


# A Systematic Review of Common Beginner Programming Mistakes in Data Engineering

Max Neuwinger

*Professorship for Open-Source Software*  
Friedrich-Alexander University Erlangen-Nürnberg  
Erlangen, Germany  
max.neuwinger@fau.de 

Dirk Riehle

*Professorship for Open-Source Software*  
Friedrich-Alexander University Erlangen-Nürnberg  
Erlangen, Germany  
dirk@riehle.org 

**Abstract**—The design of effective programming languages, libraries, frameworks, tools, and platforms for data engineering strongly depends on their ease and correctness of use. Anyone who ignores that it is humans who use these tools risks building tools that are useless, or worse, harmful. To ensure our data engineering tools are based on solid foundations, we performed a systematic review of common programming mistakes in data engineering. We focus on programming beginners (students) by analyzing both the limited literature specific to data engineering mistakes and general programming mistakes in languages commonly used in data engineering (Python, SQL, Java). Through analysis of 21 publications spanning from 2003 to 2024, we synthesized these complementary sources into a comprehensive classification that captures both general programming challenges and domain-specific data engineering mistakes. This classification provides an empirical foundation for future tool development and educational strategies. We believe our systematic categorization will help researchers, practitioners, and educators better understand and address the challenges faced by novice data engineers.

**Index Terms**—Data Engineering, Programming Errors, Novice Programmers, Systematic Review

## I. INTRODUCTION

The rapid growth of data science and artificial intelligence applications has created strong demand for skilled data engineers. According to the U.S. Bureau of Labor Statistics, data scientist positions are projected to grow by 35% from 2022 to 2032 [1], while the World Economic Forum expects similar growth of 30-35% in demand for data analysts, scientists, and engineers by 2027 [2].

As the field grows, more beginners and students are entering the profession, leading to wider ranges of skill levels among practitioners. The essential role of data engineering in the data science pipeline makes this variation in expertise particularly important. Data scientists reportedly spend up to 80% of their time resolving data issues upstream of modeling activities [3], showing how programming mistakes can significantly affect data integrity, analysis accuracy, and decision-making processes.

The challenges are especially clear in educational settings, where research shows that students struggle with abstract programming concepts and algorithm design [4]. This mix of growing demand, varying skill levels, and learning challenges highlights the need for better tools and educational practices.

Our research addresses this need by systematically identifying common beginner programming mistakes in data engineering, aiming to develop more effective educational methods.

This work was motivated by our research into domain-specific languages (DSL) for data engineering (the Jayvee<sup>1</sup> DSL of the JValue<sup>2</sup> research project). To ground our design decisions in empirical evidence and educational insights, we conducted a systematic review of existing literature on common programming mistakes in data engineering, as presented in this article. This review informs both our tool development process and the formulation of educational recommendations, which we are currently validating and will continue to evaluate through controlled experiments assessing the effectiveness of specific tool features.

Through this research and the development of the JValue project, we argue for the importance of basing both tool and educational design decisions on solid empirical evidence rather than intuition alone. This approach promises to enhance the effectiveness of data engineering tools, learning environments, and consequently, the quality and reliability of data-driven insights across various fields.

To this end, we asked the following research question:

*What are the most common programming mistakes made by beginners in data engineering projects, and what are their consequences?*

Our contributions are as follows:

- 1) The presentation of a systematic review (following Kitchenham et al. [5]) to identify and filter pertinent literature,
- 2) The quality-assured thematic analysis of the identified-as-relevant literature, and
- 3) A resulting classification of common programming mistakes by beginners of data engineering, with implications for tool design and educational strategies.

By identifying and analyzing these mistakes, we provide insights that inform the design of data engineering languages, tools, libraries, and frameworks, as well as educational interventions. We thereby expect to boost the proficiency of novice data engineers. Our analysis includes both general

<sup>1</sup>See <https://github.com/jvalue/jayvee>

<sup>2</sup>See <https://jvalue.com>

programming mistakes and domain-specific challenges unique to data engineering tasks.

This article is structured as follows. In Section 2 we first review related work. In Section 3 we present the research approach we took and in Section 4 we present the results of the research. In Section 5 we conclude the article and in Section 6 we discuss the limitations of the work presented.

## II. RELATED WORK

To the best of our knowledge, our work is the first systematic review of common beginner programming mistakes in data engineering.

Prior systematic reviews have examined different aspects of programming education. Qian & Lehman [6] reviewed students' misconceptions and difficulties in introductory programming, finding that students struggle with syntactic, conceptual, and strategic knowledge, often due to factors like unfamiliarity with syntax and lack of prior knowledge. Medeiros et al. [7] analyzed teaching and learning challenges in introductory programming, highlighting that problem-solving and mathematical ability were the most cited necessary skills, while syntax learning and lack of appropriate teaching methods were common challenges. More recently, Memarian & Doleck [8] focused on data science education specifically, examining pedagogical tools and practices through the lens of technological and pedagogical knowledge quality.

While these previous reviews have broadly covered programming education [6], teaching methodologies [7], or data science pedagogy [8], our systematic review addresses a crucial gap by examining both general and data engineering specific programming mistakes, providing a comprehensive view of the challenges faced by beginners in this domain.

## III. RESEARCH APPROACH

We performed a systematic literature review (SLR) to answer our research question. Our research focuses on student populations as representative of programming beginners, as evidenced by our analyzed literature. The vast majority of empirical studies in this space examine students in introductory programming contexts, with many of these studies explicitly equating students with novice programmers [9]–[24].

We conducted our systematic review following established guidelines for systematic literature reviews [25]–[27], while structuring our reporting according to the SEGRESS guidelines [5]. For the analysis of the identified literature, we employed Braun & Clarke's [28] thematic analysis method to derive a theory from the data. The resulting theory takes the form of a classification of programming mistakes made by beginners.

Figure 1 illustrates the iterative research process. The process begins with performing a keyword search, which results in an initial set of papers. From this list of potential papers, we apply a relevance filter, checking titles and abstracts against relevance criteria. This produces a subset of potentially relevant papers.

Next, we applied a quality filter, evaluating the full text against quality criteria, resulting in a subset of qualified papers. These qualified papers then undergo thematic analysis. Throughout the process, we perform backward and forward snowballing to identify additional relevant articles [29].

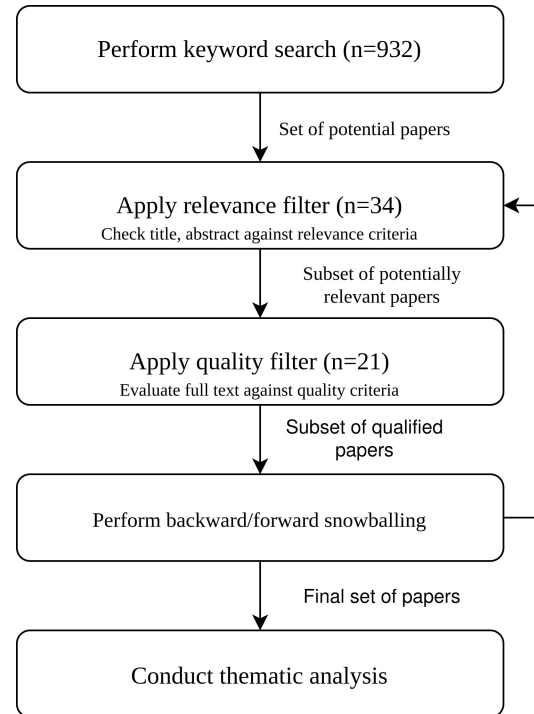


Fig. 1. Visualization of data extraction and synthesis

### A. Literature Search

Our search approach utilized two major electronic databases: IEEE Xplore and Scopus. We developed two search strings to capture both data engineering-specific and general programming errors that could be relevant to data engineering:

- 1) ("data engineering" OR "data science") AND ("mistake" OR "fault" OR "error" OR "misconception" OR "bug") AND ("beginner" OR "novice" OR "student")
- 2) "programming" AND ("mistake" OR "error") AND ("beginner" OR "novice" OR "student") AND "data"

The first search string, focusing specifically on data engineering and data science mistakes, yielded 98 results in Scopus and 107 in IEEE Xplore. The second search string, which captured broader programming mistakes in data-related contexts, returned 458 results in Scopus and 269 in IEEE Xplore.

### B. Initial Selection Criteria

Before applying our detailed relevance and quality filters, we established the following practical criteria for study selection:

- Language: We limited our review to English-language publications due to the researchers' language capabilities and to ensure consistent interpretation of technical terminology.

- **Accessibility:** All relevant papers were accessible through institutional subscriptions or were openly available, requiring no direct author contact.
- **Publication Type:** We included only peer-reviewed publications and full conference papers to ensure a baseline of scientific rigor, excluding non-peer-reviewed publications and brief conference abstracts.
- **Lack of Empirical Data:** Studies relying solely on anecdotal evidence or opinion pieces without empirical support were excluded.
- **Methodological Weaknesses:** Research with unclear designs, inadequate sample sizes, or insufficient detail on data analysis methods was excluded. Studies that did not address validity and reliability issues were also omitted.

### C. Inclusion and Exclusion Criteria

We adopted a two-phase approach to assess the relevance and quality of the identified studies.

1) *Relevance Filter:* The relevance filter was applied to ensure that the selected studies aligned with the core focus of our research. The criteria for this phase were as follows:  
Inclusion Criteria:

- **Topic Relevance:** Studies were included if they met one of the following criteria:
  - Focused directly on programming mistakes in data science or data engineering contexts
  - Provided comprehensive analysis of novice programming mistakes in languages commonly used in data engineering (Python, R, SQL, Java)
  - Offered insights into fundamental programming concepts and error patterns that impact data manipulation, analysis, and processing tasks
- **Target Audience:** Research should address novice programmers.
- **Research Type:** Empirical studies, case studies, theoretical papers, reviews, and meta-analyses contributing to understanding or defining best practices in error prevention or management in programming were included.
- **Educational Context:** Papers discussing lessons learned from teaching data engineering classes were considered relevant, if they offered insights into the programming mistakes.

2) *Quality Filter:* Studies that passed the relevance filter were then subjected to a rigorous quality assessment. The criteria for this phase were as follows:

Inclusion Criteria:

- **Methodological Rigor:** Studies must demonstrate clear research design and systematic data collection methods. For quantitative studies, this included appropriate sample sizes and statistical analyses. For qualitative studies, this meant well-documented methodological approaches such as grounded theory, thematic analysis, or case study protocols with clear data collection and analysis procedures.
- **Publication Quality:** Peer-reviewed publications in journals or conference proceedings in the field of data engineering or computer science were included.
- **Comprehensive Analysis:** Studies providing in-depth analysis of programming mistakes, including causes, consequences, and potential solutions, along with detailed discussions of findings, practical implications, and future research directions were favored.

Exclusion Criteria:

To provide a nuanced evaluation of each paper's relevance and quality, a scoring system was implemented. Each criterion was marked as "yes" (2 points), "partial" (1 point), or "no" (0 points). The final score for each paper was calculated as the square root of the sum of squared values for each criterion. This two-phase approach ensured a comprehensive and rigorous selection process, aiming to provide a holistic view of programming mistakes in data engineering.

### D. Data Extraction and Analysis

Selected papers were accessed through various online academic platforms using university credentials. The qualitative data analysis tool QDAcity<sup>3</sup> was used for detailed data analysis.

We employed Braun & Clarke's [28] thematic analysis approach to systematically identify, analyze, and report patterns (themes) within the data. This method involves six phases:

- 1) **Familiarization with the data:** We thoroughly read and re-read the selected papers, making initial notes on potential codes and themes.
- 2) **Generating initial codes:** We systematically coded interesting features across the entire dataset, collating data relevant to each code.
- 3) **Searching for themes:** We collated codes into potential themes, gathering all data relevant to each potential theme.
- 4) **Reviewing themes:** We checked if the themes work in relation to the coded extracts and the entire dataset, generating a thematic 'map' of the analysis.
- 5) **Defining and naming themes:** We conducted ongoing analysis to refine the specifics of each theme, generating clear definitions and names for each theme.
- 6) **Producing the report:** We selected compelling extract examples, conducted final analysis of selected extracts, related the analysis back to the research question and literature, and produced a scholarly report of the analysis.

The analysis involved iterative coding of the papers to identify and categorize different types of programming mistakes. This dynamic process allowed for continuous refinement and redefinition of categories as new patterns emerged from the data. We employed an inductive approach in our coding process [30]. This method allowed themes to emerge organically from the data itself, without preconceived notions or existing frameworks. As we progressed through the analysis, we consistently revisited and refined our categorizations to

<sup>3</sup>See <https://qdcity.com>

ensure they accurately reflected the patterns observed in the collected data.

To ensure the reliability and validity of our analysis, we employed several strategies:

- Constant comparison: We continuously compared new data with existing codes and themes, refining our analysis throughout the process [30].
- Negative case analysis: We actively searched for and discussed cases that did not fit within the emerging patterns to ensure a comprehensive analysis [31].

Throughout the analysis process, both authors held regular review meetings to discuss emerging themes, validate coding decisions, and ensure consistent interpretation of the data.

Our use of thematic analysis aligns with best practices in qualitative research in software engineering [25], [30], allowing for a rigorous and systematic exploration of the literature. The resulting themes form the basis of our classification of programming mistakes, providing a structured framework for understanding and addressing common challenges faced by novice data engineers.

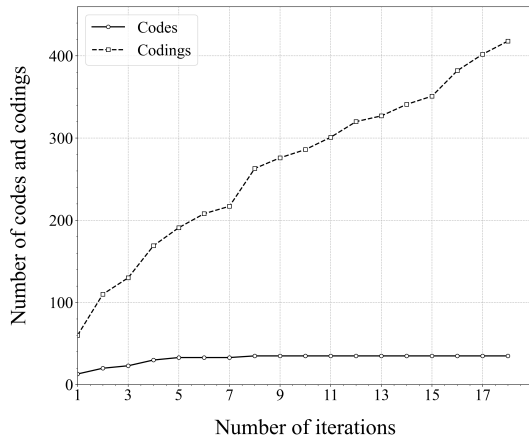


Fig. 2. Codes and codings over time, showing theoretical saturation [32]

Figure 2 illustrates the progression of our coding process over time. The graph shows the cumulative number of unique codes (solid line) and total codings (dotted line) as we analyzed each paper. The declining and eventual lack of growth of codes over time indicates theoretical saturation, suggesting that we reached a point where additional data analysis was not yielding new insights or categories. This saturation provides confidence in the comprehensiveness of our thematic analysis and the resulting classification of programming mistakes.

#### IV. RESULTS

This literature review analyzed 21 publications spanning from 2003 to 2024, comprising 6 journal articles, 14 conference proceedings papers, and 1 symposium article (see A). Given the limited literature specifically addressing programming mistakes in data engineering, our review encompasses both domain-specific studies and broader research on beginner programming mistakes that are relevant to data science contexts.

The temporal distribution of these publications, shown in Figure 3, reveals the evolution of research interest in this area.

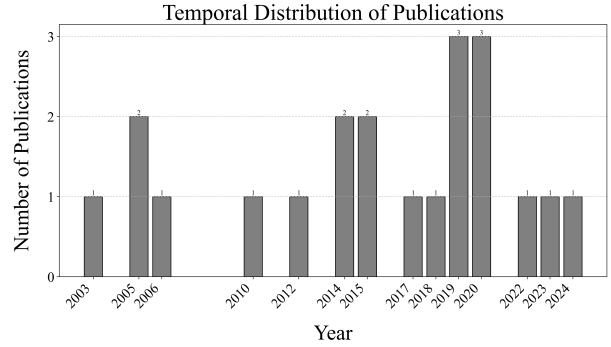


Fig. 3. Distribution of Publications by Year

The broader literature covers a diverse range of programming languages illustrated in figure 4:

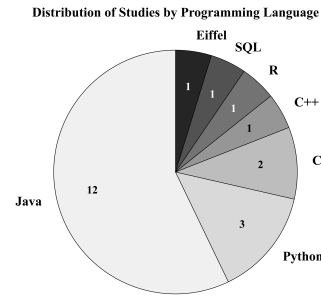


Fig. 4. Distribution of Studies by Programming Language

Our analysis shows Java as the predominant programming language in the studied papers, followed by Python. This aligns with recent industry research by 365 Data Science [33] which found that SQL, Python, and Java remain the most crucial programming languages in data engineering, with SQL appearing in 79.4% of job postings, Python in 73.7%, and Java in 22.6% of listings. Java’s strong presence in our academic literature likely reflects its historical importance in building scalable data processing systems and its widespread use in enterprise data engineering tools like Apache Hadoop and Apache Spark.

The research methodologies employed in these studies reveal a diverse range of approaches, with 13 quantitative studies and 8 mixed-methods studies. The quantitative studies can be further divided into three main subcategories:

- Post-hoc data analysis: 6 studies analyzed programming logs to identify error patterns and frequencies [9]–[11], [16], [20], [34].
- Large-scale analysis: 3 studies conducted extensive error analyses in controlled environments [12], [14], [35].
- Targeted analysis of specific programming tasks: 4 studies focused on detailed evaluations of specific assignments or programming patterns [15], [17], [22], [36].

These quantitative studies primarily utilized numerical data and statistical methods to identify trends and draw conclusions. The remaining 8 mixed-methods studies [13], [18], [19], [21], [23], [24], [37], [38] combined quantitative techniques with qualitative methods such as thematic analysis, interviews, and think-aloud protocols. This methodological diversity enabled us to identify both general programming mistakes that persist across contexts and those specific to data engineering tasks.

The following sections provide a detailed analysis of the common mistakes made by beginners in data engineering, as summarized in Table I. These sections are organized according to the main themes identified in our research. Each section explains a specific mistake type within these categories, providing definitions, examples, and discussions to offer a comprehensive understanding of the challenges faced by novice data engineers. A detailed mapping between each identified mistake type and the supporting literature can be found in Appendix A.

### A. Domain-Specific Mistakes (1)

The Domain-Specific Mistakes are about programming mistakes specific to data engineering and data science.

#### (1.1) Dataset Misunderstandings

Definition: Mistakes arising from misunderstanding the structure, schema, or specific attributes of the dataset.

Examples:

- *Incorrect Attribute Selection*: In one study [37], a common mistake was selecting the wrong dataset attribute. Students often used the attribute 'PDAT' instead of 'P\_NUMVRC', misinterpreting 'PDAT' to indicate the number of varicella doses received rather than adequate provider data.

- *Incorrect Value Usage*: Another frequent mistake involved using incorrect dataset values. For example, in the same study [37], students used 'male' and 'female' instead of numerical codes (1 and 2) when filtering rows by gender.

#### (1.2) Faulty Data Analysis Logic

Definition: Mistakes in implementing data analysis algorithms or logical data processing.

Examples:

- *Incorrect Use of DataFrame Functions*: In one instance [37], students incorrectly used the `where()` function from pandas to select rows, which "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."

- *Incorrect DataFrame Operations*: Another example involved "dividing a DataFrame slice by another slice rather than dividing their lengths to compute a given ratio." [37]

#### (1.3) Incorrect Data Handling

Definition: Errors in manipulating or accessing data, leading to incorrect data processing.

Examples:

- *Incorrect Conditions*: An example from [37] involved using `df['P_NUMVRC']!=0` instead of `df['P_NUMVRC']>0` to select rows indicating children who received at least one varicella dose. This condition fails because it does not correctly account for missing values, which are counted in addition to the number of rows where 'P\_NUMVRC' is greater than 0.

- *Not Accounting for NaN Values*: Another common mistake was replacing all missing values with 0, which led to incorrect calculations. For instance, when 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 were incorrectly counted towards children receiving zero doses. [37]

- *Misunderstanding Function Outputs*: Students also made mistakes in using the `groupby()` function incorrectly, where they misunderstood the output format of the object returned by `groupby` and performed incorrect operations in subsequent lines of code. [37]

#### (1.4) Misunderstanding Data Types

Definition: Incorrect handling or interpretation of different data types.

Examples:

- *Format Errors*: Students often made errors related to data formats, such as trying to substitute categorical data into a histogram, which is not appropriate. An example error message observed was: "Error in `mutate_impl(.data, dots)` : Evaluation error: non-numeric argument to binary operator". [34]

### B. Strategic Errors (2)

#### (2.1) Suboptimal Coding

Definition: Inefficient use of coding constructs, leading to suboptimal performance, especially with large datasets. This includes not using appropriate vectorized functions for data manipulation.

Examples:

- *Inefficient Loop Usage*: A prevalent issue identified in multiple papers ([37], [19]) is the use of for-loops to iterate over DataFrame rows, instead of using vectorized operations. For instance, students used for-loops to count rows satisfying a condition, which is significantly slower than applying a Boolean mask.

- *Complex Solutions*: Some students submitted unnecessarily complex solutions for problems that could be solved with simpler, more efficient code. This was observed in [19], where students wrote complex, multi-line solutions instead of concise, optimal code.

TABLE I: Summary of Common Beginner Programming Mistakes in Data Engineering.

| Top-level Code                                     | Mistake Type                               | Definition   | Occurrences |
|--|--|--|-------------|
| <b>Domain-Specific Mistakes (1)</b>                | (1.1) Dataset Misunderstandings            | Mistakes arising from misunderstanding the dataset, its schema, or associated data guides.   | 1           |
|  | (1.2) Faulty Data Analysis Logic           | Mistakes in implementing data analysis algorithms.   | 2           |
|  | (1.3) Incorrect Data Handling              | Mistakes in manipulating data.   | 2           |
|  | (1.4) Misunderstanding Data Types          | Incorrect handling of different data types.  | 2           |
| <b>Strategic Errors (2)</b>                        | (2.1) Suboptimal Coding                    | Inefficient use of coding constructs.  | 3           |
|  | (2.2) Wrong Algorithm                      | Choosing the incorrect algorithm for the task.   | 5           |
| <b>Misinterpretation Errors (3)</b>                | (3.1) Incorrect Programming Assumptions    | Making wrong assumptions about how programming constructs work.  | 10          |
|  | (3.2) Task Misunderstanding                | Misinterpreting the problem statement or requirements.   | 3           |
| <b>Environment and Language Misconceptions (4)</b> | (4.1) Incorrect Language Understanding     | Misunderstandings about the overall functionality, rules, and behaviors of the programming language, as well as the development environment in which novices work. | 18          |
|  | (4.2) Library Misuse                       | Incorrect use of libraries or failure to import necessary libraries.   | 4           |
|  | (4.3) Path and I/O Errors                  | Incorrectly specifying file paths or managing file I/O.  | 4           |
| <b>Logical Errors (5)</b>                          | (5.1) Incorrect Loop Conditions            | Errors in loop conditions, leading to infinite loops or off-by-one errors.   | 2           |
|  | (5.2) Faulty Conditional Logic             | Errors in if-else statements that cause incorrect branching.   | 3           |
|  | (5.3) Off-by-One / Out-of-Bounds Errors    | Misplacing indices in loops or arrays.   | 6           |
| <b>Semantic Errors (6)</b>                         | (6.1) Incorrect Function Usage             | Using functions incorrectly.   | 12          |
|  | (6.2) Type Mismatches                      | Assigning or passing incorrect data types to variables or functions.   | 15          |
|  | (6.3) Variable Scope Issues                | Using variables outside their defined scope.   | 5           |
| <b>Syntax Errors (7)</b>                           | (7.1) Incorrect Loop & if-else Syntax      | Errors in the syntax of loops or if else statements.   | 5           |
|  | (7.2) Incorrect Operators                  | Using operators incorrectly.   | 12          |
|  | (7.3) Invalid Keywords Usage               | Misusing language keywords.  | 3           |
|  | (7.4) Missing Semicolons                   | Omitting semicolons where required.  | 7           |
|  | (7.5) Unbalanced Delimiters                | Unbalanced brackets, parentheses, or braces.   | 11          |
|  | (7.6) Variable not declared or initialized | Using variables that are not declared or initialized.  | 9           |
| <b>Sloppiness Errors (8)</b>                       | (8.1) Typographical Errors                 | Simple typos in code.  | 7           |
| <b>Memory Management Mistakes (9)</b>              | (9.1) Memory Language Misconceptions       | Misunderstanding memory management in programming languages.   | 3           |
|  | (9.2) Memory Leaks                         | Failing to deallocate memory.  | 2           |

### (2.2) Wrong Approach

Definition: Using an inappropriate or fundamentally flawed algorithm to solve a problem, which often leads to incorrect results or inefficient solutions.

Examples:

- *Incorrect Ratio Computation*: An example from [37] involved a student computing a ratio by dividing one DataFrame slice by another, rather than dividing their lengths. This fundamental misunderstanding of the problem led to incorrect results.

- *Incorrect Maximum Value Calculation*: In another example

from [17], a student's algorithm failed to correctly handle edge cases in determining the maximum value in an array, leading to incorrect outputs when all values were negative.

### C. Misinterpretation Errors (3)

#### (3.1) Incorrect Programming Assumptions

Definition: Misconceptions about how specific programming constructs or functions work.

Examples:

- *Bitwise Operators Misuse*: In [37], students incorrectly used the bitwise AND operator on DataFrame objects, which is not valid. The correct approach involves combining Boolean masks with appropriate precedence rules.

- *Incorrect Function Assumptions*: Another example from [19] involved students misunderstanding how functions like `getchar()` and `scanf()` handle input buffers, leading to persistent errors in their programs.

#### (3.2) Task Misunderstanding

Definition: Misinterpreting the problem statement, leading to incorrect implementation.

Examples:

- *Ratio Calculation Errors*: In [37], students misunderstood the problem requirements, calculating ratios incorrectly by including all vaccinated children instead of only those who did not contract chickenpox.

- *Incorrect Output Format*: Papers [37], [15], [19], [38], [17] noted that students often misunderstood the required output format, leading to errors like returning rounded ratios instead of exact values or incorrect dictionary key ordering.

### D. Environment and Language Misconceptions (4)

#### (4.1) Incorrect Language Understanding

Definition: Making incorrect assumptions about how the used programming language works, leading to fundamental errors. This is a major issue mentioned across many studies and examples.

Examples:

- *Misunderstanding Control Flow*: A common error is control flow reaching the end of a non-void method without returning a value. For example, in [9], a method was defined as follows:

```
public int foo(int x) {
    if (x < 0) return 0;
    x += 1;
}
```

This leads to a compilation error because the method does not return a value in all code paths.

- *Incorrect Method Invocation*: In [9], students often attempted to invoke non-static methods as if they were static.

- *Parameter Types in Method Calls*: Including parameter types when invoking a method is another error noted in [9].

- *Misunderstanding Scope Rules*: In Python, misunderstanding

scope rules can lead to errors like referencing a variable before assignment.

#### (4.2) Library Misuse

Definition: Incorrect use of libraries, such as failing to import them correctly or misunderstanding their usage.

Examples:

- *Missing Imports*: As seen in [37], students often forgot to import necessary libraries, which led to runtime errors. An example error is failing to import 'pandas' but still attempting to use 'pd.DataFrame'.

- *Improper Use of Libraries*: Another example is not defining commonly used variables or aliases (like 'df' for DataFrame), causing confusion and errors in subsequent code cells.

#### (4.3) Path and I/O Errors

Definition: Errors related to incorrectly specifying file paths or managing input/output operations.

Examples:

- *Hardcoded Paths*: Students often hardcoded absolute paths that were inaccessible on other machines, such as:

```
import pandas as pd
df = pd.read_csv('/path/to/data.csv')
```

- *Misunderstanding Input Operations*: Spohrer and Soloway [39] found that novices often misunderstood how input operations handle whitespace. For instance, some students failed to understand that whitespace in the input data is a character that can be read, not just a separator that is automatically ignored. This high-frequency bug was caused by a misunderstanding of how the READLN statement works in the context of characters. Some novices thought that READLN ignores all whitespace when parsing an input line and assigning values to a sequence of variables, possibly because that's how READLN works on a sequence of numeric inputs.

### E. Logical Errors (5)

Logical errors are mistakes in the code's logic that lead to incorrect behavior. Common examples include incorrect loop conditions, which can result in infinite loops or off-by-one errors, and faulty conditional logic in if-else statements, leading to incorrect branching. Notably, off-by-one or out-of-bounds errors, where indices in loops or arrays are misplaced, occur frequently and can cause significant issues by accessing elements outside their intended range.

### F. Semantic Errors (6)

Semantic errors in data engineering arise from the misuse of functions, operators, or data types, leading to incorrect program behavior. A significant issue in this category is the incorrect usage of functions, particularly regarding the number of arguments and their types. This mistake is especially prevalent among novices and can lead to numerous errors. Type mismatches, which involve assigning or passing incorrect data types to variables or functions, are almost universally encountered in the literature and represent a major source of misunderstanding for beginners learning a programming

language. Additionally, variable scope issues, where variables are used outside their defined scope, can result in unexpected behavior.

### G. *Syntax Errors (7)*

Syntax errors in data engineering are mistakes that violate the grammatical rules of the programming language, preventing code from compiling or running. These errors are ubiquitous and occur frequently, but are typically quick and easy to fix. They include issues such as missing semicolons, incorrect operators (e.g., confusing assignment with comparison), errors in loop and if-else syntax, invalid keyword usage, and undeclared or uninitialized variables. Among these, unbalanced delimiters - such as mismatched parentheses, curly braces, or square brackets - stand out as the most significant and common error. Despite their prevalence, the relative ease of identifying and correcting syntax errors makes them less problematic in the long run compared to logical or semantic errors. However, their frequency can still significantly impact productivity and learning curves for novice data engineers, highlighting the importance of robust syntax checking tools and clear error messages in development environments.

### H. *Sloppiness/Typographical Errors (8)*

Sloppiness or typographical errors are mistakes that occur due to carelessness or simple typing errors in the code. These errors can afflict programmers of all skill levels, from novices to experts. While often easy to fix once detected, they can create a cascade of problems if left unnoticed. Typos in variable names, function calls, or numerical values can lead to unexpected behavior, logical errors, or even syntax errors. The insidious nature of these mistakes lies in their potential to introduce subtle bugs that may not be immediately apparent, potentially causing significant issues in data processing or analysis down the line. However, with proper attention to detail and the use of code review practices or linting tools, these errors can usually be caught and corrected quickly, minimizing their impact on the overall data engineering process.

### I. *Memory Management Errors (9)*

Memory management errors occur when programmers mishandle memory allocation and deallocation, which, though less frequently discussed, can have significant impacts. Two common mistakes include *memory language misconceptions* and *memory leaks*.

Memory misconceptions arise when students assume that declared objects are automatically allocated memory without instantiation [38] or attempt to access memory via unallocated pointers [19]. These misunderstandings can lead to hard-to-detect bugs.

Memory leaks occur when students fail to deallocate allocated memory, leading to resource wastage and potential system slowdowns [19]. This mistake can have serious performance consequences, especially in long-running applications.

## V. DISCUSSION

The findings of this study reveal a complex landscape of programming mistakes encountered by novices. While some mistakes are common across various programming domains, others are particularly unique to data engineering tasks.

### A. *Discussion of programming mistakes*

General programming mistakes, such as syntax mistakes involving missing semicolons or unbalanced delimiters (IV-G), are omnipresent among novice programmers regardless of their specific field. Similarly, logical errors related to conditional statements, loops, or off-by-one errors are frequent occurrences (IV-E). These types of mistakes, while common, often represent opportunities for targeted instruction and early intervention in educational settings, where students could benefit from exercises and real-time feedback that reinforce proper syntax and logical reasoning skills.

More significant and persistent challenges emerge in areas specific to data engineering. A primary concern is the misunderstanding or mishandling of datasets (IV-A). This issue often arises due to a lack of familiarity with data structures, insufficient domain knowledge, or inadequate understanding of data manipulation techniques. In educational contexts, addressing these issues through curriculum adjustments, such as including more hands-on training with diverse datasets and focused instruction on data processing techniques, could help students build a deeper understanding. By incorporating problem-based learning or interactive simulations into teaching, educators can provide practical exposure that reduces the likelihood of these errors.

Another critical area of difficulty lies in algorithm selection and implementation. Novices frequently struggle to identify the most suitable algorithms for specific data engineering tasks and may face challenges in correctly implementing these algorithms (IV-B). This issue is often compounded by inefficient coding practices, such as failing to utilize vectorized functions in favor of less efficient manual loops (IV-B). Educators could address these gaps by designing curricula that focus on algorithmic thinking and computational efficiency, using real-world examples to show the impact of suboptimal practices. Educational tools that visualize the performance differences between different coding strategies can also help students better understand these concepts.

Data type handling and file path management present particularly relevant challenges in the data engineering context. Given the diverse nature of data sources and formats, students must develop a strong understanding of data types (IV-A) and file handling (IV-D). Educational interventions that provide students with a variety of data-handling scenarios could help mitigate these issues. For example, structured labs or assignments that focus on reading, writing, and managing data from different sources can reinforce these critical skills, preparing students to handle real-world data challenges.

Many mistakes also simply stem from a basic lack of understanding of the programming language and environment. This deficit manifests in various ways, from misunderstanding



basic programming constructs to incorrectly assuming how specific functions operate. In educational settings, these issues could be addressed by incorporating detailed instruction on language-specific quirks and encouraging exploratory learning, where students test their assumptions in sandbox environments. Educators should also emphasize the importance of libraries and memory management, which are critical for efficient data processing, through hands-on labs and debugging exercises.

Interestingly, this study highlights that novices frequently struggle with interpreting problem statements correctly (IV-C). This finding underscores the importance of developing not only technical skills but also analytical and comprehension abilities. Educators could address this challenge by incorporating assignments that require students to break down and restate complex problems, thus ensuring a solid understanding of the task at hand. This approach could help students learn how to approach data engineering tasks methodically, reducing the risk of misinterpretation.

### *B. Implications for educators and industry*

These findings have several practical implications for both tool development and educational practices in data engineering. Integrated Development Environments (IDEs) can be enhanced with features that specifically target common novice errors in data engineering. For example, advanced debugging tools, real-time feedback on data manipulation, and suggestions for efficient coding practices could potentially reduce the occurrence of these mistakes. Similarly, educational platforms could integrate these features to give students immediate feedback during coding exercises, reinforcing correct practices while they learn.

In the industry, companies should consider incorporating these findings into their onboarding programs for new data engineers. This could include workshops on common pitfalls and best practices in data engineering, ensuring that novices are well-prepared to handle the challenges they will face. Likewise, educational institutions should incorporate this knowledge into their curricula, aligning their teaching with real-world challenges to better prepare students for professional environments. Establishing strong mentorship programs, encouraging peer code reviews, and utilizing interactive learning platforms can help novices in both academia and industry environments learn from experienced data engineers, reducing the frequency of common mistakes.

By addressing these common mistakes through targeted educational interventions, tool features, and supportive industry practices, we can significantly enhance the proficiency and effectiveness of novice data engineers. This, in turn, will lead to more reliable and accurate data processing, ultimately contributing to better data-driven decision-making across various fields.

## VI. LIMITATIONS

We discuss the limitations of our work using Guba & Lincoln's [40] quality criteria for qualitative research. The four

quality criteria for qualitative research (credibility, transferability, dependability, and confirmability) mirror the traditional four quality criteria for quantitative research (internal validity, external validity, reliability, and objectivity).

### *A. Credibility*

The credibility of findings in qualitative research (internal validity in quantitative research) rests on the connection (ideally isomorphism) between the studied phenomena and the empirical data gathered about the phenomena. Data quality assurances like prolonged engagement and persistent observation support credibility and are fulfilled in our work through the accuracy of our search queries and selection filters: We captured all work (to the extent that search engines could find it) and hence identified all relevant data we needed for our analysis, similarly to how prolonged engagement and persistent observation would have afforded it to an investigator of a primary study (rather than a secondary study like ours).

A strength of a systematic review is the built-in triangulation afforded by the different data sources (articles) utilized. Each article provides both investigator triangulation (different people's work) and data triangulation (different underlying data sets for the findings) to stabilize the findings. Even if any article's particular findings were off target, the rest would reign them in. The theoretical saturation our data analysis achieved shows that our work, based on this broad and deep primary data, would not have found much to add if we had continued, and thereby shows completion of our analysis.

### *B. Transferability*

The transferability of findings in qualitative research (external validity in quantitative research) is about changing contexts and still being able to apply the findings. Transferability is less important for us; we believe the goal of improving data engineering is wholly sufficient for our work. Still, the main quality criterion of a valid context transfer, called thick description, is built-in into the empirical data (the articles) our research is built on. The breadth and depth of data and interpretation in our primary materials is ensured by the quality of the research publications we identified and built the systematic review from. The combination of having exhausted the search space (available articles) together with having finished the possible interpretation (theoretical saturation) shows that our data was as thick as it could get. It's important to note that most of our findings are derived from studies of beginner student courses. The transferability of these findings to beginners or novices in professional data engineering contexts may be limited. The challenges faced by students in controlled academic environments may differ from those encountered by professionals learning on the job. Additionally, our findings may not fully represent all aspects of data engineering, as the field encompasses a broad range of skills beyond programming, including data modeling, Extract Transform Load (ETL) processes, and data pipeline management.

### C. Dependability

The dependability of findings in qualitative research (reliability in quantitative research) is the stability of findings after all random variation has been removed. Here, a systematic review really shines to the extent that the search query and selection filters allowed the investigators to identify all relevant work. The replication of the same search query and article selection at a different point in time would only be different for the primary data to the extent that new work would have been found (as is possible at a future point in time).

### D. Confirmability

The confirmability of findings in qualitative research (objectivity in quantitative research) is the independence of findings from a particular investigator. We applied a form of investigator triangulation in that the second author reviewed and confirmed the first author's work along multiple dimensions: The second author both reviewed and confirmed (a) the search queries, selection filters, and their results and (b) the qualitative data analysis of the first author. In addition, working from a corresponding work log (laboratory book), the second author audited the steps taken by the first author to arrive at these findings (confirmability audit) and confirmed the findings.

### E. More limitations

In addition to the qualitative research criteria discussed above, our study has several specific limitations typical of systematic literature reviews. Our focus on English-language publications may have introduced a language bias, potentially excluding relevant studies published in other languages. While the second author provided supervision and feedback throughout the coding process, we acknowledge that a more formalized inter-coder reliability assessment could have further strengthened our methodological rigor. However, the thorough review process and regular discussions between authors helped ensure consistency in data extraction and analysis.

A key limitation of our study is the relatively limited number of studies specifically focused on data engineering mistakes. To address this limitation, we deliberately included broader programming studies that provided insights into mistakes made with languages commonly used in data engineering (such as SQL for data manipulation and Java for data processing frameworks). This methodological choice allowed us to build a comprehensive foundation of programming mistakes while identifying patterns across technologies relevant to data engineering practice. However, many of our findings are derived from more general programming or data science contexts and may not fully capture the unique challenges in data engineering. This limitation is compounded by the fact that most of the studies we reviewed focused on beginner programming courses in academic settings. It's not clear how well these findings translate to beginners or novices in professional data engineering contexts.

The review may also be affected by publication bias, as studies with positive or significant results are more likely to be published than those with negative or non-significant results.

Furthermore, data engineering is a rapidly evolving field, and some of the older studies may not reflect the current state of tools and practices in the industry. Our study may also not fully capture all aspects of data engineering, as the field encompasses a wide range of skills and tools beyond just programming.

These limitations highlight the need for future research that focuses more specifically on data engineering contexts, particularly in professional settings, to validate and expand upon our findings.

## VII. CONCLUSION

This study highlights common programming mistakes made by beginners in data engineering, revealing challenges that range from basic syntax mistakes to complex, domain-specific issues. While general programming mistakes are widespread, the most significant hurdles for novices are unique to data engineering: misunderstanding datasets, inefficient algorithm implementation, and misinterpretation of data-specific problems.

Our findings have important implications not only for the development of data engineering tools but also for educational practices in the field. These insights can guide the creation of more intuitive programming languages and libraries specifically tailored to data engineering tasks, potentially mitigating common mistakes through improved design and functionality. Simultaneously, they underscore the need for educational interventions that focus on practical skill development and problem-solving abilities in real-world data engineering scenarios.

Future research could explore how addressing these common mistakes through educational methods impacts the efficiency and effectiveness of novice data engineers in both academic and real-world projects. Additionally, investigating the interplay between tool development, educational platforms, and practical experience could provide valuable insights for refining both technical tools and teaching methodologies.

This study serves as a foundation for improving not only the toolset available to aspiring data engineers but also the educational frameworks that support their growth. Ultimately, this dual approach will contribute to the development of more proficient professionals capable of addressing the complex data challenges of tomorrow, fostering a generation of data engineers who are better prepared for both technical and practical demands.

## REFERENCES

- [1] U.S. Bureau of Labor Statistics, "Occupational outlook handbook: Data scientists," <https://www.bls.gov/ooh/math/data-scientists.htm>, 2024, accessed: January 05, 2025.
- [2] World Economic Forum, "The future of jobs report 2023," World Economic Forum, Tech. Rep., April 2023, accessed: January 05, 2025. [Online]. Available: [https://www3.weforum.org/docs/WEF\\_Future\\_of\\_Jobs\\_2023.pdf](https://www3.weforum.org/docs/WEF_Future_of_Jobs_2023.pdf)
- [3] B. Howe, M. Franklin, L. Haas, T. Kraska, and J. Ullman, "Data science education: We're missing the boat, again," in *2017 IEEE 33rd international conference on data engineering (ICDE)*. IEEE, 2017, pp. 1473–1474.

- [4] M. Konecki, "Problems in programming education and means of their improvement," *DAAAM international scientific book*, vol. 2014, pp. 459–470, 2014.
- [5] B. Kitchenham, L. Madeyski, and D. Budgen, "Segress: Software engineering guidelines for reporting secondary studies," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 1273–1298, 2023.
- [6] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, oct 2017. [Online]. Available: <https://doi.org/10.1145/3077618>
- [7] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão, "A systematic literature review on teaching and learning introductory programming in higher education," *IEEE Transactions on Education*, vol. 62, no. 2, pp. 77–90, 2018.
- [8] B. Memarian and T. Doleck, "Data science pedagogical tools and practices: A systematic literature review," *Education and information technologies*, vol. 29, no. 7, pp. 8179–8201, 2024.
- [9] N. C. Brown and A. Altadmri, "Novice java programming mistakes: Large-scale data vs. educator beliefs," *ACM Transactions on Computing Education (TOCE)*, vol. 17, no. 2, pp. 1–21, 2017.
- [10] —, "Investigating novice programming mistakes: Educator beliefs vs. student data," in *Proceedings of the tenth annual conference on International computing education research*, 2014, pp. 43–50.
- [11] R. Smith and S. Rixner, "The error landscape: Characterizing the mistakes of novice programmers," in *Proceedings of the 50th ACM technical symposium on computer science education*, 2019, pp. 538–544.
- [12] A. Altadmri and N. C. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," in *Proceedings of the 46th ACM technical symposium on computer science education*, 2015, pp. 522–527.
- [13] P. O. Jegede, E. A. Olajubu, O. O. Bakare, I. O. Elesemoyo, and J. Owolabi, "Analysis of syntactic errors of novice python programmers in a nigeria university," in *Science and Information Conference*. Springer, 2023, pp. 285–295.
- [14] M. Hristova, A. Misra, M. Rutter, and R. Mercuri, "Identifying and correcting java programming errors for introductory computer science students," *ACM Sigcse Bulletin*, vol. 35, no. 1, pp. 153–156, 2003.
- [15] A. S. Júnior, J. C. A. de Figueiredo, and D. Serey, "Analyzing the impact of programming mistakes on students' programming abilities," in *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 30, no. 1, 2019, p. 369.
- [16] P. Denny, A. Luxton-Reilly, and E. Tempero, "All syntax errors are not equal," in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 2012, pp. 75–80.
- [17] A. Ettles, A. Luxton-Reilly, and P. Denny, "Common logic errors made by novice programmers," in *Proceedings of the 20th Australasian Computing Education Conference*, 2018, pp. 83–89.
- [18] D. McCall and M. Kölling, "Meaningful categorisation of novice programmer errors," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, 2014, pp. 1–8.
- [19] E. Albrecht and J. Grabowski, "Sometimes it's just sloppiness-studying students' programming errors and misconceptions," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 340–345.
- [20] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debugging among novice computer science students," in *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, 2005, pp. 84–88.
- [21] E. L. Kiran and K. M. Moudgalya, "Evaluation of programming competency using student error patterns," in *2015 International Conference on Learning and Teaching in Computing and Engineering*. IEEE, 2015, pp. 34–41.
- [22] J. Jackson, M. Cobb, and C. Carver, "Identifying top java errors for novice programmers," in *Proceedings frontiers in education 35th annual conference*. IEEE, 2005, pp. T4C–T4C.
- [23] D. McCall and M. Kölling, "A new look at novice programmer errors," *ACM Transactions on Computing Education (TOCE)*, vol. 19, no. 4, pp. 1–30, 2019.
- [24] D. Miedema, E. Aivaloglou, and G. Fletcher, "Identifying sql misconceptions of novices: Findings from a think-aloud study," *ACM Inroads*, vol. 13, no. 1, pp. 52–65, 2022.
- [25] B. Kitchenham, "Procedures for performing systematic reviews," Keele University, Keele, UK, Tech. Rep. 33, 2004.
- [26] A. Booth, A. Sutton, M. Clowes, and M. M.-S. James, *Systematic approaches to a successful literature review*. Sage Publications, 2021.
- [27] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS quarterly*, pp. xiii–xxiii, 2002.
- [28] V. Braun and V. Clarke, "Thematic analysis," in *APA handbook of research methods in psychology, Vol. 2. Research designs: Quantitative, qualitative, neuropsychological, and biological*, H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, and K. J. Sher, Eds. American Psychological Association, 2012, pp. 57–71.
- [29] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.
- [30] J. Corbin and A. Strauss, *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage Publications, 2014.
- [31] Y. S. Lincoln and E. G. Guba, *Naturalistic inquiry*. Sage Publications, 1985.
- [32] G. A. Bowen, "Naturalistic inquiry and the saturation concept: a research note," *Qualitative research*, vol. 8, no. 1, pp. 137–152, 2008.
- [33] S. Magnet. (2024) The data engineer job market in 2024 [research on 1,000 job postings]. Accessed: 2024-01-06. [Online]. Available: <https://365datascience.com/career-advice/data-engineer-job-market/>
- [34] O. Yarygina, "Learning analytics of cs0 students programming errors: The case of data science minor," in *Proceedings of the 23rd International Conference on Academic Mindtrek*, 2020, pp. 149–152.
- [35] M. N. C. Vee, B. Meyer, and K. L. Mannock, "Empirical study of novice errors and error paths in objectoriented programming," in *Proceedings of the 7th Annual HEA-ICS Conference*. Citeseer, 2006.
- [36] M. Kaczorowska, "Analysis of typical programming mistakes made by first and second year it students," *Journal of Computer Sciences Institute*, vol. 15, 2020.
- [37] A. Singh, A. Fariha, C. Brooks, G. Soares, A. Z. Henley, A. Tiwari, H. Choi, and S. Gulwani, "Investigating student mistakes in introductory data science programming," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 1258–1264.
- [38] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proceedings of the 41st ACM technical symposium on Computer science education*, 2010, pp. 107–111.
- [39] J. C. Spohrer and E. Soloway, "Novice mistakes: Are the folk wisdoms correct?" *Communications of the ACM*, vol. 29, no. 7, pp. 624–632, 1986.
- [40] Y. S. Lincoln and E. G. Guba, "Establishing dependability and confirmability in naturalistic inquiry through an audit." 1982.

## APPENDIX

The search process led to the following list of articles used in the literature review:

- Singh et al., "Investigating Student Mistakes in Introductory Data Science Programming" [37]
- Brown and Altadmri, "Novice Java programming mistakes: Large-scale data vs. educator beliefs" [9]
- Brown and Altadmri, "Investigating novice programming mistakes: Educator beliefs vs. student data" [10]
- Jegede et al., "Analysis of Syntactic Errors of Novice Python Programmers in a Nigeria University" [13]
- Kaczorowska, "Analysis of typical programming mistakes made by first and second year IT students" [36]
- Smith and Rixner, "The error landscape: Characterizing the mistakes of novice programmers" [11]
- Altadmri and Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data" [12]
- Hristova et al., "Identifying and correcting Java programming errors for introductory computer science students" [14]
- Júnior et al., "Analyzing the Impact of Programming Mistakes on Students' Programming Abilities" [15]
- Denny et al., "All syntax errors are not equal" [16]
- Ettles et al., "Common logic errors made by novice programmers" [17]
- McCall and Kölling, "Meaningful categorisation of novice programmer errors" [18]

- Albrecht and Grabowski, "Sometimes it's just sloppiness-studying students' programming errors and misconceptions" [19]
- Kaczmarczyk et al., "Identifying student misconceptions of programming" [38]
- Yarygina, "Learning analytics of CSO students programming errors: The case of data science minor" [34]
- Ahmadzadeh et al., "An analysis of patterns of debugging among novice computer science students" [20]
- Kiran and Moudgalya, "Evaluation of programming competency using student error patterns" [21]
- Jackson et al., "Identifying top Java errors for novice programmers" [22]
- McCall and Kölling, "A new look at novice programmer errors" [23]
- Miedema et al., "Identifying SQL misconceptions of novices: Findings from a think-aloud study" [24]
- Vee et al., "Empirical study of novice errors and error paths in object-oriented programming" [35]

This part of the appendix provides the mapping between each identified programming mistake type and the corresponding literature sources where these mistakes were observed and analyzed.

- **Domain-Specific Mistakes (1)**
  - (1.1) Dataset Misunderstandings [37]
  - (1.2) Faulty Data Analysis Logic [34], [37]
  - (1.3) Incorrect Data Handling [34], [37]
  - (1.4) Misunderstanding Data Types [34], [37]
- **Strategic Errors (2)**
  - (2.1) Suboptimal Coding [19], [35], [37]
  - (2.2) Wrong Algorithm [17], [19], [35]–[37]
- **Misinterpretation Errors (3)**
  - (3.1) Incorrect Programming Assumptions [10], [12], [14], [16], [17], [19], [20], [22], [24], [36]
  - (3.2) Task Misunderstanding [17], [19], [37]
- **Environment and Language Misconceptions (4)**
  - (4.1) Incorrect Language Understanding [9]–[14], [17], [19]–[24], [34]–[38]
  - (4.2) Library Misuse [11], [22], [36], [37]
  - (4.3) Path and I/O Errors [19], [21], [34], [37]
- **Logical Errors (5)**
  - (5.1) Incorrect Loop Conditions [19], [38]
  - (5.2) Faulty Conditional Logic [17], [19], [36]
  - (5.3) Off-by-One / Out-of-Bounds Errors [11], [17], [19], [20], [36], [38]
- **Semantic Errors (6)**
  - (6.1) Incorrect Function Usage [9], [10], [14], [20]–[24], [34]–[37]
  - (6.2) Type Mismatches [9]–[12], [14], [16]–[23], [35], [36]
  - (6.3) Variable Scope Issues [19], [21], [24], [36], [37]
- **Syntax Errors (7)**
  - (7.1) Incorrect Loop&if-else Syntax [10], [12], [14], [22], [36]
  - (7.2) Incorrect Operators [9], [10], [12], [14], [17], [19], [21], [22], [24], [35]–[37]
  - (7.3) Invalid Keywords Usage [10], [12], [14]
  - (7.4) Missing Semicolons [16], [18], [19], [21]–[23], [36]
  - (7.5) Unbalanced Delimiters [9], [10], [12]–[14], [18], [19], [22]–[24], [36]
  - (7.6) Variable not declared or initialized [13], [17]–[20], [22], [23], [36], [37]
- **Sloppiness Errors (8)**
  - (8.1) Typographical Errors [18], [19], [22]–[24], [34], [35]
- **Memory Management Mistakes (9)**
  - (9.1) Memory Language Misconceptions [11], [36], [38]
  - (9.2) Memory Leaks [11], [36]