Investigating Student Mistakes in Introductory Data Science Programming

Anjali Singh
University of Michigan
singhanj@umich.edu

Anna Fariha
University of Utah
afariha@cs.utah.edu

Christopher Brooks
University of Michigan
brooksch@umich.edu

Gustavo Soares
Microsoft, USA
Gustavo.Soares@microsoft.com

Austin Z. Henley
Microsoft, USA
austinhenley@microsoft.com

Ashish Tiwari
Microsoft, USA
Ashish.Tiwari@microsoft.com

Chethan M
Microsoft, India
chethanm@microsoft.com

Heeryung Choi
Massachusetts Institute of Technology
heeryung@mit.edu

Sumit Gulwani
Microsoft, USA
sumitg@microsoft.com

ABSTRACT

Data Science (DS) has emerged as a new academic discipline where students are introduced to data-centric thinking and generating data-driven insights through programming. Unlike traditional introductory Computer Science (CS) education, which focuses on program syntax and core CS topics (e.g., algorithms and data structures), introductory DS education emphasizes skills such as analyzing data to gain insights by making effective use of programming libraries (e.g., `re`, `NumPy`, `pandas`, `scikit-learn`). To better understand learners' needs and pain points when they are introduced to DS programming, we investigated a large online course on data manipulation designed for graduate students who do not have a CS or Statistics undergraduate degree. We qualitatively analyzed students' incorrect code submissions for computational notebook-based assignments in Python. We identified common mistakes and grouped them into the following themes: (1) programming language and environment misconceptions, (2) logical mistakes due to data or problem-statement misunderstanding or incorrectly dealing with missing values, (3) semantic mistakes due to incorrect use of DS libraries, and (4) suboptimal coding. Our work provides instructors insights to understand student needs in introductory DS courses and improve course pedagogy, and recommendations for developing assessment and feedback tools to support students in large courses.

CCS CONCEPTS

• **Applied computing → E-learning**.

KEYWORDS

Introductory Data Science Programming, Types of Mistakes, Qualitative Analysis, Data Manipulation in Python

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03…$15.00
https://doi.org/10.1145/3626252.3630884

ACM Reference Format:
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), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630884

1 INTRODUCTION

The growing importance of Data Science (DS) education demands more research on the curriculum and assessment practices in DS courses [29]. The rapid growth of enrollment in DS courses requires pedagogy and assessment approaches that work at scale, which requires a deeper understanding of students' difficulties in these new courses. This is particularly important for introductory DS (hereafter referred to as DS1[1]) courses, such as data manipulation, which form the foundation of students' future DS undertakings.

Data manipulation courses typically cover topics such as data cleaning, preparation, and exploration [9, 12], e.g., selecting, filtering, aggregating, and transforming values in a DataFrame (the core DS1 data structure). In a DataFrame each row represents a single observation, and each column represents an attribute (or variable in the statistics sense) of the observations. Practitioners spend significant time on data manipulation activities, yet there have been few efforts to formalize data exploration and cleaning. Such topics are relatively neglected in teaching, especially when compared to data modeling topics—covered in Statistics and Machine Learning (ML) courses—using datasets that are already cleaned and ready for ingestion [17, 47]. While significant efforts have been employed to understand CS education, including the formation of the ACM SIGCSE conference, CS1 pedagogy research may not generalize to DS1 [36]. Beyond learning how to solve DS problems computationally, two other important aspects of DS education are precise knowledge of the problem domain and data literacy [17] (a term used to broadly describe the set of abilities around the use of data for solving real-world problems [51]). While both CS1 and DS1 require the students to learn the fundamentals of a programming language, the DS1 approach tends to embed this in data manipulation activities that vary with the data being analyzed in contrast with the

[1] We refer to introductory DS as *DS1*, noting that currently this is not a standard term.

CS1 approach of solving general-purpose computing tasks [19]. This can cause fundamental differences in how a student considers computation: for instance, many DS libraries contain *vectorized* functions, promoting their use to effectively manipulate data, as opposed to implementing algorithms and data structures through student-defined functions or classes [29, 36].

Our work advances knowledge of the nascent field of DS Education through an evidence-based approach. We focus on DS1 student programming errors and misconceptions by investigating a large online data manipulation course. Described in more detail in Section 3, this Masters' degree course introduces data cleaning and analysis techniques using the popular `pandas` library in Python. By understanding some of the mistakes these novice DS learners make, we aim to understand the skills and competencies on which DS1 learners require additional guidance and feedback. Therefore, we explore the following research question:

RQ: *What are novice learners' mistakes and inappropriate strategies when they are introduced to programming in a graduate level data manipulation course?*

We explore this question by qualitatively analyzing 47 students' 136 incorrect code submissions. Our key contribution is a categorization of DS1 mistakes that instructors can use to understand student needs and pain points, revise curricula, and provide directed feedback. To understand the competencies on which to support learners, we consider both students' mistakes and inappropriate programming strategies given the course's learning objectives. We show that students' code needs to be evaluated along five axes that should accordingly inform the nature of instructional support they receive: (1) programming in computational notebooks, (2) data literacy, (3) programming with DS libraries, (4) programming strategies, and (5) writing optimal code using vectorized operations and functions. However, current assessment tools [6, 23, 33, 37, 52] focus primarily on the correctness of the code output and do not support students' skills acquisition along all of these competencies. Based on the identified mistake types, we provide pedagogical recommendations for supporting learners in data manipulation courses and insights for developing assessment and feedback tools to help students acquire skills even in large courses. We also contribute the dataset of 136 incorrect student code submissions labeled with mistake categories[2].

2 RELATED WORK

Data Science Education. There are several publications on course design and experience reports of conceptualizing and teaching specific DS courses [3, 5, 9–12, 15, 19, 42, 43, 50]. The 2019 report by the ACM Task Force on DS Education [16] provides suggestions for core competencies a graduating student should leave with and suggests topics that a full DS curriculum should integrate. Recently, Lau et al. [30] reported their experience of balancing the Statistics and CS concepts while launching a DS1 course. They demonstrated how CS and Statistics instructors typically approach DS differently. Kross and Guo shared their findings from interviewing DS practitioners who taught novices in both industry and academic settings [29]. They found that practitioners often lack formal pedagogical training, raising the importance of conducting more research on DS pedagogy for novices. This body of work reveals that despite the growing importance of DS, due to its recent formalization as an academic discipline, more consensus is needed on DS1 teaching practices. Our work addresses this research gap by identifying student mistakes in DS1 courses to improve DS1 pedagogy.

Diagnosis of Programming Difficulties. Capturing and understanding students' common errors and misconceptions are important to enhance instructors' pedagogical content knowledge [41]. Several computing-education researchers have analyzed students' difficulties in CS1, using a diverse set of definitions and approaches [1, 2, 8, 26, 28]. Luca et al. [13] developed a curated inventory of programming language misconceptions, focusing primarily on syntax and semantic errors. They described a way to organize a collection of such misconceptions to present a synthesis of past research to educators in an accessible form, thus, bridging the gap between research and educational practice. Similarly, we provide actionable insights to improve DS1 curriculum and assessment practices.

Limited research has explored DS student challenges. Nguyen et al. [36] took a first step towards analyzing DS students' code at scale by leveraging metrics from traditional software engineering (e.g., Halstead Volume [24] and Cyclomatic Complexity [34]) in combination with DS-specific metrics such as number of library calls. They identified metrics indicative of task complexity, submission runtime, and submission score. However, these metrics operate over the entire code as a single entity and, thus, cannot be used to isolate incorrect parts of the code. Skripchuk et al. [48] qualitatively analyzed incorrect student code to identify common errors and misconceptions in open-ended projects from an upper-division ML course and provided suggestions on how instructors can mitigate these errors. We use a similar approach for a DS1 course and find some common mistake categories, e.g., both works identified students who did not leverage DataFrame operations for writing optimal solutions and instead used for loops to iterate over DataFrame rows.

While scalable code analysis methods for DS1 are limited, there is significant research on identifying students' mistakes in CS1 courses using unsupervised techniques, such as code clustering [22, 25, 45], which typically rely on static code analysis by representing code as Abstract Sytnax Trees (ASTs) or neural embeddings [38]. While these approaches work well in CS1, a challenge in DS1 is that two qualitatively different solutions to a problem may have similar ASTs, which is not commonly observed for CS1 problems. This is due to the nature of DS1 programming, which relies extensively on using existing API and library functions, where changing the function parameter values does not lead to a change in the AST. Since CS1 code clustering techniques do not generalize to DS1, we qualitatively analyzed students' incorrect code submissions to reveal student mistakes. To instigate future research in this area, we identified features of DS1 code that can be useful for automatically detecting mistakes using semi-supervised approaches [18].

3 METHOD

Course Context and Study Population. The DS1 course we analyzed is the first technical course offered as part of a one-year online graduate program in applied DS offered by the University of Michigan. This program attracts students from diverse backgrounds, many of whom are mid-career professionals. This program is designed for students who have only introductory programming and statistics

[2]Dataset is available here

knowledge. All the courses in the program are fast-paced 4-week courses. The technical instruction is provided in the JupyterLab computational notebook environment [27] using Python, and the environment is set up with all the necessary libraries to complete the course. Graduates of this program are expected to have practical skills in a breadth of DS topics (e.g., applied ML, experiment design, information visualization, etc.) using Python-based toolkits.

The DS1 course's learning objectives include learning basic data manipulation and cleaning techniques by effectively using the `re` [44] and `pandas` [39] libraries. This includes ingesting, cleaning, manipulating, and transforming data and performing basic inferential statistical analyses. Weekly programming assignments (4 in total) are automatically graded using nbgrader [35], a tool that facilitates creating and grading Jupyter-based assignments. Each assignment consists of 3–4 individually graded questions. A series of unit tests are used to provide feedback to learners regarding the correctness of their solutions. Each question is graded as pass or fail, and the overall assignment grade is equally weighted among all questions. Importantly, students are allowed unlimited submissions for each assignment until the deadline (one week after the assignment is released), resulting in most students eventually achieving full points for the assignment.

Data. All the assignments were based on the `pandas` library, except the first one, which was the easiest and based on the `re` library. Given the popularity of `pandas` and since we were interested in understanding learners' mistakes when they are *introduced* to DS1 programming, we chose the second assignment for analysis of student errors and misconceptions. We sampled from a collection of 542 notebook submissions for this assignment by 151 students enrolled in the Fall'21 offering of the course. The assignment had 3 questions on selecting parts of the provided dataset, performing simple manipulations, aggregating dataset values, dealing with missing values, and performing a basic statistical correlation test. All questions are based on the 2017 data on immunizations provided by the U.S. Center for Disease Control (CDC) and its accompanying 252-page data guide. The assignment was designed as an authentic DS inquiry with the goal of emulating typical *data cleaning* and *wrangling* tasks performed by data scientists as the first step in a DS workflow following data acquisition [17]. Figure 1 shows the instructions for the second question in this assignment.

Sampling Procedure. We sampled notebooks for one assignment question at a time. First, all notebook submissions were autograded using nbgrader. Then, from the subset of students who had at least one incorrect submission for a given question, we iteratively picked a student at random and qualitatively analyzed a sequence of their incorrect submissions until a correct submission was found. Therefore, the sample consisted of multiple notebooks per student including both intermediate and final submissions. The sampling procedure was repeated until we achieved data saturation [20] for each question (i.e., new mistakes were no longer discovered in subsequent submissions that were analyzed). In this way, we sampled 136 notebooks submitted by 47 students in total. Table 1 shows aggregates of the analyzed data.

Qualitative Analysis. We qualitatively analyzed the sampled submissions to understand learners' mistakes and inappropriate strategies,

Statistic	Sampled Data	Question 1	Question 2	Question 3
# students	47	23	24	10
# analyzed submissions	136	45	49	42
Max # submissions analyzed for a student	15	7	7	15
Mean (± std err) # submissions per student	2.89 (± 0.48)	1.96 (± 0.34)	2.04 (± 0.32)	4.2 (± 1.48)

Table 1: Summary of the data sample used in our analysis. The second column presents statistics for the full sample, and the other columns represent each question in the data sample.

one question at a time. The unit of analysis was a student's code, which they were asked to write in a single notebook cell. After reading the code line-by-line, we referred to the other notebook cells if we were unable to understand the student's approach. For instance, some students defined variables outside the cell in which they coded. In our analysis, we also looked at: (i) the code output and compared it with the correct output and (ii) Python interpreter errors (if any). While analyzing a sequence of incorrect submissions by a student, we noted all the mistakes in each submission.

Using thematic analysis [7] to explore the emergent themes, the first author, who had previously been a teaching assistant for another data manipulation course, consolidated a coding scheme for mistake categories. This categorization was based on the misconceptions that were likely to be causing the student mistakes. The mistake categories were developed and refined after repeated readings of the data. Following this, the first author and another author (who had taught the DS1 course previously) independently coded a random sample of 30 (20%) submissions using the coding scheme, and labeled each submission with one or more mistake categories. An 82.76% agreement was achieved by averaging the Jaccard similarity (the intersection of all applied codes over the union of all applied codes) for all 30 pairs of coded submissions. All discrepancies were discussed and resolved to finalize the mistake categories in this round of coding. Then the first author updated the coding scheme and coded all the data.

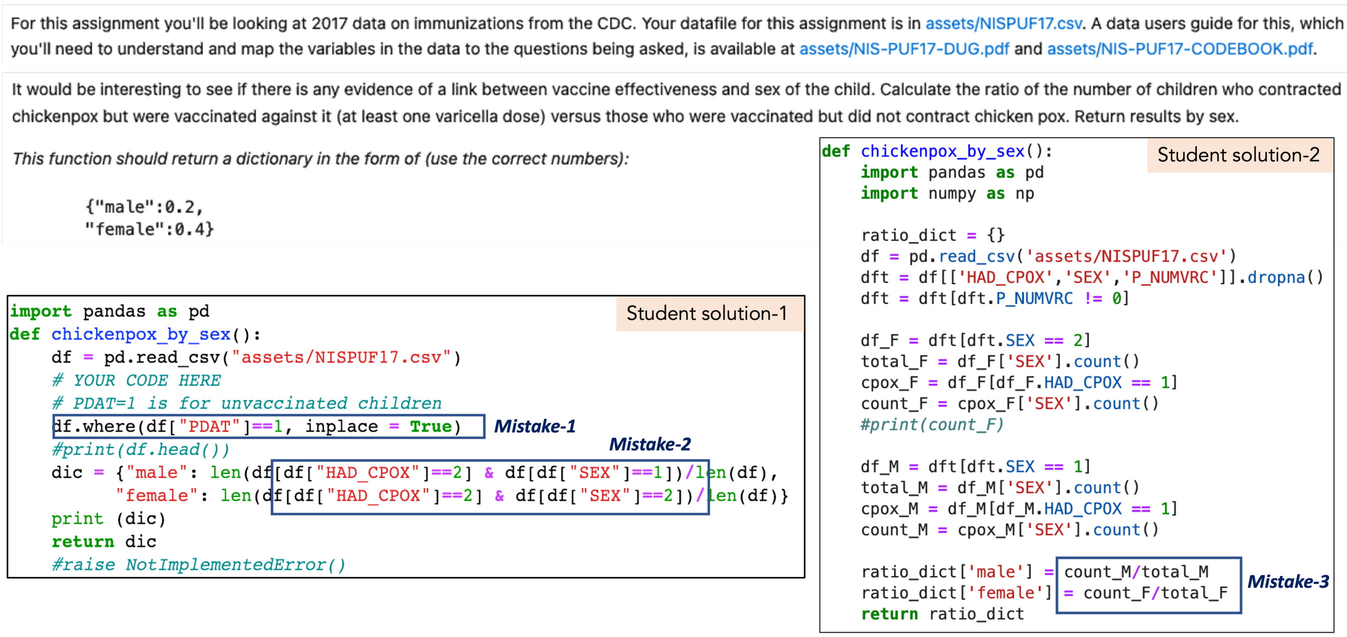
4 RESULTS: TYPES OF DS1 MISTAKES

We now describe the types of DS1 mistakes we identified following the method described in Section 3. To demonstrate how mistakes in each category are manifested in students' code, we primarily use examples of incorrect code submissions for the second question of the assignment. In this question, students are asked to calculate, for the genders male and female, the ratio of the number of children who contracted chicken pox but were vaccinated against it (i.e., received at least one varicella dose) versus the number of children who were vaccinated but did not contract chicken pox. Students were instructed to return the output as a dictionary with the keys "male" and "female". In the dataset, the column `'P_NUMVRC'` indicates the number of varicella doses received by a child, column `'HAD_CPOX'` indicates whether a child contracted chicken pox (`'HAD_CPOX'`=1) or not (`'HAD_CPOX'`=2), and column `'SEX'` indicates gender (1 for male and 2 for female).

A common categorization for programming errors is to divide them into syntactic, semantic, and logical errors [26]. We extend this categorization to DS1 programming as follows:

4.1 Logical Mistakes

A program with logical mistakes does not behave as expected (in our case, does not produce the expected output) according to the



For this assignment you'll be looking at 2017 data on immunizations from the CDC. Your datafile for this assignment is in assets/NISPUF17.csv. A data users guide for this, which you'll need to understand and map the variables in the data to the questions being asked, is available at assets/NIS-PUF17-DUG.pdf and assets/NIS-PUF17-CODEBOOK.pdf.

It would be interesting to see if there is any evidence of a link between vaccine effectiveness and sex of the child. Calculate the ratio of the number of children who contracted chickenpox but were vaccinated against it (at least one varicella dose) versus those who were vaccinated but did not contract chicken pox. Return results by sex.

This function should return a dictionary in the form of (use the correct numbers):

```
{"male":0.2,
"female":0.4}
```

Student solution-1

```python
import pandas as pd
def chickenpox_by_sex():
    df = pd.read_csv("assets/NISPUF17.csv")
    # YOUR CODE HERE
    # PDAT=1 is for unvaccinated children
    df.where(df["PDAT"]==1, inplace = True)
    #print(df.head())
    dic = {"male": len(df[df["HAD_CPOX"]==2] & df[df["SEX"]==1])/len(df),
           "female": len(df[df["HAD_CPOX"]==2] & df[df["SEX"]==2])/len(df)}
    print (dic)
    return dic
    #raise NotImplementedError()
```

Mistake-1

Mistake-2

Student solution-2

```python
def chickenpox_by_sex():
    import pandas as pd
    import numpy as np

    ratio_dict = {}
    df = pd.read_csv('assets/NISPUF17.csv')
    dft = df[['HAD_CPOX','SEX','P_NUMVRC']].dropna()
    dft = dft[dft.P_NUMVRC != 0]

    df_F = dft[dft.SEX == 2]
    total_F = df_F['SEX'].count()
    cpox_F = df_F[df_F.HAD_CPOX == 1]
    count_F = cpox_F['SEX'].count()
    #print(count_F)

    df_M = dft[dft.SEX == 1]
    total_M = df_M['SEX'].count()
    cpox_M = df_M[df_M.HAD_CPOX == 1]
    count_M = cpox_M['SEX'].count()

    ratio_dict['male'] = count_M/total_M
    ratio_dict['female'] = count_F/total_F
    return ratio_dict
```

Mistake-3

Figure 1: A snapshot of the analyzed assignment and two student submissions. Mistake-1 is a semantic mistake (function `where()` used incorrectly), and a logical mistake due to dataset misunderstanding (attribute '`PDAT`' incorrectly used instead of '`P_NUMVRC`'). Mistake-2 is another semantic mistake due to incorrect use of the bitwise AND (&) operator on two DataFrame objects, which is an invalid operation. Mistake-3 is a logical mistake due to problem-statement misunderstanding (ratio denominators incorrectly computed as the number of *all* vaccinated children instead of vaccinated children who did not contract chicken pox).

problem statement. This typically does not throw a runtime error. We further divide logical mistakes into the following subcategories (some mistake types may belong to multiple subcategories):

4.1.1 Dataset misunderstanding. This category includes mistakes from misunderstanding the dataset, its schema, or associated data guide. Such mistakes can be caused due to the following reasons:

- *Selecting incorrect dataset attributes (columns in a DataFrame).* Mistake-1 in Figure 1 is an example of this category. Instead of using the column '`P_NUMVRC`' from the DataFrame, the student has used the '`PDAT`' column which denotes whether a child has adequate provider data. Based on the comment written by the student before the line of code with Mistake-1, they misinterpreted the meaning of the '`PDAT`' column.
- *Using incorrect dataset values or selecting the wrong rows from a DataFrame.* For instance, a student incorrectly used the values `'male'` and `'female'` instead of 1 and 2 when filtering rows using the `'SEX'` column.

4.1.2 Problem-statement misunderstanding. This category includes mistakes due to incorrectly translating the problem-statement instructions into code due to misinterpretation. Sometimes it may be difficult to conclude whether a student made a mistake because of misunderstanding the problem statement or the dataset. For this reason, we only categorize mistakes as belonging to the 'problem-statement misunderstanding' subcategory if they do not involve any misinterpretation related to the dataset.

Mistake-3 in Figure 1 is a mistake of this type as instead of returning the ratio of vaccinated children who contracted chicken pox versus vaccinated children who did not, the ratio of vaccinated children who contracted chicken pox versus *all* vaccinated children was returned. Other examples include rounding up ratios instead of returning the actual ratios as instructed, incorrectly ordering dictionary keys, and returning percentages instead of ratios.

4.1.3 Incorrectly dealing with missing values. Mistakes where students incorrectly deal with a DataFrame's missing values typically belong to one of the two aforementioned sub-categories in addition to this sub-category. For instance, instead of using the condition `df['P_NUMVRC']>0`, a student used the condition `df['P_NUMVRC'] !=0` to select rows indicating children who received at least one varicella dose. This is incorrect as the column `'P_NUMVRC'` consists of missing values, which are counted in addition to the number of rows where '`P_NUMVRC`' is greater than 0. Another example is from a solution to the third question, which required computing the correlation between the `'P_NUMVRC'` and `'HAD_CPOX'` columns. A student replaced all missing values in both columns with `0`, leading to incorrect computation of the correlation as missing values from the `'P_NUMVRC'` column are incorrectly counted towards children receiving zero doses.

4.2 Semantic Mistakes

Semantic mistakes arise from incorrectly selecting or using a DS library, function, or operator, which may or may not throw a runtime error. In addition to a logical mistake (mentioned in Section 4.1.1) Mistake-1 in Figure 1 contains a semantic mistake as the `where()` function from `pandas` was incorrectly used to select rows that satisfy the given condition. However, this does not filter out the rows where the condition is not satisfied, as `where()` is used for replacing values where the condition is `False` to some specific value (specified in the call through a parameter named `other`). Consequently, in the subsequent lines of code, `len(df)` returns the length of the full DataFrame rather than the number of rows where the condition is satisfied. Another example of a semantic mistake is Mistake-2, where the student attempted to use the bitwise AND operator directly on two DataFrame objects, which is not a valid operation. The correct way of computing the numerator for the key "`male`" is: `df[(df["HAD_CPOX"]==2)&(df["SEX"]==1)])`, i.e., combining two boolean masks[3] using a bitwise operator, with parentheses around each mask to override the default operator precedence rules. Some other examples of semantic mistakes that we observed were: (i) dividing a DataFrame slice[4] by another slice rather than dividing their lengths to compute a given ratio, and (ii) using the `groupby()` function incorrectly, where the student misunderstood the output format of the object returned by `groupby` and performed incorrect operations in subsequent lines of code.

4.3 Suboptimal Coding

A data-processing code is suboptimal if it lacks proper optimizations like vectorized operations, and, therefore, fails to scale to large datasets. Suboptimal code may or may not throw an error. In our data sample, we found instances of for-loops being used to iterate over DataFrame rows such as to count the number of rows that satisfy a given condition, instead of the more optimal method of applying a Boolean mask to do the same. While it is not necessarily incorrect to do so, this is an inappropriate DS programming strategy as it can slow down the performance of the code, especially when working with large datasets. Iterating through a DataFrame row by row is a relatively slow operation in Python compared to using vectorized operations (e.g., `map()`, `groupby()`, `join()`, `merge()`, and filtering using Boolean masks), which have the ability to run an operation across a whole DataFrame at once.

4.4 Language & Environment Misconceptions

This category consists of mistakes because of misunderstanding the syntax or the programming environment (in our case, JupyterLab), leading to Python interpreter errors, e.g., (i) incorrectly specifying the path to the dataset file, (ii) using incorrect python syntax, (iii) not defining a variable or failing to import a library, and (iv) defining a variable or importing a library in a cell different from the one in which the solution is being written, such that it is out of scope. For instance, some students defined variables used within a function in another function used for answering a different question, thus making those variables out of scope. Another mistake in this category was not defining `df` (a common variable name used for a DataFrame) or `pd` (a commonly used alias for `pandas`) or defining them in a cell below the one where they were referred.

5 DISCUSSION AND IMPLICATIONS

We now discuss the results by structuring our discussion around the DS1 competencies that emerged from our analysis.

[3]A `pandas` feature that enables filtering a DataFrame based on a set of conditions.
[4]A subset of a DataFrame that includes selected rows and/or columns.

Data Literacy and Programming Strategies. Beyond strategic knowledge, problem-solving in DS requires accurate domain knowledge. The mistakes related to misunderstanding the dataset and problem statement or incorrectly dealing with missing values highlight students' struggle with understanding the *dataset domain*. The skills needed include going over the dataset and its data guide (which can be very long and tedious to read) and finding associations between the ask in the problem statement and the information gathered from the data. This is even more challenging for novices who lack prior exposure to real-world datasets and, therefore, cannot quickly identify candidate data elements from the data guide and reduce those to the ones needed for a given question. Furthermore, data guides may describe several different dataset attributes that have similar yet slightly different meanings, which can be confusing for novices. This applies to the 2017 CDC immunizations data guide used in the analyzed assignment, which had several attributes with similar meanings as the `'P_NUMVRC'` attribute (which denotes the number of varicella doses). For instance, the following is mentioned on page 18 of the data guide: *"Use PDAT = 1 to identify children with adequate provider data (includes unvaccinated children)"*. It is likely that the student who made Mistake-1 in Figure 1 quickly read this line and misunderstood that `PDAT = 1` is for unvaccinated children.

For novices, learning a new programming language is a high-cognitive-load task [21]. In DS1, students learn about new libraries and functions while also learning how to work with large datasets. This raises the importance of supporting DS1 students in developing correct assumptions about the data and dataset domain, especially when they are going through the early stages of learning about new programming libraries and functions. This can be achieved by providing facilitative hints with specific details regarding the dataset domain. This also raises the importance of teaching students data literacy skills early on, such as by exposing students to real-word datasets in high school education.

Programming in Computational Notebooks. Our analysis revealed mistakes that were caused due to misconceptions about computational notebooks and defining libraries and variables inappropriately, which made them out of scope. Similar student struggles have also been cited in prior research [46]. Supporting students in understanding how the programming model of a computational notebook environment works, along with the tools to identify, understand, and address errors in code authored in notebooks can be helpful. Introducing students to debugging tools, such as JupyterLab's built-in debugger, can be helpful in this regard. Further, validating students' knowledge of computational notebooks through formative assessments before they are taught the main course concepts can equip them with the necessary skillset for programming in notebooks.

Code Optimality. Several students did not use vectorizated operations to write code in a functional paradigm, as was taught. This finding is also corroborated by prior research [48]. The autograder used in the course was not equipped to detect suboptimal code. Therefore, students received full points and were not discouraged from writing suboptimal code. In CS1, one of the widely reported causes of student difficulties is the misapplication of their prior knowledge [41, 49]. In the course we analyzed, students may have leveraged their prerequisite knowledge of iteration to DS1 programming. Therefore, DS1 instructors could help students *unlearn* some

Competency	*What it Means*	*Pedagogical Recommendations*
Programming in computational notebooks	• Can effectively program and debug in the computational notebook environment	• Demonstrate how the programming model of a given computational notebook environment works, along with debugging strategies using DS debugging tools • Validate this competency early on before students move on to the main concepts
Data literacy	• Can adequately study the data and its schema • Can relate the data to the data guide while programming • Can appropriately handle missing values in datasets from different domains	• Emphasize the conceptual principles of data cleaning and how alternative approaches are realized in code and effect analyses • Discuss a diversity of authentic datasets with missing or anomalous values and familiarize learners with the implications of various data cleaning and missing value imputation techniques • Support students in understanding the dataset domain through facilitative hints containing knowledge of the dataset domain
Programming with DS libraries	• Can choose appropriate libraries and their functions, methods, attributes, and operators • Can appropriately set function parameters	• Provide worked examples to help students understand the usage of libraries or functions; as `pandas` documentation is enriched with worked examples, specific examples could be extracted from there
Programming strategies	• Building upon 'data literacy' and 'programming with DS libraries', can devise effective strategies for problem-solving	• Remind students of the task constraints that their incorrect code does not satisfy, e.g., by highlighting the specific part of the problem-statement that the student misinterpreted and elaborating on it
Code optimality	• Can write code that scales for large datasets, using vectorized operations and functions	• Explain why certain strategies are suboptimal; demonstrate using examples of optimal code by comparing runtimes with suboptimal code over large datasets

Table 2: Implications for supporting novice data science learners corresponding to each of the five identified competencies.

CS1 concepts that do not transfer well to DS1. This can be done through both directed pedagogy and autograder tools that detect suboptimal code and provide feedback to make the code more optimized. Setting limits on runtime and memory may be useful, as done by the UNCode autograder [23]. However, the use of such limits may be problematic, as underlying computational architectures rapidly undergo iteration, necessitating the need to regularly verify that the set limits are appropriate. We recommend computing the number of times a loop executes or using static code analysis [31] to detect whether a loop is used to iterate over a DataFrame.

Programming with DS Libraries. Semantic mistakes were primarily caused due to incorrect knowledge of `pandas` functions. In CS1, the focus is on teaching language constructs and then using them to build up an understanding of algorithms and data structures. In contrast, in DS1 particular libraries (as opposed to language constructs) are taught immediately, and learners then apply their understanding of the library functions to accomplish tasks. The libraries are generally sophisticated software products, and understanding their implementation details is beyond students' skillset. For instance, `pandas` uses many levels of inheritance and is layered on top of `NumPy`, making exploration of its inner workings difficult for novices. Further, `pandas` has an order of 100 methods, and each method has an order of 10 arguments [14], which makes navigating the library's built-in functions even more challenging for novices. Teaching with worked examples of DS functions and operations can be helpful, and tools such as Pandas Tutor [40] can be used to visualize how different functions and operations transform DataFrames in a step-by-step manner.

Based on the aforementioned DS1 competencies, we provide pedagogical recommendations for supporting learners in Table 2.

Towards Automatic Feedback Tools for DS1. Scaling feedback generation is necessary to provide individual support to each learner in large courses. One way to do so is using automatic methods to detect DS1 mistakes from student code. However, as discussed in Section 2, scalable program analysis methods for DS1 are limited. Moreover, the lack of portability of unsupervised approaches from CS1 to DS1 suggests that semi-supervised or supervised approaches may be more useful for DS1. One such approach has been described by Effenberger et al. [18], which utilizes domain knowledge to define context-specific features (e.g., variable value sequences in CS1) to generate interpretable clusters of student code. Based on our findings, some useful features of DS1 code (other than ASTs) that can be used to generate interpretable clusters are: (i) *code output*, (ii) *library and function calls*, (iii) *function parameter values*, (iv) *dataset attributes and values*, and (v) *code optimality metrics*, e.g., number of loop executions. Features (ii)–(iv) can be extracted by parsing the code's AST. Features (i) and (v) require code execution. These features can further be used to analyze students' DS1 code at scale, rather than manually, as we did. These features can also be used to prompt Large Language Models [4, 32] (which can generate high-quality text-based responses to natural language prompts) along with relevant information such as the problem statement, a correct solution, the incorrect solution, and students' prior knowledge in the prompt text, to generate hints for incorrect DS1 code.

6 LIMITATIONS AND FUTURE WORK

Based on an analysis of various types of mistakes and inappropriate programming strategies of learners in a data manipulation course, we provided guidelines for improving DS1 pedagogy and assessment tools. We focused specifically on a single course with adult learners. Hence, it is possible that the reported errors were due to specific teaching decisions and may not generalize to other DS1 courses. However, given the lack of literature on DS1 errors, our results and the shared dataset serve as an important starting point to identify areas in which students require more assistance. Additionally, our analysis may not provide enough context about the factors contributing to the students' mistakes, such as their prior knowledge and motivation. Thus, there is a need to broaden such inquiry both within and across the different courses in DS programs.

REFERENCES

[1] 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*. 340–345.

[2] Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 522–527.

[3] Ruth E Anderson, Michael D Ernst, Robert Ordóñez, Paul Pham, and Ben Tribelhorn. 2015. A data programming CS1 course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 150–155.

[4] David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *Available at SSRN 4337484* (2023).

[5] Austin Cory Bart, Dennis Kafura, Clifford A Shaffer, and Eli Tilevich. 2018. Reconciling the Promise and Pragmatics of Enhancing Computing Pedagogy with Data Science. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 1029–1034.

[6] Douglas S Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas L Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, et al. 2019. nbgrader: A tool for creating and grading assignments in the Jupyter Notebook. *The Journal of Open Source Education* 2, 11 (2019).

[7] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[8] Neil CC Brown and Amjad Altadmri. 2014. Investigating novice programming mistakes: Educator beliefs vs. student data. In *Proceedings of the tenth annual conference on International computing education research*. 43–50.

[9] Robert J Brunner and Edward J Kim. 2016. Teaching data science. *Procedia Computer Science* 80 (2016), 1947–1956.

[10] 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*. 297–303.

[11] Data Carpentry. 2018. Building communities teaching universal data literacy. *Instructor Training. Building Teaching Skill: Getting Feedback* (2018).

[12] Mine Çetinkaya-Rundel and Victoria Ellison. 2021. A fresh look at introductory data science. *Journal of Statistics and Data Science Education* 29, sup1 (2021), S16–S26.

[13] Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L Santos, and Matthias Hauswirth. 2021. A curated inventory of programming language misconceptions. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 380–386.

[14] Bhavya Chopra, Anna Fariha, Sumit Gulwani, Austin Z Henley, Daniel Perelman, Mohammad Raza, Sherry Shi, Danny Simmons, and Ashish Tiwari. 2023. CoWrangler: Recommender System for Data-Wrangling Scripts. In *Companion of the 2023 International Conference on Management of Data*. 147–150.

[15] Sarah Dahlby Albright, Titus H Klinge, and Samuel A Rebelsky. 2018. A functional approach to data science in CS1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 1035–1040.

[16] Andrea Danyluk, Paul Leidig, Lillian Cassel, and Christian Servin. 2019. Acm task force on data science education: Draft report and opportunity for feedback. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 496–497.

[17] David Donoho. 2017. 50 years of data science. *Journal of Computational and Graphical Statistics* 26, 4 (2017), 745–766.

[18] Tomáš Effenberger and Radek Pelánek. 2021. Interpretable Clustering of Students' Solutions in Introductory Programming. In *International Conference on Artificial Intelligence in Education*. Springer, 101–112.

[19] Alan Fekete, Judy Kay, and Uwe Röhm. 2021. A data-centric computing curriculum for a data science major. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 865–871.

[20] Patricia I Fusch and Lawrence R Ness. 2015. Are we there yet? Data saturation in qualitative research. *The qualitative report* 20, 9 (2015), 1408.

[21] Stuart Garner. 2002. *Reducing the cognitive load on novice programmers*. Association for the Advancement of Computing in Education (AACE).

[22] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)* 22, 2 (2015), 1–35.

[23] Cristian D González-Carrillo, Felipe Restrepo-Calle, Jhon J Ramírez-Echeverry, and Fabio A González. 2021. Automatic Grading Tool for Jupyter Notebooks in Artificial Intelligence Courses. *Sustainability* 13, 21 (2021), 12050.

[24] Maurice H Halstead. 1977. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc.

[25] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*. 89–98.

[26] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin* 35, 1 (2003), 153–156.

[27] Jupyter 2022. https://jupyter.org/.

[28] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. 2010. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 107–111.

[29] Sean Kross and Philip J Guo. 2019. Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–14.

[30] 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 V. 1*. 29–35.

[31] Panagiotis Louridas. 2006. Static code analysis. *Ieee Software* 23, 4 (2006), 58–61.

[32] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 931–937.

[33] Hamza Manzoor, Amit Naik, Clifford A Shaffer, Chris North, and Stephen H Edwards. 2020. Auto-grading jupyter notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1139–1144.

[34] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[35] nbgrader 2022. https://nbgrader.readthedocs.io/en/stable.

[36] Huy Nguyen, Michelle Lim, Steven Moore, Eric Nyberg, Majd Sakr, and John Stamper. 2021. Exploring Metrics for the Analysis of Code Submissions in an Introductory Data Science Course. In *LAK21: 11th International Learning Analytics and Knowledge Conference*. 632–638.

[37] OK 2020. https://okpy.org/.

[38] Benjamin Paaßen, Jessica McBroom, Bryn Jeffries, Irena Koprinska, and Kalina Yacef. 2021. ast2vec: Utilizing Recursive Neural Encodings of Python Programs. *arXiv preprint arXiv:2103.11614* (2021).

[39] pandas 2022. https://pandas.pydata.org/.

[40] Pandas Tutor 2022. https://pandastutor.com/.

[41] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–24.

[42] Bina Ramamurthy. 2016. A practical and sustainable model for learning and teaching data science. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 169–174.

[43] Suraj Rampure, Allen Shen, and Josh Hug. 2021. Experiences Teaching a Large Upper-Division Data Science Course Remotely. In *SIGCSE '21: The 52nd ACM Technical Symposium on Computer Science Education, Virtual Event, USA, March 13-20, 2021*, Mark Sherriff, Laurence D. Merkle, Pamela A. Cutter, Alvaro E. Monge, and Judithe Sheard (Eds.). ACM, 523–528. https://doi.org/10.1145/3408877.3432561

[44] re 2022. https://docs.python.org/3/library/re.html.

[45] Kelly Rivers and Kenneth R Koedinger. 2017. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 37–64.

[46] 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*. 1–12.

[47] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[48] James Skripchuk, Yang Shi, and Thomas Price. 2022. Identifying Common Errors in Open-Ended Machine Learning Projects. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 216–222.

[49] John P Smith III, Andrea A DiSessa, and Jeremy Roschelle. 1994. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The journal of the learning sciences* 3, 2 (1994), 115–163.

[50] Greg Wilson. 2006. Software carpentry: getting scientists to write better code by making them more productive. *Computing in Science & Engineering* 8, 6 (2006), 66–69.

[51] Annika Wolff, Daniel Gooch, Jose J Cavero Montaner, Umar Rashid, and Gerd Kortuem. 2016. Creating an understanding of data literacy for a data-driven society. *The Journal of Community Informatics* 12, 3 (2016).

[52] Florin Stefan Zamfir and Emil Pricop. 2022. On the design of an interactive automatic Python programming skills assessment system. In *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE, 1–5.