Ten Simple Rules for Teaching Data Science*

Tiffany Timbers

September 25, 2025

Introduction

Data science is the study, development, and practice of using reproducible and transparent processes to generate insight from data (Berman et al. 2018; Wing 2019; Irizarry 2020; Timbers, Campbell, and Lee 2022). With roots in statistics and computer science, data science educators use many of the teaching strategies used in statistics and computer science education (Carver et al. 2016; Zendler and Klaudt 2015; Fincher and Robins 2019). However, data science is a distinct discipline with its own unique challenges and opportunities for teaching and learning. Here I collate and present ten simple rules for teaching data science which have been piloted by leading data science educators in the community, and which I have used and tested in my own data science classroom with success.

Rule 1: Teach data science by doing data analysis

The first rule is teaching data science by doing data analysis. This means in your first lesson, not in your third lesson, not in your 10th lesson, not at the end of the semester, but in your first data science lesson, get the students to load some data, do some simple data wrangling and make a data visualization. Why do I suggest this? Because it's extremely motivating to students. In the beginnings, students have signed up for a data science course or workshop because they're interested in asking, and answering, questions about the world using data. They don't necessarily have enough knowledge to care deeply about detailed and technical things, such as object data types, whether one should use R versus Python, or if you are using R, whether you should use the tidyverse or base R. As a consequence, we should show them something interesting very early on, so we hook them. After they are hooked, they will be begging you to answer questions about the detailed, technical aspects you intentionally omiteed. An example of this is shown in Figure 1; code from the first chapter of *Data Science:*

*Corresponding author: tiffany.timbers@stat.ubc.ca.

A First Introduction (Timbers, Campbell, and Lee 2022). In this first chapter, we get the learners to load a data from a CSV, do some introductory data wrangling through filtering, arranging and slicing. Finally create a plot to answer a question about indigenous languages in Canada; how many people living in Canada speak an indigenous language as their mother tongue? Other leading data science educators which advocate and practice this rule include David Robinson in his introductory online Data Science course Robinson (2017b); Robinson (2017a)] and Jenny Bryan — who summed this up nicely in a tweet "[...] I REPEAT do not front load the 'boring', foundational stuff. Do realistic and nonboring tasks and work your way down to it." (2017)

```
library(tidyverse)

# load the data set
can_lang <- read_csv("data/can_lang.csv")

# obtain the 10 most common Aboriginal languages
aboriginal_lang <- filter(can_lang,
                          category == "Aboriginal languages")
arranged_lang <- arrange(aboriginal_lang,
                         by = desc(mother_tongue))
ten_lang <- slice(arranged_lang, 1:10)

# create the visualization
ggplot(ten_lang, aes(x = mother_tongue,
                     y = reorder(language, mother_tongue))) +
  geom_bar(stat = "identity") +
  xlab("Mother Tongue (Number of Canadian Residents)") +
  ylab("Language")
```

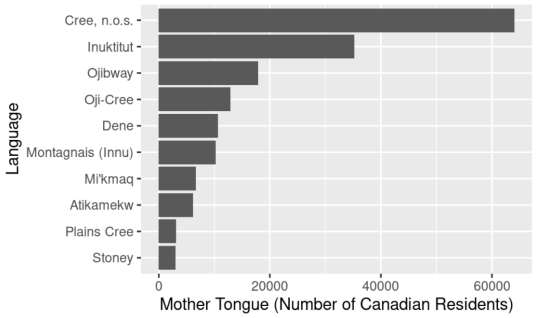

Figure 1: Example code from the first chapter of *Data Science: A First Introduction* (Timbers, Campbell, and Lee (2022)) that gets students doing data analysis on day one.

Rule 2: Use participatory live coding

The second rule is to use participatory live coding. This means when you are doing something with code in the classroom, instead of showing it in a static slide, or just running it in an executable slide or IDE, actually type the code out and narrate it as you are teaching. Have the participants follow along as well. The reason for this is it demonstrates your best practices for processes and workflows; topics that are important in practice but unfortunately often just an afterthought in teaching computational subjects. You can talk about why you're doing things in different ways as you're doing it You are also likely going to make mistakes as you live code — and that's actually a good thing. It helps you appear human to the students because they're going to make mistakes too. More importantly, it allows you to demonstrate how you do debugging to solve problems with code, which they'll be able to leverage in their homework and use later in their work outside the course. Participatory live coding also slows you down, so you don't go too fast for the students. This pedagogy originates from the "I do, we do, you do" method of knowledge transfer (Fisher and Frey 2021) and its use in teaching programming was pioneered by by the global nonprofit called The Carpentries (https://carpentries.org). Best practices for doing this has been refined and shared as ten quick tips by Nederbragt and colleagues (2020).

Rule 3: Give tons and tons of practice

The third simple rule is give tons and tons of practice. Give the learners many, many, many problems to solve, probably many, many, many more than you think that they might need. The reason for this is that repetition leads to learning (Ebbinghaus 1913). That's not just in humans, it is more fundamental than that. Looking at the field of animal behavior, across the animal kingdom, repetition leads to learning (Harris 1943; Shaw 1986). So students really need to do things many, many times to understand and then to perform those tasks. For example, when I am teaching students to read in data from a file, I don't just give them one file to read in, I get them to do this with six different variants of a very similar file. The students have to investigate each file in detail, including: look at what type of file it is, look at the column spacing, check if there's metadata to skip over, whether there are column names, *et cetera*. They will do these six variants in an in-class worksheet, as well as in a lab assignment, and then again on a quiz. Meaning, that by the end of the course they will have practiced this skill over 15 times. Many excellent data science educational resources use this pedagogy. Examples include the `{swirl}` R package, ADD 2 MORE EXAMPLES HERE. For those new to designing practice excercises for data science, I recommend looking at the "Exercise Types" chapter of *Teaching Tech Together: How to Make your lessons work and build a teaching community around them* by Greg Wilson (2019).

Rule 4: Give tons and tons of timely feedback

Pairing with giving lots of practice, you also want to pair that with tons and tons of timely feedback. And practice without feedback has limited value. So how can we give tons and tons of timely feedback? One way we can do this is through automated software tests. So in data science, a lot of the problems we're going to give students to do are working through code, and so we can write software tests that if a student gives a particular answer and it's wrong in this particular way, then we can give this feedback back to the student, and then they can maybe try again and solve the problem in a different way. Here's an example that we use in one of our courses where the students were given some ggplot code, and it was jumbled up in the wrong order, and they have to rearrange it. And in this rearranging it, the student is missing, somehow missing the plus sign to add another layer, and that ends up with a poorly formatted access type title. When they run our test, then they get some feedback that there's a problem with one of the access labels, so that will direct the students attention back to look at the, hopefully at the label code, they can fix it, and then they'll get a message that they were successful in fixing the code, and it's what we've expected. The first time that I had heard about doing something like this was, there's a wonderful literate, in addition to like different types of coding exercises in Greg's Teaching Tech Together book, there's also a section, or there's many sections where he's done a very thorough review of the literature, and in that review of the literature was this idea to do these automated testing. And there seems to be quite a convergence of this in the computer science field and coming into the data

science field, so we actually have lots of tools, open source tools now to do this. So LearnR is a package you can do this with in the R programming language, and then there's NBgrader and Ottergrader, which are language agnostic tools, or at least work with R and Python for providing these kinds of feedback to students on their practice problems.

Rule 5: Use tractable or toy data examples

My fifth simple rule is to use tractable or toy data examples. Rule number five is to use tractable or toy data examples. And so what do I mean by this? So when you're introducing a new tool, a new method, a new algorithm to students, you give them data to learn this that has a countable number of things, things that will kind of fit inside of our working memory, so students can track where everything is going through different steps of the algorithm. And this allows students to see how the elements are manipulated and really get the intuition for these things. An example from one of our courses, the introduction in data science course, where we teach k-means clustering, we use the polymer penguins data set to introduce that, but instead of throwing that entire data set that has hundreds of observations at students, what we do initially is we subset that to just a handful of observations, like shown here on the screen. And then we can walk the students through what happens to these observations at each step of the algorithm. And students can very clearly see from step one, to step two, to step three, to step four, what is happening with those observations and build an understanding and an intuition. Inspiration from this comes from Jenny Bryan's great dplyr join cheat sheet that she created in the STAT 545 course at UBC. And so in this example, she's teaching all the different joins from the dplyr package. There's left joins and right joins and fold joins and inner joins and semi joins. And depending on the direction, you might end up getting a different result. And so to help students understand these things, it's really hard to get an idea of what's going on and all these things that are also similarly named when you have really big data sets. So she created an example cheat sheet where there's only like six rows in one data set, three rows in another data set. And then she goes through all the different possible joins and shows the output of it. And so as a learner looking at this becomes, this becomes very tractable and easy to understand.

Rule 6: Use real and rich, but accessible data sets

Now, after you've reached, after you've used a toy or tractable data set, and the students have found conceptual understanding of the new method or algorithm you're teaching them, the next thing I think to do is to use a real and rich data set paired with like a real question. But as you're doing this, make sure that that data set is also accessible. So what do I mean by that is that you want to have a data set where the question is something that a real data scientist, either in academia, industry, government would really want to answer. It's a larger data set so that it's not just so easy that you would be able to come up with the answer by eye. That's

really motivating. But the catch here is that the observations in the data set, the different variables, the different columns in the data set have to be things that very quickly your learners can understand. And really, what can become kind of like an expert blind spot for us when we're teaching this is especially if we come from like a particular domain. So for myself, from like the neuroscience or biology domain, I might, for example, think that using some deep sequencing data data set might be a great example for this particular algorithm that I want to teach the students. However, that's because I have this, you know, deeper understanding of biological processes that you need to understand that data set. And depending on your learners, that data set might not be appropriate. So what you really want is you don't want that so much of the students cognitive resources spent trying time and energy trying to understand just what the data set is about, and then mapping that to the question. And then you see, it just gets to be a bit too much. So instead, we want to use something where people understand what the observations are and what the columns are quite easily. So one example here from our introductory data science course is a data set from the can laying our package, where we have, you know, number of Canadians reporting a particular language as their mother tongue. So the language that they learned at birth, and then we have different categories. So they in the data set, they use Aboriginal language as a category, it really probably should be Indigenous languages versus official languages versus non official and non Indigenous languages. Inspiration from this game, again, from Jenny, Brian. So she created the Gapminder R data package that's used quite widely in teaching R in many universities. And it's really nice because it's a biggish data set. So there's, you know, hundreds of observations in it. And the observations though are, you know, something most people understand. It's a country in a given year. And, you know, things that we're actually interested in, because everybody grew up in a country or more than one country in their lifetime. And we all grew up at a different time in history. And so those things make us generally maybe interested about asking questions with the data set. Also, the variables in the data set are country, continent, year, life expectancy, population, per capita GDP. These are all concepts that we as adults have an understanding of without having to have a deep explanation about it. So it makes it really accessible. So I've listed here some more explanations of data sets or data packages that might fit that. But the biggest thing is keeping in mind that, you know, it doesn't take a lot of time for your students to understand the data set. Because that's, you know, the data set isn't necessarily what you're trying to teach at that moment.

Rule 7: Provide cultural and historical context

My seventh simple rule is to provide a cultural and historical context for what you're teaching. And what this kind of like, what do you mean by that? What I'm talking about is sometimes when we're teaching things in software, or with software, things might seem weird or odd or strange. And it's really helpful to say why they are that way. If you give the history for why, and even tell students that like, this was a design choice, people thought about this and decided this was the best way to implement this, particularly, you know, maybe for reason X, Y, and

Z. It helps students understand, A, that, you know, tech software is built by humans, and so it's going to be influenced by humans' perspective, human history, and human culture. And it helps prevent frustration or annoyances with the software. They can start to see the reason behind that. And why I think it's important to help prevent those frustration or annoyances, because I think for some people, they can become real blockers and make people maybe like dislike or written up a certain piece of software, for example. So an example for this is like when I'm teaching the programming language R with the tidyverse, where you see something that's quite different from a lot of other programming languages. So what is that? It's the heavy use of unquoted column names. So you think about anytime you're accessing a data frame column with the tidyverse, you don't put quotations around it. That's actually really strange when you come from other programming languages and actually can make, you know, writing functions with the tidyverse a bit more difficult. And so some people coming, say, from Python or from, you know, Java or C coming into this situation might be like, that's a really dumb idea. But if they understand that, you know, R was written by statisticians for statisticians and like with the intent that many, much of the time would be typing things into the console or running code interactively and not having to like always keep track of opening and closing those quotes and making mistakes, trying to minimize that, then students can be like, oh, I understand why that is a good design choice given that situation. Inspiration for this rule came from Colin Fay. He's a data scientist at Think R and he wrote a really great post on why in R we use the arrow as an assignment operator. So he talks about that. And this can be very, from my experience, teaching students R can be very confusing. He talks about like how R comes from another programming language named S and then actually S was inspired by another programming language type by called APL. And APL was designed for a particular keyboard that had one key that made the assignment operator. When you see, when you, when you know that, then it makes a little, it makes a lot more sense as to like why that design choice may have been made at the beginning. And maybe a bit more understanding and less frustration and maybe kind of digging into like, what does it mean now? And do we have to use it? And those sorts of things. If you're interested in learning more about the history of at least the R program language, Roger Pang has a great section on it in his R programming data science book.

Rule 8: Build a safe, inclusive and welcoming community

Number eight is build a safe, inclusive and welcoming community. And I actually think, you know, practicing this talk a couple of times and giving it now, I might move this to number one. This is probably the first thing you need to do when teaching in general at all, but I'm talking about data science, so I'm going to include it here. And what do I mean by this? It means that as the instructor, you should be putting in place scaffolding and guidelines to facilitate a safe learning environment. And that's your responsibility. And the reason for that is that students can't learn effectively if they don't feel safe. So if they don't feel safe to ask the question without being made look dumb, they're not going to ask that question, they're

not going to be curious. If students don't feel safe to show up in the classroom because of maybe how they are talked to, you know, just out even just in the hallway, or in, you know, on the course forum, for example, that's, that's not setting up a good learning environment. So all of that kind of needs to be in place before you can have real expectations of student learning. It's a big responsibility, but it's one that we do bear as course instructors, I think. So one example of what we, it's not the only thing we do, but one thing that we do in our courses at UBC, or at least my courses at UBC, is that we have a code of conduct for each course. And that code of conduct holds a place of prevalence in the classroom. So we make time in the classroom to be like, here is our code of conduct. And we're going to highlight parts of it for you at the beginning of the course. We talk, it's very explicit. It talks about expected behavior, behaviors that will not be tolerated. It talks about what is the process for reporting something that violates the code of conduct, and who do you talk to? And if it's the instructor who violates the course of conduct, who do you talk to then? And all of this, I think, is, is really important for at least one piece of scaffolding to help students feel safe. Inspiration for me came from the carpentry's code of conduct. So for all of their workshops, they take this very seriously, and they apply that not just to their workshop setting, but even to with all of the spaces that are considered under the carpentry. So, you know, when instructors are getting together, working on material, at conferences, anywhere, their online discussion forums, everything is covered by this code of conduct.

Rule 9: Use checklists to focus and facilitate peer learning

Number nine for the simple rule is using checklists to focus and facilitate peer learning. So the pedagogical literature suggests that, you know, students, we can harness these communities of learning by having peers learn from each other. And peer review is one way that that can happen. However, peer review can be really hard and difficult if you haven't done it before, or if it's reviewing something new that you're just learning. It's hard to know, like, what should I even be looking at? So what we can do as instructors, we're used to making rubrics for these things. So we can kind of hijack those rubrics and come up with review checklists. So here's an example of a data analysis checklist, sorry, a data analysis review checklist that we've generated for the master of data science program courses. And what it does is it gives kind of the students like a list of things to focus in on, you know, we think are important for that assessment being completed to high quality. And then when we ask students to give comments and feedback, like written feedback, in addition to this checklist, they can focus their feedback in on the things they didn't check off. Right. So if there was issues with tests, for example, and issues with the conclusions, they would not check those boxes. And then they would, in their review comments, really be able to know that that's probably what I should spend my time telling them about. Inspiration for this came from the R OpenSci organization. So this is an organization that reviews R packages, does peer review and publishing of R packages, and they use checklists for their reviewers. I've acted as a reviewer for this organization. And I can say that I personally found reviewing for that far easier than maybe I found reviewing for

journals that didn't have these types of checklists. Because again, it like, it's a great way for editors or publishing organizations to communicate to the reviewers, like what is important to them for their publication? What is really valuable? And it really helps focus your reviewer. This is becoming more and more prevalent in the publishing world. So the Journal of Open Source Software, the Journal of Open Source Education used this. There's now a PI OpenSci, Python package organization. And I think there's real value in these things in the classroom and outside.

Rule 10: Have students do projects

And finally, my last item on the checklist is to have students do projects. So what I mean is have students perform a data analysis project on a topic, ideally, if they're choosing if you can make it happen from beginning to end. And, you know, this sometimes can feel daunting as an instructor, depending on how many teaching resources you have or how much time you have for grading. So that's why I put the word scoped here. You may have to say that the project has to be about this particular topic, has to use this particular data set, or it has to use this particular method, this particular programming language. So you can have some homogeneity for grading and making a rubric that's going to apply for everybody. But that can help make things scalable. And why do we think this is important? Well, I think it really helps provide students with motivation. We get lots of good feedback about courses that have projects because they have this flexibility and choice aspect that students really like. But it also gives students, you know, good experience with messiness of real data. And we all know that in the real world, data is quite messy. So here's an example of our project from a collaborative software development course that I teach. And in this project in particular, the students have decided to make an R package that extracts and analyzes song lyrics. And so they go all the way from taking the raw data and coming up with outputs from that raw data and all of the software and all of the version control and building of an R package that went into that. The inspiration from this comes from STAT545, which is a course at UBC that Jenny Bryan created that was made before the Master of Data Science program. And in the classroom, Jenny would give lots of examples of teaching, you know, the new concepts using that kind of like more tractable data set, still rich and interesting. So the gap minor data set. But then the students homework was actually wrapped up into a project, which was working on like what something related to their thesis project. And so they got to be working in their discipline, but applying everything that they were learning in the classroom to that throughout the entire semester and wrapping up in a final project at the end of the year.

Conclusion

So I'm going to now quickly run through these 10 simple rules that I or remind you of these 10 simple rules that we just went through for teaching data science. So the first one, teach data

science by doing data analysis and do that on day one. Use participatory-led coding. So have the students watch you do things and have them do it with you. Give tons and tons of practice. And along with that, give tons and tons of feedback. Leverage, if you're using software in the classroom, leverage the automated software test is one way to do this. Use tractable or toy data examples to build intuition, to build concepts. Once they have those concepts, add on real rich data sets that are more motivational and more real world, but keep them accessible. Provide cultural and historical context to the software and the tools that you're using to avoid frustration and blockers. Build a safe and inclusive welcoming community. Again, that should be number one. Use checklist to focus and facilitate peer learning and have students do projects. Have them get an idea, get a sense of what the entire workflow is from beginning to end. And as I said, I'm standing on the shoulders of many other people giving this talk. I'm really just curating a list of good things I've seen in the data science education community that are already being practiced by so many other people.

Berman, Francine, Rob Rutenbar, Brent Hailpern, Henrik Christensen, Susan Davidson, Deborah Estrin, Michael Franklin, et al. 2018. "Realizing the Potential of Data Science." *Communications of the ACM* 61 (4): 67–72.

Bryan, Jenny. 2017. ""So True: Do Not i REPEAT Do Not Front Load the 'Boring', Foundational Stuff. Do Realistic and Nonboring Tasks and Work Your Way down to It."." Twitter.

Carver, Robert, Michelle Everson, John Gabrosek, Nicholas Horton, Robin Lock, Megan Mocko, Allan Rossman, et al. 2016. "Guidelines for Assessment and Instruction in Statistics Education (GAISE) College Report 2016."

Ebbinghaus, Hermann. 1913. *Grundzüge Der Psychologie v. 2, 1913*. Vol. 2. Veit.

Fincher, Sally A, and Anthony V Robins. 2019. *The Cambridge Handbook of Computing Education Research*. Cambridge University Press.

Fisher, Douglas, and Nancy Frey. 2021. *Better Learning Through Structured Teaching: A Framework for the Gradual Release of Responsibility*. ASCD.

Harris, J Donald. 1943. "Habituatory Response Decrement in the Intact Organism." *Psychological Bulletin* 40 (6): 385.

Irizarry, Rafael A. 2020. "The Role of Academia in Data Science Education." *Harvard Data Science Review* 2 (1). https://doi.org/10.1162/99608f92.dd363929.

Nederbragt, Alexander, Rayna Michelle Harris, Alison Presmanes Hill, and Greg Wilson. 2020. "Ten Quick Tips for Teaching with Participatory Live Coding." *PLOS Computational Biology* 16 (9): e1008090.

Robinson, David. 2017a. "Announcing "Introduction to the Tidyverse", My New DataCamp Course." Variance Explained (blog). http://varianceexplained.org/r/intro-tidyverse/.

———. 2017b. "Introduction to the Tidyverse." DataCamp Online Course. https://www.datacamp.com/courses/introduction-to-the-tidyverse.

Shaw, GL. 1986. "Donald Hebb: The Organization of Behavior." In *Brain Theory: Proceedings of the First Trieste Meeting on Brain Theory, October 1–4, 1984*, 231–33. Springer.

Timbers, Tiffany, Trevor Campbell, and Melissa Lee. 2022. *Data Science: A First Introduction*. Chapman; Hall/CRC.

Wilson, Greg. 2019. *Teaching Tech Together: How to Make Your Lessons Work and Build a*

Teaching Community Around Them. Chapman; Hall/CRC.

Wing, Jeannette M. 2019. "The Data Life Cycle." *Harvard Data Science Review* 1 (1). https://doi.org/10.1162/99608f92.e26845b4.

Zendler, Andreas, and Dieter Klaudt. 2015. "Instructional Methods to Computer Science Education as Investigated by Computer Science Teachers." *Journal of Computer Science* 11 (8): 915–27. https://doi.org/10.3844/jcssp.2015.915.927.