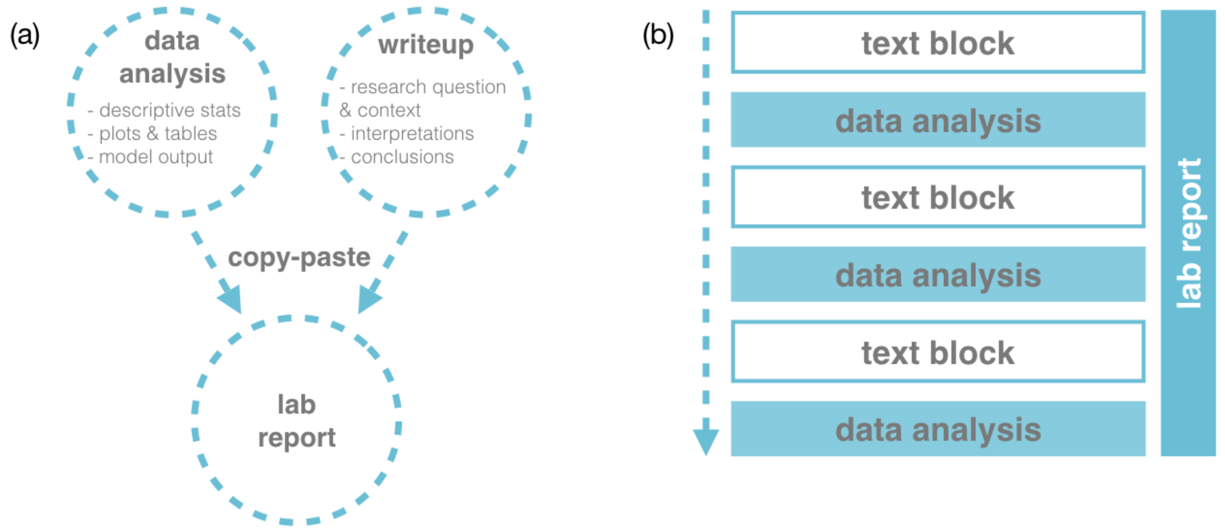


Figure 3: (a) Copy-paste approach to report generation (b) Literate programming approach to report generation

in one document, in fact, makes them inseparable (See Figure 3a). From an instructional perspective this approach has many advantages over the copy-paste paradigm:

i. Reports produced using R Markdown present the code (with syntax highlighting) and the output in one place (as input and output) making it easier for students to learn R and locate the cause of an error.

ii. Enforces students to keep their code organized and workspace clean, which is difficult for new learners to achieve if primarily using their R console to run their code.

iii. Uniformity of the output and the enforced structure of the reports significantly aid the instructors in debugging issues as they arise as well as simplifying the task of grading.

iv. Inherently reproducible analyses make collaboration for teamwork on class projects easier than if students were maintaining their code and narrative on separate documents.

We use R Markdown for all of the courses listed in Table **??**, including the lower level courses where students start out with no previous background in computation. We are able to do this due to the very lightweight syntax of the markdown language as we do not want students to be overburdened by having to learn both R syntax as well as another language at the same time. We also facilitate students getting started with R Markdown by providing them templates that they can use as starting points for their lab reports. For earlier labs in the semester these templates include section headings for each exercise, some pre-populated and some empty R chunks where they can enter their code, as well as directions for where to type their descriptions and interpretations. Throughout the semester we take away the scaffolding in the templates slowly, and by the end of the semester students are able to produce a fully reproducible data analysis project that is much more extensive than any of their weekly labs.

In our higher level courses (STA 323 and STA 523) students encounter assignments that push the limits of what R Markdown can handle (e.g. long running computational steps). In these cases we build on the foundation of R Markdown by introducing additional tools like the GNU make (Stallman et al. 2016) build system and discuss how to maintain a reproducible workflow (Bostock 2013, Broman (2016)).

3.2 Version control and collaboration

3.2.1 git

One of the defining principles behind how we teach computation is that everything we and our students produce should be reproducible – how you got a result is just as important as the result itself. Implicit in the idea of reproducibility is collaboration, the code you produce is documentation of your process and it is critical that you are able to share it (even if only with yourself in the future). In our more computationally focused courses, our goal is to teach students tools that make this documentation and collaboration as robust and painless as possible. This is best accomplished with a distributed version control system like git. Much has already been written about the utility of this type of tool for enhancing a reproducible workflow (Bryan (2017*a*), Ram (n.d.), and Loeliger & McCullough (n.d.)). In this article we focus on using git specifically in a classroom setting.

In the classroom we have adopted a very top down approach to teaching git – the students are *required* to use it for *all* assignments. These type of tools tend to suffer from delayed gratification as when they are first introduced the students view them as an unnecessarily and clunky addition to their workflow and it is not until weeks or even months later that they experience the value first hand.

The learning curve for these tools is unavoidable but we have found it best to focus on core functionality. Specifically, we teach a simple centralized git workflow which only requires the student know how to use git `push`, `pull`, `add`, `rm`, `commit`, `status`, and `clone`. These seven commands are more than enough to handle almost all of the situations students will encounter early on.

Depending on the level of the class we introduce these functions directly through the command line (when we are also spending time teaching the Unix shell) or via RStudio's project based git GUI. We have found that the vast majority of students prefer to interact with git via this GUI when given the chance, but it is also not unusual for students to mangle their repositories such that the command line tools become necessary. The most complicated task students regularly encounter are merge conflicts, most of which are straight forward to resolve. We have noticed that students often develop elaborate workflows to avoid these types of issues but they eventually come to understand the resolution process. It has also been helpful to encourage students to commit early and often to reduce the size of each change as well as requiring that they only commit code files (e.g. `.Rmd`) and not intermediate (`.md`) or output (`.html`) files. Finally, we have found that in the early stages of learning git it is useful to engineer situations in which the students encounter problems while they are in the classroom so that the professor and teaching assistants are present to troubleshoot and walk them through the process in person.

3.2.2 GitHub

The use of GitHub also goes a long way to help students visualize and understand the git process which also aids in student buy-in. The web interface allows students to easily view diffs (file changes over time) in files they are collaborating on, keep track of commit histories, as well as search both the current state as well as the entire history of the code base. Within the classroom environment GitHub can be thought of as an advanced and flexible learning management system (compared to traditional tools like Blackboard or Sakai).

At its most basic, GitHub can be used as a central repository where students turn in their work and where the professor and teaching assistants then collect it and provide feedback. However using this ecosystem for only assignment submission ignores the most compelling features and advantages. In our classes students are expected to push their work in progress throughout the assignment period. This is not enforced explicitly, but rather through the design of the assignments. Most assignments are large scale and team based, meaning no one student can easily complete all the work on their own. In addition, the various tasks within the assignment are interdependent, meaning students are not able to divide up the work and complete each piece individually. This type of design strongly encourages the students to share their work in progress which they are able to do using GitHub. This is also useful to the instructor as it allows for opportunities for observation and feedback through the course of the assignment without forcing students to turn in "drafts".

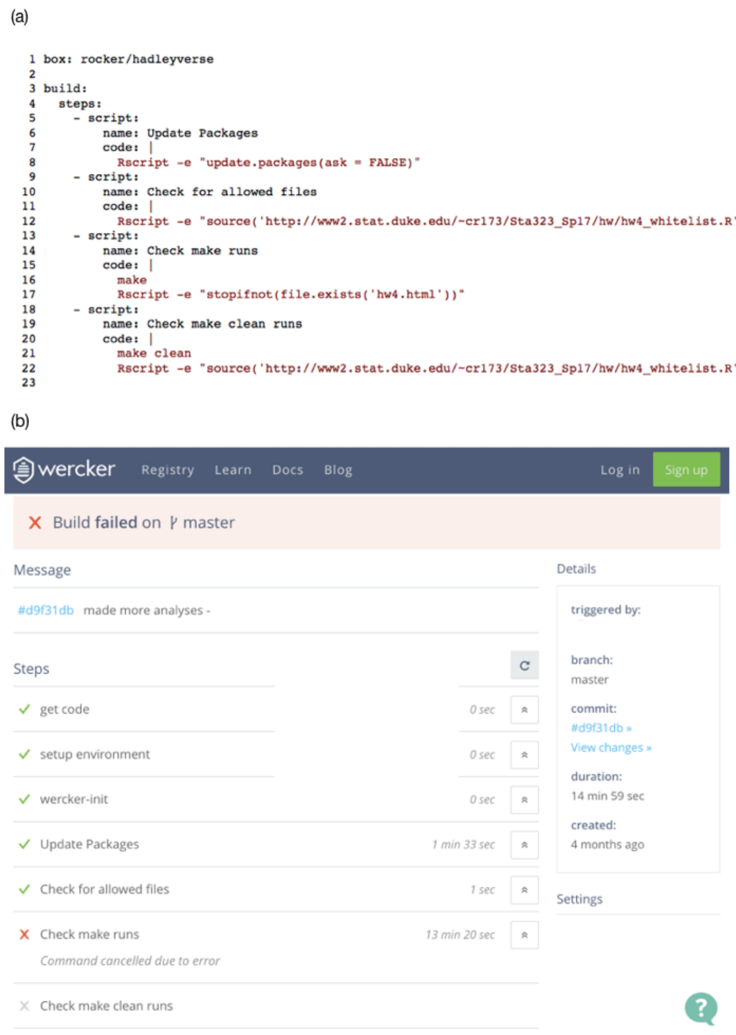
Additionally, GitHub's organization and teams features are a natural fit for managing course related tasks. We have used a model where each class has a separate organization to which the students are invited at the beginning of the semester. For most classes the computation components are team based which are then represented by teams within the GitHub organization. This allows for the creation of separate team based repositories along with fine grained access permissions. In general, we have found that using one repository per team per assignment works best. In order to comply with Family Educational Rights and Privacy Act (FERPA) requirements all student repositories are kept private by default, which is possible at no cost thanks to GitHub's generous academic discount policy. Setup and management for larger classes can be somewhat challenging due to the sheer number of components, however we have found that most actions can be scripted via the GitHub API which can dramatically reduce the course administrative workload. Examples of some of the tools we have developed for this purpose can be found in a GitHub repository we have made publicly available for use by interested instructors (Rundel 2017).

3.2.3 Continuous integration

Another advantage to the GitHub ecosystem is that it provides access to a number of third party tools that offer additional functionality. One area of tooling that we are particularly excited about is continuous integration. Tools for continuous integration have become increasingly popular in the software development community as they allow developers to define specific actions to take place after code is pushed to GitHub. Most often this is used to run unit tests which check whether the most recent changes break any existing functionality. Within the R community there has been widespread adoption of the Travis CI suite of tools for testing R packages as they are developed. These types of tools are immensely useful in terms of providing (almost) immediate feedback and helping developers maintain high quality, working code.

Instant feedback has been shown to have positive outcomes in student learning and performance in many disciplines, including computer science (Edwards 2003, Wilcox (2015)). However it is less obvious how testing tools that generate instant feedback can best be applied within the classroom context. For example, if students are asked to write a function that calculates values in the Fibonacci sequence, it is straightforward to write tests that check that for a given input their function returns the correct value. However, if instead the students were asked to develop a model for predicting housing prices, it is not necessarily obvious what a correct answer should look like and how best to test for it.

In our statistics courses we most often fall into the latter case, and we do not want to constrain assignments to only contain tasks that are easily testable. In order to take advantage of the immediate feedback of these continuous integration tools we have opted to focus on testing process over testing correctness. Specifically, we primarily use the continuous integration tool Wercker to test the reproducibility of the students' work. For the simpler assignments this involves checking that the assignment's R Markdown document can be compiled. This allows for a quick check for common reproducibility mistakes like setting the wrong working directory or assuming that a package is universally available. In this way the students get immediate feedback that helps them identify and correct this kind of problem without instructor intervention. Figure 4 shows an example of Wercker checks as well as the immediate feedback a student has received via this tool upon pushing their work to GitHub.



```
box: rocker/hadleyverse

build:
  steps:
    - script:
        name: Update Packages
        code: |
          Rscript -e "update.packages(ask = FALSE)"
    - script:
        name: Check for allowed files
        code: |
          Rscript -e "source('http://www2.stat.duke.edu/~cr173/Sta323_Sp17/hw/hw4_whitelist.R')"
    - script:
        name: Check make runs
        code: |
          make
          Rscript -e "stopifnot(file.exists('hw4.html'))"
    - script:
        name: Check make clean runs
        code: |
          make clean
          Rscript -e "source('http://www2.stat.duke.edu/~cr173/Sta323_Sp17/hw/hw4_whitelist.R')"

```

✗ Build failed on master

Message

#d9f31db made more analyses -

Steps

✓	get code	0 sec
✓	setup environment	0 sec
✓	wercker-init	0 sec
✓	Update Packages	1 min 33 sec
✓	Check for allowed files	1 sec
✗	Check make runs *Command cancelled due to error*	13 min 20 sec
✗	Check make clean runs	

Details

triggered by:

branch: master

commit: #d9f31db »

View changes »

duration: 14 min 59 sec

created: 4 months ago

Settings

Figure 4: Examples of a Wercker checks (as specified by the YAML in (a)) and build results (b)

These continuous integration tools can also be used for more complex tasks. For example one assignment in STA 523 involves the students predicting the spatial boundary of the 23 police precincts in Manhattan. Using Wercker we are able to automatically score the accuracy of each team's prediction and provide a live leader-board similar to a Kaggle contest. (See Appendix **??** for a detailed description of this assignment.)

4 Discussion

In designing statistics courses and deciding on their computational components, our overall goal is to increase student buy-in with approachability and usability. The combination of the computing infrastructure and the software toolkit and ecosystem we describe here is not the only way to do reproducible data analysis. However, we believe that it is an efficient and effective way to minimize the friction of onboarding students to reproducible statistical computing, and that this approach can be tailored to all stages (introductory to advanced) of a statistics curriculum.

Another way by which we ensure student buy in is by making computing, and the entire toolkit associated with it, a central component of required course content *and* assessment. For example, using GitHub as the sole course management system means students must use it to be able to submit their assignments for assessment, and hence they get acquainted with the system early and make sure to ask questions no later than the due date of the first assignment. Making the use of git and GitHub optional would not have nearly the same impact. Similarly, by requiring all students to complete their assignments using R Markdown forces students to employ a literate programming approach to their analyses. Employing the principles early in the curriculum is particularly valuable, since we are able to teach students to produce fully reproducible work before they learn any other workflow. It is far more efficient to inoculate researchers against bad computational habits than to retrain them after those habits have already formed.

We hope that there will be at least three main trickle down effects of this approach. The first is that students will be better prepared for upper level statistical courses where the methodologies being taught have a substantial computation component (e.g., MCMC). Very often instructors of these classes have had to start each semester from scratch with an introduction to R session before they can move on to teaching the methods and applications that are actually relevant to their course. Through a more systematic and earlier introduction, redundant and often ad hoc introduction to computation can be eliminated. The second trickle down effect we would like to see is better preparation of students to engage in research through either independent study or as part of a senior thesis. Given the Bayesian focus of our department there is a substantial computational component to almost all of our facultys' research projects - students will be better prepared to contribute to these projects if they are also not expected to learn computational and research best practices at the same time. Lastly, the third trickle down effect is in computational expertise of faculty in statistics departments. Implementing the infrastructure and teaching the tools we describe in this article requires that faculty have techical experience and expertise using and teaching them. Statistics departments need to encourage, hire, and promote faculty whose expertise lies in these domains as well as be willing to invest in computing resources since they are the skills that our students want and need to learn (Waller 2017).

While it is too early to tell the long term effects of all of the changes to the curriculum that we discussed in this article, student feedback from initial runs of these courses have been very positive, as seen in course evaluations and exit interviews with graduating students. With appropriate infrastructure and scaffolding, introductory statistics students are receptive to the addition of the computational data analysis component. The higher level courses we offer as electives are in high demand and have been effective in preparing students for higher quality research projects and making them competitive for internship and employment opportunities.

Appendix

Assignment example: "Parking Wars: Manhattan"

Background

New York City is at the forefront of the open data movement among local, state and federal governments. They have made publicly available a huge amount of data (NYC Open Data) on everything from street trees, to restaurant inspections, to parking violations. It is the last of these that we will be focusing on for this homework assignment.

This will be our first foray in to big*ish* data as the CSV file containing these data is roughly 1.7 gigabytes in size with 9.1 million observations over 43 variables. This is not so big that we can't run our analyses on a moderately powerful laptop, but it is large enough that we need to start paying attention to what we are doing and the performance of our code.

The data contains all parking violations from the five boroughs of New York City from between August 2013 and June 2014. We will simplify matters somewhat by focusing our analyses solely on Manhattan (excluding Brooklyn, the Bronx, Queens, and Staten Island).

Data

The violation data along with supplementary data is available on gort in `/data/nyc_parking/`. The available data are:

- `NYParkingViolations.csv` - all parking violations data available from between August 2013 and June 2014 (source).
- `pluto_manhattan` - directory containing a shapefile from MapPLUTO which merges tax lot data with tax lot features (polygons). This shapefiles will be used for geocoding, as it connects property boundary polygons with addresses.
- `nybb` - directory containing NYC borough boundary shapefile.

- `altnames.csv` - some possibly useful information on alternative names for certain streets in NYC.
- `fine_definition.csv` - information on the type and amount of fine based on violation code.

Task 1 - Geocoding

The parking violation data contains a large number of variables that we do not care about for the time being. For this first task your job is to attempt to geocode (find latitude and longitude for each entry) as much of the data as possible using the given variables. Note that this data has had minimal cleaning done, there are a large number of errors, omissions, and related issues. Also, note that there is a very large number of citations issued, 9.1 million over the course of the year, so even under the most optimistic of circumstances you will not be able to, nor should you, use any of the standard web based geocoding services.

In order to be successful at this task you do not need to geocode every address, or even most addresses. The goal is to geocode as many as possible with as much accuracy as possible to enable you to be successful with the 2nd task. This is a messy, large, and complex data set and at the best of times geocoding is a very difficult problem - go for the low hanging fruit first and then work on the edge cases / exceptions later as needed.

Your write up for this task should include a description of any and all cleaning / subsetting / etc. that was done to the data, as well as a description of your geocoding approach(es) and a discussion of how successful each was.

Task 2 - Recreating NYC's Police Precincts

The ultimate goal of this assignment is to attempt to reconstruct the boundaries of the 22 Manhattan New York City police precincts (numbered between 1 and 34). The parking violation data set contains the column, `Violation.Precinct`, that lists the police precinct in which the violation ostensibly took place. Your goal is to take this data along with the geocoded locations from Task 1 and generate a set of spatial polygons that represents the boundaries of the precincts.

As mentioned before, the data is complex and messy so keep in mind that there is no guarantee that the reported precinct is correct, or the street address, or even your geocoding. As such, the goal is not perfection, anything that even remotely resembles the precinct map will be considered a success. No single approach for this estimation is likely to work well, and an iterative approach to cleaning and tweaking will definitely be necessary. I would suggest initially focusing on a single precinct to develop your methods before generalizing to the entirety of Manhattan.

To make things more interesting I will be offering a prize for the team that is best able to recreate the precinct map as judged by the smallest total area of discrepancy between your prediction and the true map. In order to win the prize you must abide by the rules as detailed below. I will maintain a leader board so that you will be able to judge how well you are doing relative to the other teams in the class.

For this task you are expected to produce a GeoJSON file called `precinct.json`, for details on formatting see the hw_example repo. Your write up should include a discussion of your approaches to generating the boundaries and at least a simple visualization of your boundaries on top of the Manhattan borough boundary.

Rules

- There will is a hard limit of 2 hours of run time for this assignment, I should be able to run `make` and have the entire analysis and all output finished within 2 hours on saxon.
- If you wish to use any additional data source(s), *you must* first check it with me and I will approve it or not. Additional data may only be used to improve the quality of your geocoding, you are not allowed to use anything beyond the geocoded parking violation data to directly estimate the precinct boundaries.
- You may not use any existing precinct data regardless of source. I am aware that the

precinct boundaries are only a google search away - avoid the temptation. If I suspect that you have used this data at any point in your analysis your whole team will be disqualified from the performance contest and I also reserve the right to penalize your assignment grade for particularly egregious cases. This applies even to instances where you solely use the data to score yourself.

- Do not create unnecessary copies of the data, you *should not* create a local copy of `NYParkingViolations.csv` in your home directory on saxon. If you absolutely must, you can maintain a copy on your own laptop. If you are saving intermediary files, make sure to remove as much unnecessary data as possible (columns and or rows) and save the file in a binary format (e.g. .Rdata). Be aware of the size of your files and your disk usage and be very careful to not commit any large files to git as this can prevent you from being able to push to github.

Submission and Grading

This homework is due by 11:59 pm Monday April 10th. You are to complete the assignment as a group and to keep everything (code, write ups, etc.) on your team's github repository (commit early and often). All team members are expected to contribute equally to the completion of this assignment and group assessments will be given at its completion - anyone judged to not have sufficient contributed to the final product will have their grade penalized. While different teams members may have different coding backgrounds and abilities, it is the responsibility of every team member to understand how and why all code in the assignment works.

The final product for this assignment is a single document named `hw6.Rmd` that contains a write up for your approach to both the geocoding of the violation addresses and the reconstruction of the police precinct boundaries. Your repository should also include commented R script(s) that implement both of these tasks as well as an appropriate Makefile which can be used to rerun the analyses, generate the `precinct.json` file, and compile `hw6.Rmd`.

References

Allaire, J., Cheng, J., Xie, Y., McPherson, J., Chang, W., Allen, J., Wickham, H., Atkins, A. & Hyndman, R. (2016), *rmarkdown: Dynamic Documents for R.* R package version 0.9.6.
URL: *https://CRAN.R-project.org/package = rmarkdown*

Baumer, B., Cetinkaya-Rundel, M., Bray, A., Loi, L. & Horton, N. J. (2014), 'R markdown: Integrating a reproducible analysis tool into introductory statistics', *Technology Innovations in Statistics Education* **8**.

Boettiger, C. (2015), 'An introduction to docker for reproducible research', *ACM SIGOPS Operating Systems Review* **49**(1), 71–79.

Bostock, M. (2013), 'Why use make', https://bost.ocks.org/mike/make/. Accessed: 2016-09-15.

Broman, K. (2016), 'minimal make: A minimal tutorial on make', http://kbroman.org/minimal_make/. Accessed: 2016-09-15.

Bryan, J. (2017*a*), 'Excuse me, do you have a moment to talk about version control?', *PeerJ Preprints* **5**, e3159v1.

Bryan, J. (2017*b*), 'Ubc stat 545 - data wrangling, exploration, and analysis with r', http://www.stat545.com/. Accessed: 2017-08-17.

Edwards, S. H. (2003), Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance, *in* 'Proceedings of the International Conference on Education and Information Systems: Technologies and Applications EISTA', Vol. 3, Citeseer.

Finzer, W. (2013), 'The data science education dilemma', *Technology Innovations in Statistics Education* **7**(2).

Horton, N., Baumer, B. & Wickham, H. (2014), Teaching precursors to data science in introductory and second courses in statistics, *in* 'ICOTS-9'.

Kaplan, D. (2007), 'Computing and introductory statistics', *Technology Innovations in Statistics Education* **1**(1).

Knuth, D. E. (1984), 'Literate programming', *The Computer Journal* **27**(2), 97–111.

Loeliger, J. & McCullough, M. (n.d.), *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, "O'Reilly Media, Inc.".

Nolan, D. & Lang, D. T. (2010), 'Computing in the statistics curricula', *The American Statistician* **64**(2), 97–107.
URL: *http://dx.doi.org/10.1198/tast.2010.09132*

R Core Team (2016), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
URL: *https://www.R-project.org/*

Ram, K. (n.d.), 'Git can facilitate greater reproducibility and increased transparency in science', **8**(1), 7.

RStudio Team (2016), *RStudio: Integrated Development Environment for R*, RStudio, Inc., Boston, MA.
URL: *http://www.rstudio.com/*

Rundel, C. (2017), 'GitHub Classroom Tools', https://github.com/rundel/ghclass.

Stallman, R. M., McGrath, R. & Smith, P. D. (2016), *GNU Make: A program for directing recompilation*.
URL: *https://www.gnu.org/software/make/*

Waller, L. A. (2017), 'Documenting and evaluating data science contributions in academic promotion in departments of statistics and biostatistics', *bioRxiv* .
URL: *http://www.biorxiv.org/content/early/2017/01/25/103093*

Wilcox, C. (2015), The role of automation in undergraduate computer science education, *in* 'Proceedings of the 46th ACM Technical Symposium on Computer Science Education', ACM, pp. 90–95.

Workgroup, A. S. A. U. G. (2014), 'The 2014 american statistical association curriculum guidelines for undergraduate programs in statistical science'.
URL: *http://www.amstat.org/education/curriculumguidelines.cfm*

Xie, Y. (2015), *Dynamic Documents with R and knitr*, 2nd edn, Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1498716963.
URL: *http://yihui.name/knitr/*

Xie, Y. (2016), *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.13.1.
URL: *http://yihui.name/knitr/*