Ten Simple Rules for Teaching Data Science*

Tiffany Timbers Mine Çetinkaya-Rundel

October 2, 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 we collate and present ten simple rules for teaching data science which have been piloted by leading data science educators in the community, and which we have used and tested in our 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 we 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 fig-intro-ds-code; code from the first chapter of

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

Data Science: 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." (2017b)

```
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")
```

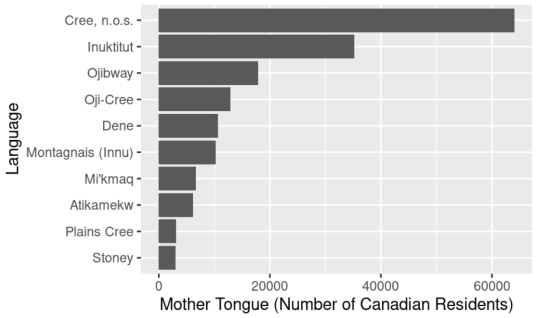

Figure 1: Example code from the first chapter of *Data Science: A First Introduction* (Timbers, Campbell, and Lee (2022)) that gets students doing data analysis on day one.

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 and timely feedback

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 teaching students to read in data from a file, don't just give them one file to read in, get them to do this with six different variants of a very similar file. With this approach 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*. In our courses, 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, including software packages (e.g., the `{swirl}` R package (Carchedi et al. 2023)), online courses (e.g., Kaggle Learn (Kaggle 2018)) and popular text books (e.g., R for Data Science (Wickham, Grolemund, et al. 2017)). For those new to designing practice excercises for data science, We 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).

When giving lots of practice, you also want to pair that with tons and tons of timely feedback. Practice without feedback has limited value. So how can we give tons and tons of timely feedback, especially with our limited teaching capacity and resources? One way we can do this is through automated software tests. In data science, a lot of the problems we give students involves writing code. As a consequence, can write software tests that act as feedback for the students; letting them know when they give a wrong answer that it's wrong in this particular way, as well as a gentle and helpful nudge to solve the problem in a different way. Figure Figure 2 shows an example of this in practice. Here students were given some ggplot code in a Parson's problem format (the lines of code were given in the wrong order, and the students need to rearrange to the correct order). In this example, a student has rearranged the code, but not quite correctly, and so a plot is created, but it's not quite what we expect. Without timely feedback, the student might not realize that there's a problem with their code until much later, or not at all if they fail to check the feedback and solutions after the assignment is graded and grades are returned (often days or weeks later). The use of automated software tests can provide that timely feedback while students focus and attention are on the topic being learned and practiced. This pedagogy was first developed and used for teaching programming in computer science courses (reviewed in Wilson 2019) and is now being adopted in data science courses. There are now many wonderful and popular software pacakges to do this in the context of data science, including the `{learnr}` R package (Kross, Çetinkaya-Rundel, et al. 2024) for R code, and NBgrader (Hamrick et al. 2016) and Otter Grader (Kim et al. 2022) software packages that work for both R and Python code.

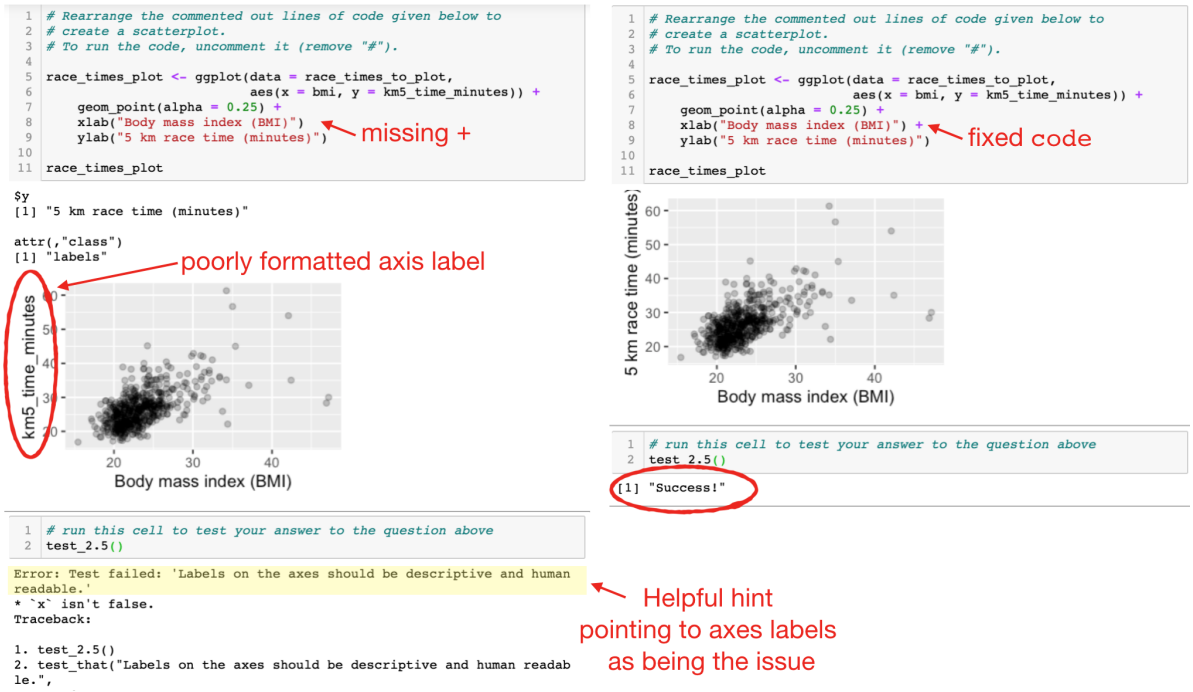

Figure 2: Example of automated software test feedback to students.

Rule 4: Use tractable or toy data examples

Our fourth simple rule is to use tractable or toy data examples when introducing a new tool, a new method or a new algorithm to students. tractable or toy data sets have a countable number of things, things that will fit inside of our working memory. This allows students to track where everything is going through different steps of the algorithm, to see how the elements are manipulated and really get the intuition for these things. For example, in one of our courses we use the Palmer penguins data set [horst2020palmerpenguins] to introduce the students to k-means clustering. And instead of giving them the entire data set, which has hundreds of observations, we first subset the data to just a handful of observations. Then we can walk the students through what happens to these observations at each step of the algorithm, and build an understanding and an intuition for the algorithm. Inspiration from this comes from Jenny Bryan's great `{dplyr}` joins cheat sheet (Bryan 2015). In her cheat sheet, she teaches all the many different joins from the `{dplyr}` (Wickham, François, et al. 2024) package. This is a difficult topic for students to understand and memorize, and when instructors teach this with large data sets it is really hard for students to get an idea of what's going on and all these joins that are also similarly named (e.g, left join, right join, inner join, outer join, etc). To address this issue, Jenny's cheat sheet uses two toy data sets on super hero comic characters and publishers. The super hero data set has only 7 rows and 4 columns, while the publisher data set has 3 rows and 2 columns. These data sets are small enough

The cheat sheet thenb goes through all the different possible joins and narrates and shows the output of each. For learners this becomes much more tractable and easy to understand.

Rule 5: Use real and rich, but accessible data sets

After you've helped students reach conceptual understanding of the new tool, method or algorithm that you are teaching them, the next thing to do is to get students to apply it with realistic question and a real and rich data. However, as you're doing this, it is critical to ensure that that data set is also accessible to all your learners. The question, as well as the the observations (i.e., rows) and variables (i.e., columns) in the data set, have to be things that your learners can very quickly understand. This can easily become an expert blind spot for us when we are teaching, especially if we have training in a particular domain. For example, one of us authors, who was trained in the biological sciences domain, might think that using a deep sequencing data set might be a great and motivating example for a particular algorithm that we want to teach the students. However, this thought likely stems from that author's deeper understanding of biological processes that one needs to understand that data set. And depending on your learners, that data set might not be appropriate. It might very well introduce too much cognitive load for the students, so much so that they cannot focus on the task at hand — refining their understanding and applicaiton of whichever tool, method or algorithm you are teaching them. We do not want to use up our students' limited cognitive resources just to understand what the data set is about. Instead, we want to use something where all our learners quite easily understand what the observations are and what the columns are. One example from one of the courses that we teach uses Canadian census data about the languages spoken in Canada as a person's mother tongue, at work, and at home, across different regions (Timbers 2020). Another example, that is more widely used in teaching data science, is the Gapminder data set (Bryan 2017a). This is a really nice example because there are hundreds of observations in it, however the observations are something most people understand; a country, a year, a population, etc. And with such a data set, we can ask questions that most learners are interested in, because everybody grew up in a country, or more than one, in their lifetime. And we all grew up at a different times in history. These lived experiences make us all generally knowledgeable and interested about asking questions with the data set. There are many more data sets that are real and rich, but also accessible that can be used. The main point here to keep in mind is that it should not take a lot of time for your students to understand the data set because a deep understanding of the data set is not what what you are trying to teach at that moment.

Rule 6: Provide cultural and historical context

Our sixth simple rule is to provide a cultural and historical context for what you're teaching. For example, when we are teaching how to use a new software tool, or a new feature of a

tool students already know how to use, and things do not seem optimally designed from the learner perspective, it's really helpful to explain why they are that way. If you give the design and historical context, for example, saying people thought about this when they built the tool and decided this was the best way to implement it for reasons X, Y, and Z, it helps the students understand that software tools are built by humans, and so they are going to be influenced by humans' perspective, human history, and human culture. Furthermore, it helps prevent frustration or annoyances with the software because they can rationalize why it works a certain way. We think it is really important to help prevent those frustration or annoyances, because we have observed that for some learners, they can become real blockers and can lead to learners disliking or avoiding a certain piece of software.

For example, when we are teaching the programming language R (Ihaka and Gentleman 1996) with the suite of `{tidyverse}` R packages (Wickham, Hester, et al. 2024). learners observe the functions from these packages heavily use unquoted column names when referencing data frame columns in their functions calls. That is really strange when you come from other programming languages, as most other programming languages require quoted strings when referencing objects attributes (which is what a data frame column is). This can also make writing functions that use the tidyverse functions a bit more difficult because of issues with indirection. For learners who have experience with Python, Java or C, they might initially view this as a really bad idea. However, once they learn that R and the suite of `{tidyverse}` R packages were written by statisticians for performing data analysis and graphics (Ihaka and Gentleman 1996; Wickham, Hester, et al. 2024), and that they designed the language and packages with the expectation that much of the users time would be typing things into the console or running code interactively, it makes a lot more sense that they would want to minimize the amount of typing and tracking of opening and closing quotations for their users.

Another example comes from the use of the arrow (`<-`) as an assignment operator in R, which may to some learners seem odd as it less efficiently uses two characters. Again however, when given the historical context that R was derived from another programming language named S, and S was inspired by another programming language, APL, that was designed for a particular keyboard with one key mapped to the assignment operator (Fay 2018), it makes a lot more sense as to why that design choice was made. Design and historical contexts helps learners understand the rationale behind differnt design choices, and allow them to see design choices that they might not initially agree with as an excellent design choice with these added contexts. As an aside, for those interested in learning more about the history of the R program language, see the "History and Overview of R" chapter in Roger Peng's *R programming data science book* (2016).

Rule 7: Build a safe, inclusive and welcoming community

The next rule is to build a safe, inclusive and welcoming community. If we were to rewrite this paper, we would probably move this to rule number one given this is the first thing you need to do when teaching any subject. The reason for this is that students can't learn effectively if they don't feel safe. For example, if they don't feel safe to ask the question without being made to look or feel dumb, then they are not going to ask the question [REFERENCE], they are not going to be curious [REFERENCE]. If students don't feel safe against discrimination or harassment in the classroom or related spaces (e.g., office hours, course forums, study groups, etc) they will stop showing up in those places, or if they do show up, they will not be able to be engaged in the learning [REFERENCE]. Creating safe, inclusive and welcoming learning environments is critical for learning, and we as instructors have a responsibility to put in place scaffolding and guidelines to make this happen. One thing that we do in our courses is to have a course code of conduct, and that code of conduct holds a place of prevalence in the classroom and related learning spaces. At the beginning of a new course we take time in the first class to present our code of conduct to the learners. The codes of conduct we use are very explicit. They talk about expected behavior, behaviors that will not be tolerated, the process for reporting violations, and the consequences for violations. We also make sure that there are multiple ways to report violations to ensure students are supported in the unexpected and rare case that the instructor is the one who is violating the code of conduct. It is also important when a violation is reported that the report is taken seriously and acted upon in a timely manner. No student concern should ever be ignored or brushed off. For instructors looking to create a code of conduct for their course, we recommend looking at existing codes of conduct from other organizations and adapting them to your own context. One particularly good one is the code of conduct from The Carpentries (https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html).

Rule 8: Use checklists to focus and facilitate peer learning

Rule number eight is to use checklists to focus and facilitate peer learning. A well documented pedagogical best practice is to have peers learn from each other [REFERENCE]. One implementation of this is peer review. 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 are still learning. So what we can do as instructors to facilitate this practice for our learners? Given that we are well practiced with making rubrics for grading student work, we can use our rubrics as a starting point to draft peer review checklists. Why checklists? Checklists can help ensure complex tasks are completed successfully and have been successfully used in safety critical systems (like aviation, surgery, or nuclear power). They can be particulary helpful in tasks that are complex, as well as those that are repetitive, and as a consequence, boring (Gawande 2010). For these reasons, checklists have recently been adopted in scientific (e.g., PLoS journals, Nature Ecology & Evolution, Journal of Open Source Education, Journal of Open Source Software)

and software (e.g., ROpenSci, PyOpenSci) publishing to help ensure that reviewers and editors increase transparency, decreases bias and call attention to essential elements of reviews that are often overlooked (Parker et al. 2018). As reviewers for some of these journals and organizations, the authors have personally found checklists to be extremely helpful in focusing our attention on the most important elements of a review, as well as helping us ensure we don't miss attending to anything. We think this idea has value in training data science students as well. Our Section shows an example of a data analysis review checklist that we have used in our data science courses. It serves to communicate the aspects we, as educators, think are important for that assessment being completed to high quality. In addition to checking off the checklist items, students are also asked to provide written comments and feedback. This checklist helps to focus students' comments and feedback on the things they were not able to check off. For example, if there was issues with the software tests (e.g., they were missing) and issues with the discussion section (e.g., they did not mention any limitations of the anlaysis), they would not check those boxes and that would help focus their review comments on critical feedback about the issues with those sections.

Rule 9: Teach students to work collaboratively

- Explicitly teach them best technical (version control tools) and social practices for collaborative work
- get the students to work collaboratively (in class activities like think-pair-share and pair programming, group assignments & projects)
- Examples we use with projects:
  - ice breaker activities to help with team formation (e.g., I used a bunch from Atlassian this past year and they were great, here's an example slide deck I put together: https://docs.google.com/presentation/d/14hojoEdOW_sdGSid-FuYZ4EsNKBXsjCdsmIbmW1gEiLs/edit?usp=sharing)
  - teamwork contracts
  - teamwork reflection activities

Rule 10: Have students do projects

Our final simple rule is to have students do projects — create an entire data product (ideally, on a topic of their choosing if feasible) from beginning to end (or from "nachos to cheesecake" to quote Jenny Bryan). Projects can be done individually or in groups. The main point is that students get to experience the entire data science workflow. Doing project work is important because it really helps provide students with motivation. It also gives students good experience with messiness of real data (we all know that in the real world data is quite messy). Additionally, courses typically focus on just a small part of a data analysis workflow.

For example, they might focus on data visualization, or data wrangling, or modeling. But in a project, they get to see how all these pieces fit together in a more realistic scenario. This is critical training for any aspiring data scientist.

From the instructors perspective however, projects can feel daunting, particularly when student numbers are large and teaching resources are not. One way to make projects more feasible is to have them be scoped. You may want to limit the project to being about a particular topic, or get students to choose a data set from a given list, or a have them use a particular method, or a particular programming language. Doing this affords some homogeneity for grading and allow you to create a rubric that's going to apply to all projects. An example of a scoped project from a course on collaborative software development in data science that we teach asks students to create a Python package with n functions, where n is the number of students in the project group. The functions must be related to a common theme, and fall under the umbrealla of data science. The students must use the packaging tools and collaborative practices taught in our course. This kind of project is very feasible to grade because the scope is well defined and limited. But it also allows students to be creative and to work on something that they are interested in. Another way to scope projects is to have students work on projects related to their own research or thesis work. This is particularly relevant for graduate students who are taking data science courses as part of their degree program. For example in the UBC STAT 545 course developed by Jenny Bryan, students learn new data science concepts and skills using a tractable data set in the classroom (e.g., the Gapminder data set), but their project encompasses applying the newly acquired concepts and skills to their own research or thesis work. Again, this approach makes grading more feasible and students get to work on something they are deeply interested in and motivated to do.

Conclusion

This list of ten simple rules for teaching data science is by no means exhaustive, but we hope it provides a useful starting point for new data science educators. This list was curated from our own experiences teaching data science, as well as from what we've seen being practiced by other leading data science educators.

Supplementary Materials

Data analysis review checklist

Reviewer:

Conflict of interest

☐ As the reviewer I confirm that I have no conflicts of interest for me to review this work.

Code of Conduct

☐ I confirm that I read and will adhere to the MDS code of conduct.

General checks

☐ **Repository:** Is the source code for this data analysis available? Is the repository well organized and easy to navigate?
☐ **License:** Does the repository contain a plain-text LICENSE file with the contents of an OSI approved software license?

Documentation

☐ **Installation instructions:** Is there a clearly stated list of dependencies?
☐ **Example usage:** Do the authors include examples of how to use the software to reproduce the data analysis?
☐ **Functionality documentation:** Is the core functionality of the data analysis software documented to a satisfactory level?
☐ **Community guidelines:** Are there clear guidelines for third parties wishing to 1) Contribute to the software 2) Report issues or problems with the software 3) Seek support

Code quality

☐ **Readability:** Are scripts, functions, objects, etc., well named? Is it relatively easy to understand the code?
☐ **Style guidelides:** Does the code adhere to well known language style guides?
☐ **Modularity:** Is the code suitably abstracted into scripts and functions?
☐ **Tests:** Are there automated tests or manual steps described so that the function of the software can be verified? Are they of sufficient quality to ensure software robsutness?

Reproducibility

- ☐ **Data:** Is the raw data archived somewhere? Is it accessible?
- ☐ **Computational methods:** Is all the source code required for the data analysis available?
- ☐ **Conditions:** Is there a record of the necessary conditions (software dependencies) needed to reproduce the analysis? Does there exist an easy way to obtain the computational environment needed to reproduce the analysis?
- ☐ **Automation:** Can someone other than the authors easily reproduce the entire data analysis?

Analysis report

- ☐ **Authors:** Does the report include a list of authors with their affiliations?
- ☐ **What is the question:** Do the authors clearly state the research question being asked?
- ☐ **Importance:** Do the authors clearly state the importance for this research question?
- ☐ **Background**: Do the authors provide sufficient background information so that readers can understand the report?
- ☐ **Methods:** Do the authors clearly describe and justify the methodology used in the data analysis? Do the authors communicate any assumptions or limitations of their methodologies?
- ☐ **Results:** Do the authors clearly communicate their findings through writing, tables and figures?
- ☐ **Conclusions:** Are the conclusions presented by the authors correct?
- ☐ **References:** Do all archival references that should have a DOI list one (e.g., papers, datasets, software)?
- ☐ **Writing quality:** Is the writing of good quality, concise, engaging?

Estimated hours spent reviewing:

Review Comments:

Please provide more detailed feedback here on what was done particularly well, and what could be improved. It is especially important to elaborate on items that you were not able to check off in the list above.

Attribution

This was derived from the JOSE review checklist and the ROpenSci review checklist.

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. 2015. *STAT 545: Data Wrangling, Exploration, and Analysis with R*. https://stat545.com/.

———. 2017a. *Gapminder: Data from Gapminder*. https://CRAN.R-project.org/package=gapminder.

———. 2017b. ""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.

Carchedi, Nick, Sean Kross, Bill Bauer, et al. 2023. *Swirl: Learn r, in r*. https://swirlstats.com.

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.

Fay, Colin. 2018. "Why Do We Use Arrow as an Assignment Operator?" https://colinfay.me/r-assignment/.

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.

Gawande, Atul. 2010. *Checklist Manifesto, the (HB)*. Penguin Books India.

Hamrick, Jessica B. et al. 2016. "Nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook." In *Proceedings of the 19th Python in Science Conference*, 68–74. https://doi.org/10.25080/Majora-629e541a-00e.

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

Ihaka, Ross, and Robert Gentleman. 1996. "R: A Language for Data Analysis and Graphics." *Journal of Computational and Graphical Statistics* 5 (3): 299–314.

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.

Kaggle. 2018. "Kaggle Learn." https://www.kaggle.com/learn.

Kim, Eric J., Sam Lau, Josh Hug, and John DeNero. 2022. "Otter: A Tool for Automated Grading of Jupyter Notebooks and More." In *Proceedings of the 23rd Python in Science Conference*, 120–27. https://doi.org/10.25080/majora-2127ed6e-00c.

Kross, Sean, Mine Çetinkaya-Rundel, et al. 2024. *Learnr: Interactive Tutorials for r*. https://rstudio.github.io/learnr/.

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.

Parker, Timothy H, Simon C Griffith, Judith L Bronstein, Fiona Fidler, Susan Foster, Hannah Fraser, Wolfgang Forstmeier, et al. 2018. "Empowering Peer Reviewers with a Checklist to Improve Transparency." *Nature Ecology & Evolution* 2 (6): 929–35.

Peng, Roger D. 2016. *R Programming for Data Science*. Leanpub Victoria, BC, Canada.

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. 2020. *Canlang: Canadian Census Language Data*. https://ttimbers.github.io/canlang/.

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

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2024. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, Garrett Grolemund, et al. 2017. *R for Data Science*. Vol. 2. O'Reilly Sebastopol.

Wickham, Hadley, Jim Hester, Lionel Henry, et al. 2024. *Tidyverse: Easily Install and Load the 'Tidyverse'*. https://CRAN.R-project.org/package=tidyverse.

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.