Best practices for teaching reproducibility: motivation, direct instruction and practice

Joel Ostblom *Tiffany A. Timbers*

Introduction

The data science definition that we embrace in the master of data science program at the University of British Columbia (UBC) and in the undergrad data science courses that we're developing is the study and development of reproducible and auditable processes to extract insight from data. Using this definition requires that we also define what is meant by a reproducible and auditable analysis. To define reproducible analysis, we embrace the National Academy of Sciences definition, which is reaching the same result given the same input, computational methods,and conditions (2019). For auditable or transparent analysis, we follow how it has been defined by Hilary Parker (2017) and Karthik Ram (2013), which is that there should be a readable record of the steps used to carryout the analysis (i.e., computer code) as well as a record of how the analysis methods evolved (i.e., a version controlled project history). This history is important for recording how and why decisions to use one method or another were made, among other things.

The reason we embrace this definition of data science, is that we believe that data science work should bring insight (e.g., answer an important research question) and employ reproducible and auditable methods so that trustworthy results and data products can be created. Results and data products can be generated without reproducible and auditable methods, however, when they are built this way there is less confidence in how the results or products were created. This is mainly because of three reasons:

1. They lack evidence that the results or product could be regenerated given the same input computational methods, and conditions.
2. There is insufficient evidence of the steps taken during creation.
3. There is an incomplete record of how and why analysis decisions were made.

In addition to contributing to the trustworthiness of data science work, employing reproducible and auditable methods and workflows bring additional benefits

to data scientists, such as in the form of more effective collaboration. Data science is an inherently collaborative science, and the emphasis of reproducible and auditable methods in data science greatly facilitates the act of collaborating in many context, further emphasizing the importance of learning this skill well.

Although the many benefits of reproducible workflows may make them seem like an exciting topic for incoming students, the experience when enter a classroom of curious data scientists in training is quite the opposite. Students are usually keen to learn about data science but what they're most excited about is the second part of its definition: extracting insights from the data.

Students are often not even aware of the reproducible and auditable processes of data science and when they first hear about them, they tend to regard them as an inconvenient means to an end rather than an important skill to master. This outlook is likely at least partly motivated by that these processes do not directly lead to novel insights in the same way as a predictive model might, which is what many students have in mind when they envision the work of a data scientist. This negative predisposition creates another barrier to overcome when teaching the reproducible and auditable aspects of data science.

An additional pedagogical challenge is that the tools that we use for reproducibility are not necessarily smooth and easy to learn, but often have a steep learning curve. Over our five years of teaching these topics at UBC we've found three pedagogical strategies that are particularly effective for teaching reproducibility successfully:

1. placing extra emphasis on motivation
2. live demonstration
3. lots of practice

In this paper, we will discuss why we believe each of these are important, provide high-level examples of how to incorporate these in your teaching, and walk through one detailed example of how we have implemented each of these in our courses on reproducibility at UBC.

Placing extra emphasis on motivation

Why do we need extra motivation when teaching reproducibility, compared to some other data science topics, such as machine learning? We think this is because students do not have intrinsic excitement or motivation on the topic of reproducibility, they have little prior knowledge on this topic, and reproducibility concepts and in particular tools are challenging to learn.

One example, the version control software that's the most commonly used one for reproducibility, Git, is notorious for being difficult to learn (Figure 1). Furthermore, there are many anecdotes that that most people don't actually learn

it deeply and they just get by trying a variety of commands until they find some things that work. This can lead to learners getting themselves into a lot of trouble. Sometimes this trouble is so difficult to get out of, that even professional data scientists and data science educators, for example Jenny Bryan, recommend the practice of "burning it all down" and starting from scratch - which really defeats many of the purposes of version control.

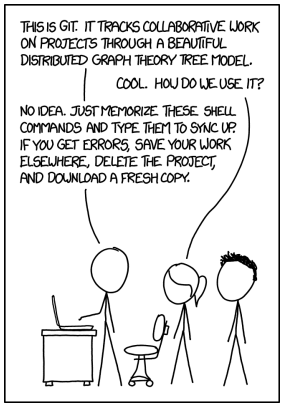

Fig. 1: Infamous xkcd comic that highlights the difficulty of learning and using the version control software Git.

Another example is R Markdown, which is an implementation of literate code documents (TODO cite Knuth here) that are useful for generating reproducible reports. Many aspects of R Markdown are quite user friendly, however rendering the source R Markdown document to PDF depends on LaTeX, and when users make formatting errors that impact LaTeX's job in the rendering you can get some very cryptic error messages that are not very clear to learners about where the error is coming from and what should be done next to fix the problem.

```
# TODO Add a figure here that illustrates an example of an R Markdown doc being
# rendered to PDF that has an error, and then the cryptic error message.
# This could simply be a screen shot from RStudio with the R Markdown doc
# open in the editor and the R Markdown tab showing the error
```

Yet another example of a reproducibility tool that is challenging to learn is Docker. Docker is a containerization tool that extends beyond just including the package dependencies of your analysis workflow, and instead creates an versioned copy of your entire computational environment including your operating system and any of its installed software. This means that you can share an exact copy of your environment with your colleagues even is you work on different operating systems, and you can test drive a server environment on your laptop. Docker is an especially challenging one to teach and motivate students to learn because it's so different from writing code to for analysis. This is because it

takes a long time to install things and it takes a long time to automate the process of installing things and making a shareable compute environment. It is also not an exciting part of a data analysis, things already work on the students own computer. So it is difficult to convince students why they should put effort into learning and using this.

```
# TODO Need to figure out a way to illustrate this...
```

So what do we do to motivate learning reproducibility concepts and tools in our classrooms? We have found the following three strategies helpful:

1. Telling stories from the trenches
2. Letting them fail (in a controlled manner)
3. Study cases of failures with real world consequences

Telling stories from the trenches

One thing that we've used is telling stories from the trenches. Those of us who teach these courses usually have had some experience with doing research in their Ph.D.'s or Postdoctoral studies, or are still currently doing research, and through these are lived experiences of learning reproducibility tools and applying it to our research we have made mistakes, our collaborators have made mistakes and we can share these with our students. In the Master of Data Science program, a lot of the students have work experience that may have touched on data before and so they also have stories from the trenches. In a classroom with such students, you can carryout think pair share exercises around these stories and get the students to talk about their stories as well as hear your stories.

Letting them fail (in a controlled manner)

Another example that we learned from Jenny Bryan's teaching of STAT 545 at UBC is to ask students to run someone else's analysis and let them fail. We have taken this approach, but tweaked it so that they experience this in a more limited, controlled manner.

Notably, many instructors (including the authors of this work) have themselves experienced failure in graduate school and during their postdoctoral research in regards to reproducibility, which negatively impacted their work. While these experiences motivate teaching and using reproducibility concepts and tools for instructors, most undergraduates and new graduate students cannot draw on similar professional experiences. Rather than letting new students live through the full perils of irreproducible research, we can set up controlled scenarios to

expose them to these downsides in a controlled, accelerated manner while still embodying much of the same motivational benefits.

Below we present a more detailed example of controlled failures, but in short we provide students an analysis that is not reproducible, and thus will likely fail on someone else's machine, and then ask them to run it. If they cannot run it, we ask them to fix it so they can, giving them hands on experience with this laborious task. Then we provide them the same analysis that has been made reproducible, and can directly be run on the students' machines without any tweaking. Under these controlled circumstances failure and frustration can have a positive impact on students overall learning as they experience the many benefits of reproducibility first hand. In a relatively short amount of time, this helps provide motivation to endure the steep learning curves of reproducibility concepts and tools.

Detailed example lesson of letting them fail (in a controlled manner) The classroom that this has been used in is DSCI 522 - Data Science Workflows that is 2/3 of the way into a professional Master's of Data Science program at UBC. This course is a project-based course where we teach them all the concepts and skills needed to conduct reproducible data analyses. Near the end of the course one of the things that we get them to do is to make their compute environments for their analysis reproducible via containerization of the compute environment using a tool called Docker.

Docker is not an easy thing to teach or learn. There is a lot of layers of pre-requisite knowledge related to Unix/Linux that students need to know. To be able to run Docker containers students need to be able to run commands in the terminal. And to be able to build their own Docker images, they need to know how to install software using Linux operating systems at the command line (as its these kinds of commands that need to be written in a text file that specifies how to build a Docker image). Most data science students do not come to the reproducibility classroom with this prerequisite knowledge, so there are additional barriers to teaching this subject, which further emphasizes the need to evoke sufficient internal motivation among students.

The specific exercise that we use to provide this motivation in the classroom is to give students a data analysis project pipeline that is available on github (https://github.com/ttimbers/data_analysis_pipeline_eg/tree/v3.0). In class we ask them to go to that GitHub repository, read the instructions and try to replicate the analysis on their own computers. In this course, our students already have enough Git skills to clone the repositories. In the GitHub repository that we provide them, we have not used any compute environment reproducibility tools and we have intentionally included in a lot of obscure software packages that we know they don't have installed. In some cases, we add them cryptically in the middle of a script so they have to do a little bit more work than just look at the top of the script to find out what packages they need to install. It takes students

some time to accomplish this task that you set before them, but eventually they figure it. However, we make it tricky enough that it's frustrating.

After this, we give them the same analysis in a different GitHub repository (https://github.com/ttimbers/data_analysis_pipeline_eg) that uses Docker as the compute environment reproducibility tool. There is a Docker image that lives on DockerHub (a container repository), and project `README` gives clear instructions on how to run and replicate the analysis using Docker. Students experience that they are able to this in a few minutes and with very little effort.

Study cases of failures with real world consequences

The third way to create motivation is something that the authors are currently trying to build into their courses and yet have limited experience test driving it, but think the idea has strong merit and so wish to present it here. This is the pedagogy of using case studies of failures that have had real world consequences.

Live demonstration

We suggest that live demonstration is important in the classroom, specifically when teaching reproducibility. From our experience, reproducibility is not something that most people or students figure out through exploration- and inquiry-based learning, or if they do it is not an efficient process. We hypothesize the reason for this is that reproducibility uses a lot of borrowed tools from software engineering that are being repurposed for science and reproducibility. Thus, a lot of the getting up and getting started with reproducibility has a lot of assumed knowledge behind it, and there are not a lot of like clear and easy on roads for learners who do not have a software engineering background. Part of this may stem from the field being still fairly new and not-yet as widely embraced as we might hope. This means that there is not a lot of culture around using these in data science and statistics, this it is not yet as obvious where the on-ramps are.

Furthermore, similar to why we need extra motivation, the challenge of learning to use the tools due to their steep learning curves suggests that having some live demonstration is beneficial to learners. These points are well stated in a blog post and essay on *The Role of Theory in Data Analysis* by Roger Peng:

> *There is no need for a new data analyst to learn about reproducibility "from experience." We don't need to lead a junior data analysis down a months-long winding path of non-reproducible analysis until they are finally bitten by the non-reproducibility bug (and "therefore learn their lesson"). We can just tell them*
>
> *"In the past, we've found it useful to make our data analysis reproducible Here's a workflow to guide you in your own analysis."*

> *Within that one statement, we can "compress" over 20 years of experience.*

The authors of this article agree with the statement above and think we owe it to our students to directly instruct them on the best practices that the reproducibility community has arrived on to date and then show them how to use these tools explicitly.

So how do we use live demonstration in the reproducibility classroom? In data science programming classes it is common to use live coding to show how to use R and Python. A similar pedagogy of live demonstration works well for reproducibility tools, including R Markdown or Jupyter notebooks for reproducible reports, using version control with Git and GitHub, and using Docker to create reproducible and shareable computational environments. We believe that live demonstration makes it more obvious to the students how to use these tools in practice, and facilitates lateral knowledge transfer where learners absorb additional material by observing *how* we work. Additionally, when you make mistakes as an instructor in these live demos, it humanizes the reality that working with these tools that are somewhat challenging, even for experts. Intentional mistakes also can provide opportunities to spend more time on that area of the topic and explain the gotchas of a common mistake, and how to fix it.

A word of caution with live demonstration when teaching reproducibility; because we are teaching things that involve graphical user interfaces and/or that come from software engineering, the tech stack moves very fast. This means that each semester we teach these tools, we need to test drive the materials before we share them with the students to see if something has changed.

A relatively recent example of this is from fall 2020, when GitHub decided to change the name of their default branch from master to main (rightfully so). This change broke several of our teaching demos, guided worksheets, and lab homework. It also caused all of our notes to have to be rewritten. Eight months later, this problem is still not completely solved as we have several resources that are still using the master branch that we have not been able to quickly change over. For the reasons argued above, we do believe that it is really important to use live demonstration when teaching reproducibility, however, this should be done with the awareness and the acceptance that these kinds of changes are going to happen relatively frequently. Which means reproducibility instructors are going to have to update or make a new live demos and guided worksheets each year. Without this, the course resources will quickly fall out of usefulness.

Example lesson using live demonstration

Here we provide an example of how we use live demonstration to teach version control in our first year introduction to data science course. In this course, we take a three-pronged approach for directed instruction.

We provide them a textbook reading for them to review before class, then we do a live demonstration in class, where they watch us use the GitHub website, and the Jupyter Git graphical user interface to add, commit, push and pull changes. Then finally they work through a guided worksheet that asks them to do the same thing that we just demonstrated. The guided worksheets also asks them questions along the way to test that they really understand what is adding, what is committing, what is pulling, what is pushing and where is the work going.

The challenge or limitation with this lesson in particular, is that we have chosen to teach using a Git graphical user interface, as opposed to the Git command line tool, due to the very novice level of the learners. Using a graphical user interface, and a newer one, means that we need to more frequently update and fix our lesson as the tool changes. The Git command line tool is more stable, and command line tools have less room to change generally, compared to graphical user interfaces, and thus would be a more stable tool to build a lesson around. However, the trade-off would be that this is a bit less intuitive for new learners, especially those who are also new to the command line in general.

Lots of practice

The third practice we argue for when teaching reproducibility is lots and lots of practice. To build lasting changes in behavior humans need to commit new information to long-term memory, which we can do in two fundamental ways: one trial learning and repetitive spaced practice. One trial learning usually requires some sort of emotional impact, such as a traumatic event, or a really positive emotional event (a great birthday or a wedding). Forming lasting memories from these emotional experiences, doesn't require repetition. However, that is not how we learn most of the topics taught at school, which tend to be less emotionally salient. Such topics are instead better learned through repetitive spaced practice, where information is committed to long-term memory by revisiting and repeating it multiple times with break in between.

When we teach reproducibility topics, we want students to do more than commit information to long term behavior. We actually want them to act on new information and induce a change in behavior by putting their knowledge into practice, both in the classroom and in their own work. Importantly and in contrast to many other data science topics, students often already have behavioral patterns in place for how they organize and name their files, or how they collaborate with their colleagues. Although these are not reproducible practices, they fill many of the same niches in a learner's workflow, e.g. we want students to replace naming documents `draft-final-version-3.md` with using git and GitHub for version control and substitute collaboration via forks and PR on GitHub for their current habits of using working together on Google docs. Therefore, teaching reproducibility ought to go one step beyond promoting effective retention of information and focus on habitual and behavioral changes,

which introduces additional challenges and opportunities.

Habit formation can be defined as the triggering of behavior from contextual cues. In this specific ... these contextual cues are manifested as the tasks that students desire to execute, such as saving a file after adding new content or wanting to share a document with a colleague. When teaching reproducible workflows we are aiming to replace the behavior with a more reproducible version as a response to the same cue. Although it might sounds like a complex task to not just unlearn an old behavior, but also learn a new one, studies has shown that behavioral change is in fact facilitated when substituting a desirable habit for an undesirable one existing rather than simply trying to unlearn the existing habit (Adriaanse, van Oosten, de Ridder, de Wit, & Evers, 2011).

Habitual behavior has been proposed to protect individuals from motivational lapses, where a desired good behavior is not expressed due to a momentary lack of willpower (TODO ref Review paper habit formation). By promoting the formation of habits in students, they opt for the reproducible workflow "by default," shielding them from relying on willpower to not "take the easy way out" and employ in a familiar, but irreproducible fashion workflow strategy.

Habits are best learned through frequent, regular, and sustained cue exposure. To support the formation of reproducible workflow habits, we therefore complement live demonstration with plenty of embed practice in the classroom, where we intentionally pause during the demos and say "okay students your turn, do what i just did." This is a more controlled form of practice, which sets students up for practicing these habits on their own. Another pedagogy that we use to complement live demonstration is guided worksheets. With these, we can provide students with many low stakes assessments comprised of short problems that are graded for completion only. In data science, this works well in literate code documents that have automated tests in them to provide feedback. Compared to in-class exercises, worksheets give learners additional change to actively engage with the material while still providing a structured way of providing exercises focused on key learning outcomes.

Importantly, habit formation is not a linear process. Instead, each successful action following a cue, adds to the formation of a new habit in an asymptotic fashion, where the initial events are the most important and the learning rate eventually plateaus as the habitual pattern solidifies. While popular literature often refers to ~30 days as the "all it takes," to develop new habits, studies have reported anything from and average of 60 to 100 days before reaching the plateau phase of habit development. We therefore believe it is paramount that learners have sustained frequent practice in reproducible workflows, which is interleaved with other topics where they would employ and benefit from these skills in real life.

To give students adequate time and practice to cement their reproducible workflow habits, we have taken intentional choices of which learning technologies and platforms are used throughout the program. This ensures that students

are practicing using reproducible tools for the full 10 months as part of the course learning technology mechanics (homework submission, grading, etc) and interleaved with learning other topics, just as they will do later in their carers (a detailed example follow in the next section). We also intentionally choose to use authentic data science reproducibility tools to provide opportunities for practicing and gaining confidence with these tools in "sharp" authentic scenarios rather than just in constructed exercises. This sustained practice not only enforces students' habits, but also increases their proficiency with these tools and they can run into problems in an environment where they can easily reach out for help without feeling intimidated to ask.

Example lesson(s) of lots of practice

A specific example of how we use lots of practice is used in almost all of the UBC Master of Data Science courses (20 one-credit courses taken over an eight-month period). In these courses we use version control, particularly Git and GitHub as our course management system. In these courses, the homework instructions and assignments are distributed to the students as GitHub repositories and the only way that they can submit their homework is by putting their homework in that GitHub repository. So they have to go through the cloning procedure, or at least be able to somehow download their assignments from the GitHub website, as well as be able to (hopefully through using Git) upload their work back to that GitHub repository.

To incentivize doing this using Git (as opposed to the GitHub web user interface) we also assign some marks of each assignment to a mechanics grade. For this we assess whether they have at least three commits associated with every single assignment and have written meaningful commit messages. We do this because we're trying to build these good habits and practices. There are good reasons why you use version control, not just to submit your homework but to active as a backup, for collaboration, or in case you want to go back in time and change things. By the end of this program the students have version controlled their work in over 80 different GitHub repositories. We hope this results in using version control becoming a habit, to the point that if they're going to work on a project it's going to go under version control by default - even when they leave the program and are no longer receiving grades for doing this.

One exciting technology that we have recently started incorporating in our teaching of reproducible workflows is GitHub actions. This tool has allowed us to automate the building of individual "playgrounds" of complex Git scenarios that would take much effort and typically fail to stage in a large classroom. One of the GitHub repositories that we created for these activities (https://github.com/ttimbers/review-my-pull-request) serves the purpose of providing a playground where students can explore and practice to learn how to use GitHub's code review feature for pull requests. To use it, students create their own copy of the repository on GitHub, create a branch named pr and then a

pull request is automatically created for them by a bot. After this simple setup, the students can spend the rest of the exercise exploring how to perform code reviews on GitHub.

Conclusions

We think over the past five years from teaching in the Master of Data Science and our Introduction to Data Science course that key things for teaching reproducibility in the data science classroom are providing extra emphasis on motivation, providing live demonstration so it's not a mystery of how you get started and what you need to do, as well as lots and lots of practice so that we can not only teach them the material and the concepts but so that this actually changes their practices and their workflows and they will use it after leaving your classroom.

References