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 UBC and in the undergrad data science courses that we're developing there is the study and development of reproducible and audible processes to obtain insight from data. When you go into the data science classroom students are usually very excited about learning data science but what they're most excited about is the second part of the definition, the insight from data part.

Often they are not even aware about the reproducible and audible processes part and they see that more as a pain/inconvenience. So you have this barrier when you're teaching the reproducibility aspects of data science. This probably arises because they likely do not even know what reproducibility is, and even if they do know about it, it is not the thing that is obviously/directly providing insight and so they're not excited about it.

Then we have this third challenge, which is that the tools that we use for reproducibility are not necessarily smooth and easy to learn. They usually have a pretty steep learning curve. Over our five years of teaching these things at UBC we've found some key things that we've experienced at least for teaching reproducibility successfully:

1. placing extra emphasis on motivation
2. direct instruction
3. lots of practice

In this paper, we will discuss why we believe each of these are important, give some high-level examples of how we do these, and then we give one detailed example of how we do each in our courses.

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 command until they find some things that work. Which can lead to users getting themselves into a lot of trouble. Sometimes this trouble is so difficult to get out of, that some professional data scientists and data science educators, for example Jenny Bryan, recommed the practice of "burning it all down" and starting from scratch - which really defeats many of the purposes of version control.

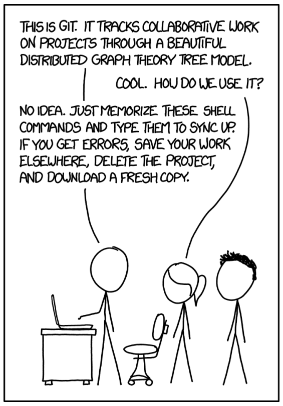
THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

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 implemenation of literate code documents (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.

```
# 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 useful for creating and sharing computational environments. ADD SENTENCE HERE WHY COMPUTATIONAL ENV's ARE IMPT FOR REPRODUCIBILITY. This tool 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.

```
# 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 I learned from Jenny Bryan's teaching of STAT 545 at UBC is to ask them 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, or controlled manner. Below we present this example in more detail, but in short though, 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, and if they cannot run it, fix it so they can run it. Then we provide them the same analysis that has been made reproducible. Activities like this, in a relatively short time with minimally painful experiences clearly illustrate to students the benefit of reproducibility, which helps provide motivation to endure the steep learning curves of reproducibility concepts and tools.

The authors of this work have experienced a lot of failure in graduate school and during their postdoctoral research in regards to reproducibility. They admit these failures slowed down down their research. This has provided them much motivation to teach and use reproducibility concepts and tools. However, most undergraduates, and new graduate students have not experienced this (and we question whether they should to its full extent in the next section). So if we can set up these scenarios where they feel a little bit of this pain but only for a short period of time that can be very useful for creating motivation.

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.

WE SHOULD FLESH THIS OUT HERE AND FIND 3-5 CASE STUDIES TO CITE THAT WE PLAN TO USE NEXT YEAR.

Example lesson of letting them fail (in a controlled manner)

An example we have used in the classroom is letting students 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. This course is a project-based course where we teach them all the concepts and skills needed to do a reproducible data analysis. 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 prequisite 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/command line. 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 Dockerfile that specifies how to build a Docker image). Most data science students do not come to the reproducibility classroom with this prequisite knowledge, and so there is a lot of barrier and stickiness to teaching this subject. It really needs motivation.

The 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) and in class, 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.

Direct instruction

We also claim that direct instruction is important when teaching reproducibility, why is that so? 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 direct instruction 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 direct instruction in the reproducibility classroom? We do a lot of live demos. In the data science programming classes it is becoming 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 demos with makes it more obvious to the students of how to use these things, and 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. These mistakes also can provide opportunities to spend more time on that area of the topic and explain why you made the mistake, and how to fix them.

The other direct instruction pedagogy that we use are guided worksheets and tutorials. As argued above, live demos are very useful, but you cannot demonstrate everything. Furthermore, it's good for people to work through and actively engage with material themselves. Guided worksheets strike a nice balance or providing practice in a very structured way.

A word of caution with direct instruction 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 and something or something has broken.

A relatively recent example of this is from fall 2020, when GitHub, which is the largest code hosting repository in the world, decided to (rightfully so) change the name of their default branch from master to main. This change caused a lot of things to break in 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 completly solved as we still have several resources that are still using 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 direct instruction 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 direct instruction

Here we provide an example of how we use direct instruction 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 best practice we argue for teaching reproducibility is lots of practice. Why do we need lots of practice to effectively teach reproducibility workflows and tools. This is likely due to the biological ways that humans are able to commit things to long-term memory, or build lasting changes in behaviours. There are really two fundamental ways that we commit things to long-term memory; one is one trial learning and repetitive spaced training. 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, don't need those things to be repeated multiple times so that you remember them. However, that is not how you learn most of the things that you learn at school. The things we typically learn about at school are not so emotionally salient, and are instead learned through repetitive space.

So the best way to commit something to long-term memory that's not really emotional is to revisit it and repeat it multiple times and have breaks between those things and so that lets you commit it to memory. However, we think you want to go even a step further with reproducibility because when we teach reproducibility workflows and skills as instructors, we actually want students to do more than just learn about these things. We actually want them to use them and put them into practice in the classroom, and outside of the classroom in their own work. And so we actually want to change their habits or behaviors. We think it's quite important to realize that it's not just understanding an algorithm, with reproducibility, it's it's understanding the concepts behind something like version control, knowing how to use those things and then once you leave the classroom wanting to and being able to use these things. Essentially, we want to get students to the point where they do these things out of habit, and they don't really have to think to hard about how to do it, nor think about other ways to do it.

One thing thing that can be done in the classroom to embed lots of practice in a complementary ways with live demos is to intentionally pause during the demos and say “okay students your turn, do what i just did,” and then give them time for the practice of typing it into the keyboard or clicking their mouse around the graphical user interface. Another way this can be done is through lots of low stakes assessments comprised of small or short problems. In data science, this works well in literate code documents that have automated tests in them to provide automated feedback. Another way lots practice can be provided is through the choice of learning technologies and platforms that are used in the course. Using authentic data science reproducibility tools as part of the course learning technology mechanics (homework submission, grading, etc) provides many opportunities for authentic practice.

Example lesson(s) of lots of practice

An example of how we use lots of practice in the classroom 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. 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. They become very practiced and very used to using version control. 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 - even when they leave the program and are no longer receiving grades for doing this.

In the UBC Master of Data Science program, we use a homegrown tool (rhomboid), to do this, but there are many tools that one could use to do this, including GitHub Classroom and Classy.

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 direct instruction 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