Ten Simple Rules for Teaching Data Science*

Tiffany Timbers Mine Çetinkaya-Rundel

September 29, 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")
```

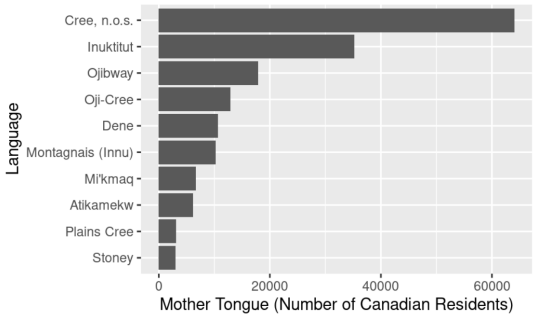

Figure 1: Example code from the first chapter of *Data Science: A First Introduction* (Timbers, Campbell, and Lee (2022)) that gets students doing data analysis on day one.

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.

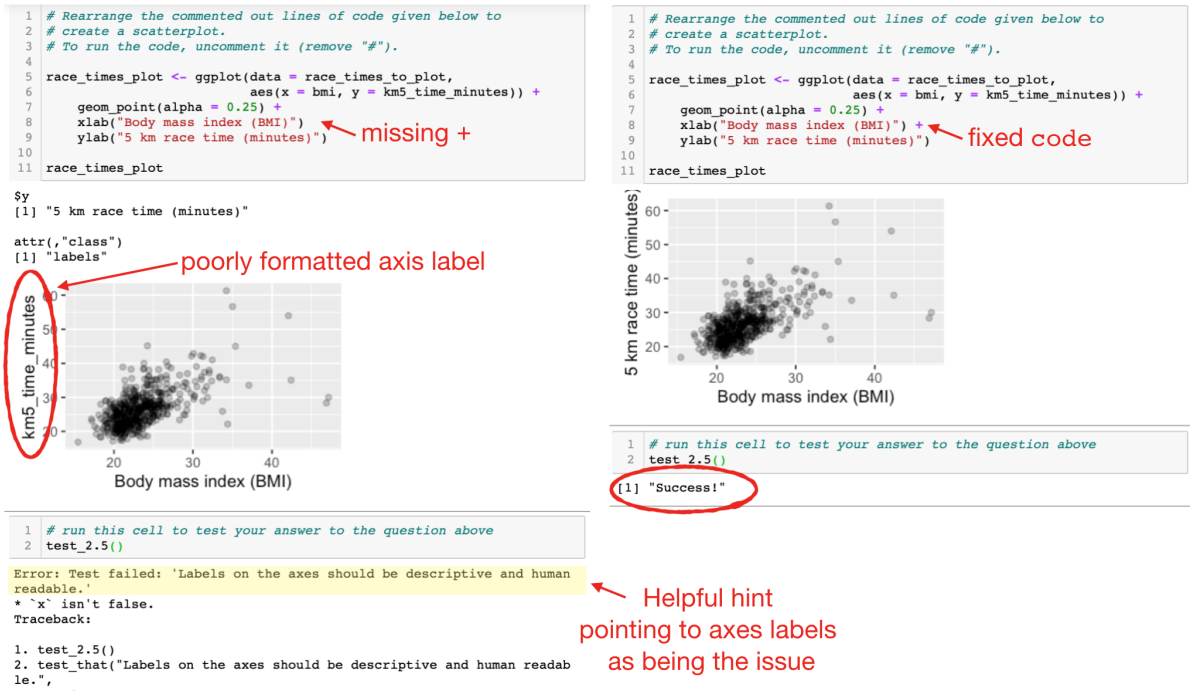

Figure 2: Example of automated software test feedback to students.

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 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 very quickly your learners can 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 intereted 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 knpowledgeable 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. 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 we think it's important to help prevent those frustration or annoyances, because we 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, we 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 our 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 7: Build a safe, inclusive and welcoming community

Number seven is build a safe, inclusive and welcoming community. And we actually think, you know, practicing this talk a couple of times and giving it now, we 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 we 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, we think. So one example of what we, it's not the only thing we do, but one thing that we do in our courses 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, we 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 8: Use checklists to focus and facilitate peer learning

Number eight 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 we should spend our

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 we can say that we personally found reviewing for that far easier than maybe we 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 we think there's real value in these things in the classroom and outside.

Rule 9: Teach students to work collaboratively

Rule 10: Have students do projects

And finally, our last simple rule is to have students do projects. So what we 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 we 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, we 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 we 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

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.

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.

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.

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.

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.

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.

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.