“I’m not sure, but…”: Expert Practices that Enable Effective Code Comprehension in Data Science

Christopher Lum*
lum@ucsd.edu
University of California San Diego
La Jolla, CA, USA

Guoxuan Xu*
g7xu@ucsd.edu
University of California San Diego
La Jolla, CA, USA

Sam Lau
lau@ucsd.edu
University of California San Diego
La Jolla, CA, USA

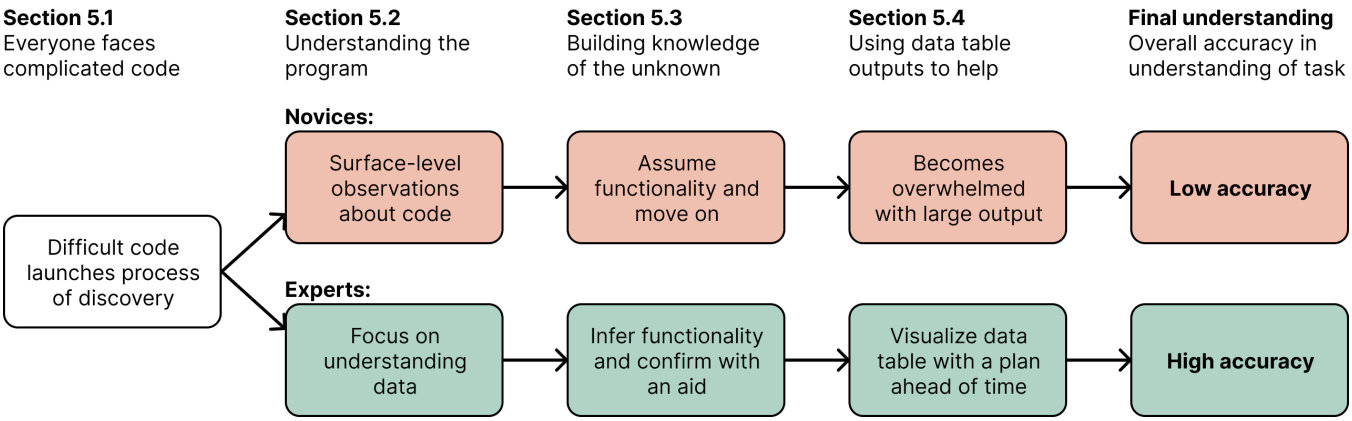

Figure 1: Our paper finds that data science novices (top row) and experts (bottom row) demonstrate different approaches to code comprehension when given data analysis of realistic complexity. These differences lead experts to achieve a better understanding of code compared to novices.

Abstract

Data scientists often need to read and understand messy and undocumented code that relies on large software libraries. What makes data science experts more effective than novices at this task? To understand expert practices, we conducted a think-aloud study where 4 novice and 5 expert data scientists reasoned about an unfamiliar data analysis script with realistic complexity that used the Python pandas library. Surprisingly, familiarity of the pandas package had relatively minor importance for experts. Instead, experts consistently performed three practices that novices did not: experts examined the data in detail rather than fixating on surface-level code features; experts consistently verified their assumptions about how the data was transformed; and experts navigated lengthy program outputs in a goal-directed way. Using these findings, we provide a practical set of guidelines for data science pedagogy and for future tools to support data science learners.

CCS Concepts

• **Human-centered computing → Empirical studies in HCI**.

*Christopher Lum and Guoxuan Xu contributed equally to this work as co-first authors.

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701933

Keywords

data science education, code comprehension, expertise

ACM Reference Format:
Christopher Lum, Guoxuan Xu, and Sam Lau. 2025. “I’m not sure, but…”: Expert Practices that Enable Effective Code Comprehension in Data Science. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701933

1 Introduction

Data scientists – people who manipulate data to gain insights from it [12] – often engage in exploratory programming, where they iteratively write code to try out multiple ideas [7]. For example, people who develop machine learning models frequently experiment with many variations of data manipulations, algorithms, and model parameters. The nature of this work often results in code that is lengthy, messy, and undocumented [28]. Consequently, professional data scientists must develop the skills to understand, debug, and modify such code, whether written by themselves or by their colleagues.

Because of this reality, it is important for data science novices to develop competencies around reading and understanding code that manipulates data tables. This need is heightened by the emergence of large language models (LLMs), which, in the context of data science programming, can often generate code that runs without raising an error but is slightly incorrect and requires careful

debugging [29]. Prior work on program comprehension has focused primarily on introductory computer science courses [18, 24]. In contrast to introductory computer science, introductory data science courses often rely on large software packages to manipulate large datasets. For instance, the paper authors regularly teach data science courses using the `pandas` Python library, which has hundreds of methods, to analyze datasets with millions of rows. Although this suggests that teaching novices code comprehension in data science might be different from teaching it in computer science, little is currently known about what we should teach novices. This raises a critical question: **What are the differences in code comprehension strategies between expert and novice data scientists?**

To address this question, we conducted a think-aloud study with four novice and five expert data scientists as they reasoned about an unfamiliar data analysis in a lab setting. Surprisingly, both novices and experts found the code highly difficult to understand at a first glance, suggesting that familiarity with `pandas` might not have played a dominant role for experts' success. Instead, we observed that experts regularly performed three practices that novices did not: experts examined the data in detail rather than fixating on surface-level code features; experts consistently verified their assumptions about how the code transformed data; and experts examined lengthy program outputs in a highly goal-directed way. These findings are summarized in Figure 1. Overall, these insights suggest metacognitive skills that can be explicitly taught to novices to support their progress towards expertise. This paper makes two main contributions:

- The first empirical study, to our knowledge, that compares expert data scientist behavior to novice data scientist behavior on a realistic data science programming task.
- Pedagogical suggestions to help novices develop metacognitive skills for understanding data analysis code.

2 Related Work

The work in this paper draws from and builds upon prior research in three main areas: data science curriculum development, code comprehension studies, and practices of professional data scientists. The following subsections elaborate.

2.1 Data Science Curricula

Data science is a rapidly growing discipline that integrates methods from both computer science and statistics to make decisions and predictions about the future using large datasets [9]. The demand for individuals with data science skills has motivated many universities around the world to design data science programs at the undergraduate level [25]. In contrast to typical introductory computer science (CS1) courses, which emphasize topics like data types, iteration, control flow, and subprograms [5], introductory data science courses typically emphasize using higher-level software libraries like `pandas` to work with data tables, or dataframes [3, 6] and also strike a balance between computer science and statistics [2, 8, 21]. Although there is general consensus among educators that teaching data science is different from teaching computer science or statistics alone [13], it is still a topic of active investigation to delineate exactly what those differences are. This paper contributes to this ongoing line of work by comparing the skills of novice and expert data scientists, which can guide instructors who are teaching data science to novices.

2.2 Novices Understanding Code

Many prior studies have investigated code comprehension for novices in computer science education. Experts are quicker to comprehend code due to their additional experience and practice [23, 33]. On the other hand, novice programmers tend to have a weak understanding of code in their introductory computer science courses [4, 22]. One way that novices can bolster their understanding of these skills is through code tracing, where novices predict the outcome of code before it is run by hand [11, 14, 15, 22]. In some cases, novices struggle to use tracing due to misunderstandings of the code's applications or simply just the code itself [15]. Another way of remedying this problem-solving issue is by having a better metacognitive understanding of the problem-solving process to know how to tackle problems as they arise [27]. For data science specifically, Singh et al. [31] examined how data science students made errors in writing their own code. However, there is generally a lack of research on how data science students understand code that they didn't write themselves, which we address through the work in this paper.

2.3 Data Science in Practice

The code that data scientists tend to write differs from that of computer scientists. Analysis tasks often requires some form of exploratory programming, where programmers don't necessarily code with an explicit goal in mind when starting [7]. To support this style of programming, computational notebooks like Jupyter enable users to selectively execute code snippets rather than the entire program [19]. However, the nature of exploratory programming can also result in messy, undocumented notebooks [28]. For example, since code snippets in notebooks can be run out-of-order, it is not uncommon for notebooks to contain code that was written to test an idea, then left unused in the final analysis [16]. Thus, data scientists who want to build upon work found online or through colleagues must be able to read and understand messy code found in computational notebooks. This study provides insight into this process by comparing experts and novices as they understand realistic data analyses.

3 Methods

To understand differences between data science novices and experts, we conducted an in-lab, think-aloud user study.

Participants. We recruited novice data scientists by posting email advertisements for undergraduate students in introductory-level data science courses at a large research-focused university in North America. We filtered for students that had taken at least one year of programming experience and had taken at least one course that used the `pandas` library. We recruited expert data scientists through personal outreach to advanced undergraduate students and graduate students in our department. In total, we recruited 4 novice and 5 expert data scientists. Participants received a $20 gift card for participating in the study. Participant demographics are presented in Table 1.

PID	Age Range	Position	Py. Exp. (Yr)	Pandas Exp. (Yr)
N1	18-21	Undergraduate	1	1
N2	18-21	Undergraduate	2	1
N3	18-21	Undergraduate	1	1
N4	18-21	Undergraduate	3	1
E1	22-25	Consultant	5+	4
E2	22-25	Ph.D	3	3
E3	26-29	Ph.D	5	5
E4	18-21	Undergraduate	5+	3
E5	18-21	Undergraduate	3	2

Table 1: Participant demographics. PID = Participant ID; Py. Exp. (Yr) = Years of experience programming in Python; Pandas Exp. (Yr) = Years of experience programming with pandas.

Protocol. Each participant completed a 50-minute Zoom user study with three phases: Training, Task, and Interview. In the Training phase (5 minutes), participants opened a Jupyter notebook with a data cleaning script using the Python pandas package, and the experimenter provided context, goals, and time limits. During the Task phase (30 minutes), participants completed three tasks, each requiring them to read, interpret, and describe code snippets. They could search online, write additional code, and modify the original code, with an unmodified copy available if needed. They had 10 minutes per task and moved to the next if time ran out. After each task, participants rated their confidence and the task difficulty. In the Interview phase (15 minutes), participants reflected on their approach and challenges. Their survey responses, screens, and audio were recorded for analysis.

Tasks. One of the paper authors used their experience in oceanography to create a Jupyter notebook using a real-life dataset from NOAA that recorded ocean current data on a single day in May 2024 [1]. The code that participants were asked to understand was split into tasks which contained between 15-40 lines of pandas code. Each task conducted a common data processing step for this domain – 1) discovering missing values, 2) imputing missing values, and 3) evaluating the effect of imputation. Each code snippet contained multiple pandas function calls for manipulating dataframes. Since the code was adapted from a previous analysis of the data, the code used pandas functions ranging in complexity so that both experts and novices would encounter familiar and unfamiliar functions. The notebook did not contain documentation, but we included a link to the dataset website and used descriptive variable names, which is similar to data science code found in previous work [28]. The complete task code is available on GitHub[1].

4 Quantitative Results

4.1 Assessed Understanding

To measure participant code comprehension, we (the paper authors) created a rubric to score responses from 1 to 5, where 1 indicates a lack of understanding of the program and its goals, and 5 indicates a complete understanding of both. To construct this rubric, we listened to participant responses and scored them as if we were grading responses to an exam question. We then asked two independent evaluators to score each participant response using our rubric. After the initial round of scoring, the evaluators discussed responses where their scores differed until they reached an agreement for all participants.

As a whole, experts performed better than novices in understanding the operations and motivations of the code. The experts had a mean score of 3.00 (σ = 1.13) and the novices had a mean score of 1.75 (σ = 0.75). This difference was found to be statistically significant with a t-test ($t(25) = 3.28, p < 0.005$).

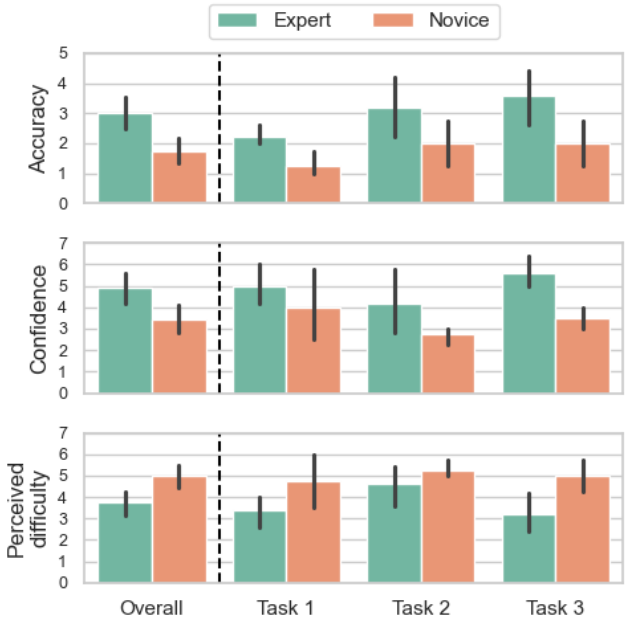

Figure 2: Experts were more accurate, more confident, and found the code easier to understand compared to novices on our code comprehension tasks. Error bars are 95% confidence intervals.

4.2 Self-Reported Understanding

Immediately following each task, the participants were asked to respond to a survey asking about their confidence and perceived difficulty of the task on a scale from 1 to 7 with 1 as the least confident/difficult and 7 being the most. Compared to novices, experts were more confident and found the tasks less difficult. Across all tasks, experts rated their confidence as 4.93 on average (σ = 1.44), and the novices rated their confidence as 3.42 on average (σ = 1.16), which was statistically significant using a t-test ($t(25) = 2.96, p < 0.005$).

As for difficulty, the experts reported a mean of 3.73 (σ = 1.16) for difficulty and novices reported a mean of 5.00 (σ = 0.95). This difference was found to be statistically significant by using a t-test ($t(25) = -3.04, p < 0.005$). Figure 2 illustrates the difference in scored accuracy, confidence, and perceived difficulty split across each of the three tasks and overall.

[1]https://github.com/dstl-lab/Code-Comprehension-User-Study

Theme	Description	Representative Quotes	Participants
Code was challenging for all participants (Section 5.1)	Both experts and novices found `pandas` code difficult to understand.	**Novice**: "There's a lot of random functions I don't think I've learned yet." (N3) **Expert**: "It can get very challenging to keep track of all the different moving parts." (E5)	N1, N2, N3, E1, E5
Experts examine the data rather than surface-level code features (Section 5.2)	Experts begin by examining the data and its schema in detail, while novices tend to use surface features of the code without looking at the data.	**Novice**: "Since the variable is named unmelting, are we melting the ships?" (N2) **Expert**: "I can open up [raw data] files here? Okay. Alright." (E3)	N1, N2, N4, E1, E3, E4
Novices guess; experts guess-and-check (Section 5.3)	Experts consistently check their assumptions about what the code is doing, while novices do not.	**Novice**: "I'm assuming [the variable] unmelted would be the same as melted." (N1) **Expert**: "Even though I think I understand that, I was trying to make that everything's right here." (E3)	N1-4, E1-5
Experts Approach Large Program Outputs With a Goal in Mind (Section 5.4)	Experts hunt for their expected outcomes in program outputs, while novices become overwhelmed by the amount of information presented on-screen	**Novice**: "oh wow, that's a lot of data." (N4) **Expert**: "I'm looking for a word 'TIME' [in the dataframe]." (E2)	N1, N3, N4, E1, E5

Table 2: Summary of quantitative results, organized by themes

5 Qualitative Results

To derive themes from the interviews, the paper authors met regularly to watch participant videos and discuss notes together. Throughout this process, we iteratively came up with a set of themes using an inductive approach [10]. An overview of these themes is provided in Table 2, with detailed descriptions presented in the following subsections.

5.1 Code was challenging for all participants

One characteristic of expertise is the ability to rapidly access past knowledge without conscious effort [30]. In the context of data science programming, this suggests that experts should be able to quickly recognize and accurately use `pandas` functions to understand what code is doing. Surprisingly, both novices and experts considered the code to be highly unfamiliar and difficult to understand. They pointed out that the code contained many function calls that they either hadn't seen before or weren't familiar with. Novices stated that "there's a lot of random functions I don't think I've learned yet" (N3) and "I wasn't really familiar with `melt` and `rename` and `transform`" (N1). Experts also found the code challenging with many unfamiliar functions: "I found it more difficult because I think, well, I didn't know what `melt` did" (E5). Even experts who recognized a function didn't necessarily know how to translate it into usable knowledge: "I know what `pd.melt` does, but I can't really simulate what it does in my head" (E1).

Even familiar function calls were difficult for both experts and novices to reason about. For example, novices stated that "even this is simple code, it's different [from what I learned in class], so I don't think it's what I was thinking" (N2). Even experts found that long sequences of familiar functions were difficult to reason about: "When you have multiple [groupby function calls], and especially with the pivot tables too, it can get very challenging to keep track of all the different moving parts" (E5).

This set of findings suggest that data science code is tricky because even one function call can completely restructure a dataframe (e.g. pivoting a table). Data cleaning scripts often have many function calls in sequence, making the code even more difficult to reason about. In addition, it's not uncommon for datasets to contain thousands of rows or more, making it very difficult to simulate or even keep track of how the code is transforming every single value. Although our experts were able to recognize and trace more function calls than novices, experts still found it initially challenging to reason about multiple function calls in sequence.

5.2 Experts examine the data rather than surface-level code features

All participants attempted to read the code line-by-line and mentally simulate what the code is doing. As mentioned earlier, this was challenging for both experts and novices. When novices couldn't mentally simulate the code, they used surface-level features like method and variable names to try to infer what the code was doing (N1, N2, N4). For example, one novice stated, "Since the variable is named `unmelting`, are we melting the ships now or was there a melting task that I missed?" (N2). Novices also relied on the column names of the dataframes that were displayed in the notebook (N2-N4). However, this strategy wasn't always successful, since variable names and column names themselves could be difficult to understand, as one novice noted, "Maybe the variable name `unmelted`

makes sense to someone who knows what melting is, but I don't know what melting is" (N1). Another novice pointed out that to improve their understanding of the program, "it would really, really help if they have good coding discipline, which is good variable naming" (N4), in spite of the fact that the code was originally written with descriptive variable and function names. In some cases, novices appeared to give up on understanding the code line-by-line and instead tried to used the visualizations generated by the code to guess what the code was doing (N2).

In contrast, experts focused on understanding the data itself rather than the surface-level features of the code and dataframes (E1-E5). In fact, some experts started their process by opening the raw data file as plain text and reading through some of the data values (E3, E4). One justified this by saying, "If I know what the data is, then I have a better sense on how to work with it or how to navigate interpreting it" (E4). Some experts even wanted to open the data file using another program like Excel, mentioning drawbacks of their notebook programming environment:

> "The default Jupyter Notebook output is not that good for looking at the entire data set and just getting an idea of what the columns are, what the types are, whether there's missingness or different types of null values. So just kind of manually inspecting it at first is just a good way to figure out, just get the lay of the land for a dataset you've never seen before." (E1)

Experts would return to the data repeatedly as they traced the code in order to understand how code transformed the dataframes. This behavior stood in stark contrast to novices, none of whom examined the original dataset and instead attempted to draw conclusions about the code immediately.

5.3 Novices guess; experts guess-and-check

Because the think-aloud study asked participants to vocalize their thinking process, we observed that both novices and experts consistently made guesses about the functionality of the code (N1-N4, E1-E5). However, all of our novice participants made guesses without verifying their accuracy. Our tasks were between 10-40 lines of code. One common observation is that novices viewed the code, ran the code to obtain the final output, made a guess about each line of code, and then formed an overall description about the code's purpose without checking whether their guesses about the code were correct. When asked about how they knew what the code was doing, one novice stated, "I read each line, and this is what the code looks like it's doing, so I'm going to go with that" (N3).

In contrast, all of our experts would constantly check their guesses by using print statements to view intermediate variables created in the code snippets. The most common strategy was to print the dataframe before and after a line of code:

> "I can take a snapshot of what the data looks like before an operation and then I could take a look at what it's like after an operation and that was able to confirm my beliefs about what is actually happening to the data. [...] It helps elucidate what's happening programmatically by just being able to go in and double check, 'are all the numbers doing what I want?' or 'is there anything that's off?' " (E5)

Most of the other expert participants echoed this sentiment (E1-E3). Notably, this process was perceived as "very tedious" (E1) and made their notebooks "a mess" (E3), because participants needed to insert new print statements, copy-paste variable names or entire code expressions, and comment out code in order to see the intermediate results of the code. Despite this extra effort, experts felt that this process was necessary to check their assumptions about the code.

5.4 Experts approach large program outputs with a goal in mind

Both novices and experts printed out dataframes to understand how the code worked, using a combination of the default notebook output, the built-in `print()` function in Python, and examining the entire dataset. Because the input data contained many rows, however, novices would often get overwhelmed by the sheer amount of data displayed when printing the dataframe and made many statements like "I'm not sure where to even start" (N1). To deal with this, novices tended to focus on the column names and the first few rows of the data to avoid reading through the entire dataframe (N1, N3, N4).

In contrast, experts had a specific goal in mind when they printed out dataframes. For example, one line of code cleaned out rows that contained the value `'TIME'`, since this value denoted header data rather than actual data values. When experts wanted to see how this code worked, they printed out the entire original dataframe and then scrolled through the dataframe to specifically look for rows with the `'TIME'` value (E1, E5) because these values were nestled in the middle of the dataframe between actual data values. This process generated insight about the data:

> I'm looking for a word `'TIME'` here, and it seems like, whoa, it seems like what this text file is all about is they've stitched multiple tables together vertically. (E5)

This goal-directed behavior seemed to help experts navigate through larger table outputs to confirm or reject their guesses about code, rather than fixating on the surface-level details of every single data point.

6 Discussion

This section describes the limitations of our study, reflects on the differences between CS1 and introductory data science, and provides recommendations for instructors who teach introductory data science.

6.1 Study limitations

Our study findings are limited in the following ways. Our sample was limited to a small segment of the broader data science novice and expert population. The novice participants were undergraduate students, but there are many other groups of novice data science learners, such as professional software engineers who want to pivot to machine learning. In addition, the expert participants were advanced undergraduates, graduate students, or individuals who recently earned a graduate degree. It's possible that data scientists with more years of experience would have greater knowledge of

pandas and thus use different strategies to understand code. Because of these limitations, we use our study observations to make recommendations for what instructors *can* do to support novices, rather than what instructors *should* do.

6.2 Expert metacognitive skills in data science

Before we began this study, we expected that the differences between expert and novice data scientists would be dominated by procedural knowledge. Since experts have had more experience programming and working with pandas, we thought that experts would be able to trace code using their knowledge of familiar function calls. Instead, we found that experts were initially just as confused as novices.

Our study revealed that the key advantage experts had over novices was their highly developed metacognitive skills. These skills enabled experts to probe code outputs and maintain a sense of skepticism about the data. However, there is currently a lack of knowledge about what these metacognitive skills entail in the context of data science. Identifying desired metacognitive skills, and explicitly teaching them is an important part of pedagogy and can particularly benefit students from underrepresented backgrounds [17, 32]. As such, we make the following pedagogical recommendations for instructors teaching data science novices:

(1) *Embrace feeling uncertain about code.* In our study, even experts couldn't remember what pandas methods did and felt confused when multiple pandas methods were called in sequence. Instructors can help novices understand that this feeling is to be expected when working with complicated data science code and is not necessarily an indication that they lack sufficient knowledge about data science to proceed.
(2) *Go back to the data.* Experts in our study wanted to examine the original data in detail and develop as much contextual information as possible before starting to understand the code. Instructors can encourage novices to open the dataset and understand it as deeply as possible, which can help novices make sense of code.
(3) *Externalize questions about the data.* Novices frequently lost track of their assumptions because they became overwhelmed by large dataframe displays. To help address this, instructors can teach novices to explicitly externalize their guesses (e.g. by writing them down) before verifying them in the data.

To our knowledge, this study is the first to provide empirical evidence for metacognitive skills that data science instructors should teach their students. Future studies should seek to discover a more comprehensive set of metacognitive skills and design pedagogical techniques to help students develop metacognitive skills in data science.

6.3 Beyond code comprehension in introductory data science

With the emergence of LLM tools that can quickly generate code, code comprehension has become an important competency in addition to writing code from scratch [20, 26]. In CS1, there is a relatively limited subset of primitive language features that students need to learn, and most operations transform single data values. Thus, students who have achieved mastery in CS1 can be expected to precisely trace a program's execution. However, in data science this is less feasible – even a single pandas function can have over ten optional arguments that change the function's behavior in subtle but important ways. Additionally, it is very difficult to create a precise mental simulation for code that repeatedly transforms thousands of rows of data. In this light, what do code comprehension skills in data science look like? We make initial suggestions based on our comparisons of novices and experts in this study.

First, code and data comprehension in data science appear to be inextricably linked. Experts were able to understand the code with greater accuracy in part because they carefully examined the raw data and its schema. Since many attributes of the dataset appear in the code (e.g. hard-coding column names in function arguments), assessing code comprehension in data science might begin by assessing how well learners understand the characteristics of their data.

Second, code comprehension in data science seems to involve developing an understanding of *why* a transformation was done in addition to *what* was done. Code that runs without raising a Python error can still be erroneous for data analysis – for example, pandas will not raise an error if you group a dataframe using a column of continuous data like temperature, even though this operation would seldom have semantic meaning. A row in a dataframe is a representation of a real-life measurement, and experts in our study sought to understand how dataframe transformations would generate conclusions about the data in context. In summary, perhaps the term *code comprehension* is too narrow for data science, since code in data science is almost always interpreted in context.

7 Conclusion

In this paper, we examine differences between novice and expert data scientists in a realistic data analysis scenario. A think-aloud study with 4 novice and 5 expert data scientists found that procedural knowledge about pandas function calls seemed to play a relatively minor role for experts, who also found the data analysis code initially overwhelming. Instead, our study revealed metacognitive practices that experts consistently performed more often than novices to develop more accurate understanding of data analysis code. Altogether, this work provides evidence that code comprehension skills in data science differ from those in computer science, and provides pedagogical recommendations for data science instructors who seek to teach both procedural and metacognitive skills to learners.

References

[1] [n. d.]. NDBC - Observations - Radial Search — ndbc.noaa.gov. https://www.ndbc.noaa.gov/radial_search.php?lat1=32.868N&lon1=117.267W&uom=E&dist=250. [Accessed 20-05-2024].
[2] Joel C. Adams. 2020. Creating a Balanced Data Science Program. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. ACM. https://doi.org/10.1145/3328778.3366800
[3] Ani Adhikari, John DeNero, and Michael I Jordan. 2021. Interleaving computational and inferential thinking: Data science for undergraduates at. *Harvard Data Science Review* 3, 2 (2021).
[4] Ella Albrecht and Jens Grabowski. 2020. Sometimes It's Just Sloppiness - Studying Students' Programming Errors and Misconceptions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. ACM. https://doi.org/10.1145/3328778.3366862
[5] Richard H Austing, Bruce H Barnes, Della T Bonnette, Gerald L Engel, and Gordon Stokes. 1979. Curriculum'78: recommendations for the undergraduate program

in computer science—a report of the ACM curriculum committee on computer science. *Commun. ACM* 22, 3 (1979), 147–166.

[6] Ben Baumer. 2015. A data science course for undergraduates: Thinking with data. *The American Statistician* 69, 4 (2015), 334–342.

[7] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. https://doi.org/10.1109/vlhcc.2017.8103446

[8] Joshua Burridge and Alan Fekete. 2022. Teaching Programming for First-Year Data Science. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE 2022)*. ACM. https://doi.org/10.1145/3502718.3524740

[9] Longbing Cao. 2017. Data science: a comprehensive overview. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 1–42.

[10] Juliet Corbin and Anselm Strauss. 2015. *Basics of qualitative research*. Vol. 14. sage.

[11] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM. https://doi.org/10.1145/3105726.3106190

[12] Thomas H Davenport and DJ Patil. 2012. Data scientist. *Harvard business review* 90, 5 (2012), 70–76.

[13] Richard D De Veaux, Mahesh Agarwal, Maia Averett, Benjamin S Baumer, Andrew Bray, Thomas C Bressoud, Lance Bryant, Lei Z Cheng, Amanda Francis, Robert Gould, et al. 2017. Curriculum guidelines for undergraduate programs in data science. *Annual Review of Statistics and Its Application* 4, 1 (2017), 15–30.

[14] Ankur Gupta and Ryan Rybarczyk. 2023. Improving Long Term Performance Using Visualized Scope Tracing: A 10-Year Study. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. ACM. https://doi.org/10.1145/3545945.3569748

[15] Mohammed Hassan and Craig Zilles. 2023. On Students' Usage of Tracing for Understanding Code. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. ACM. https://doi.org/10.1145/3545945.3569741

[16] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM. https://doi.org/10.1145/3290605.3300500

[17] Scott Horrell, Jana Marcette, and Sudarsan Kant. 2019. Metacognition: A Tool for Overcoming Discrimination. *Peer Review* 21 (2019).

[18] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, et al. 2019. Fostering program comprehension in novice programmers-learning activities and learning trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. 27–52.

[19] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks–a publishing format for reproducible computational workflows. In *Positioning and power in academic publishing: Players, agents and agendas*. IOS press, 87–90.

[20] Sam Lau and Philip Guo. 2023. From "Ban it till we understand it" to" Resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*. 106–121.

[21] Sam Lau, Deborah Nolan, Joseph Gonzalez, and Philip J Guo. 2022. How Computer Science and Statistics Instructors Approach Data Science Pedagogy Differently: Three Case Studies. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 29–35.

[22] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin* 36, 4 (June 2004), 119–150. https://doi.org/10.1145/1041624.1041673

[23] Tony Lowe. 2019. Explaining Novice Programmer's Struggles, in Two Parts: Revisiting the ITiCSE 2004 working group's study using dual process theory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. ACM. https://doi.org/10.1145/3304221.3319775

[24] Greg L Nelson, Benjamin Xie, and Amy J Ko. 2017. Comprehension first: evaluating a novel pedagogy and tutoring system for program tracing in CS1. In *Proceedings of the 2017 ACM conference on international computing education research*. 2–11.

[25] National Academies of Sciences, Division of Behavioral, Social Sciences, Board on Science Education, Division on Engineering, Physical Sciences, Committee on Applied, Theoretical Statistics, Board on Mathematical Sciences, Analytics, et al. 2018. *Data science for undergraduates: Opportunities and options*. National Academies Press.

[26] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Virginia Pettit, Leo Porter, et al. 2024. How Instructors Incorporate Generative AI into Teaching Computing. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2*. 771–772.

[27] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM. https://doi.org/10.1145/3287324.3287374

[28] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM. https://doi.org/10.1145/3173574.3173606

[29] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by your progress! large language models (gpt-4) no longer struggle to pass assessments in higher education programming courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*. 78–92.

[30] Herbert Simon and William Chase. 1988. *Skill in Chess*. Springer New York, 175–188. https://doi.org/10.1007/978-1-4757-1968-0_18

[31] Anjali Singh, Anna Fariha, Christopher Brooks, Gustavo Soares, Austin Z. Henley, Ashish Tiwari, Chethan M, Heeryung Choi, and Sumit Gulwani. 2024. Investigating Student Mistakes in Introductory Data Science Programming. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM. https://doi.org/10.1145/3626252.3630884

[32] Khanh Lam LH Tao. 2021. *Metacognition and First-Year College Success: Understanding the Experiences of the Underrepresented in STEM*. Ph. D. Dissertation. Northeastern University.

[33] Susan Wiedenbeck. 1985. Novice/expert differences in programming skills. *International Journal of Man-Machine Studies* 23, 4 (Oct. 1985), 383–390. https://doi.org/10.1016/s0020-7373(85)80041-9