

A Multi-Institutional Study of Programming Log Data Features that Predict Student Success

Samiha Marwan¹, Thomas W. Price¹

¹North Carolina State University, Raleigh, NC, USA

Abstract

Programming log data has been widely used to analyze student behavior and predict student learning outcomes, such as early identification of at-risk students. Prior work extracts features from programming log data (e.g. time spent on the task, or the number of student attempts on a given task) of datasets within a specific context in various ways, and assumed to be meaningful predictors of students' outcomes. However, it is unclear whether these same features would still be robustly predictive in other classroom contexts and programming languages. In this paper, we run a set of analytics across four datasets, some with multiple semesters across different classroom contexts and programming languages, to investigate how consistently features from prior literature can predict students' performance using early-course data (e.g. ~ the first 30% of the course). We find some features have the potential to be reliable predictors of student performance across datasets. We also identify features that are consistently predictive within semesters of the same dataset, but their predictive strength varies across datasets. We discuss how different classroom contexts and dataset features can affect the prediction of student performance. We also present how similar features, from different datasets, are calculated to help researchers standardize common calculations, and facilitate replication and secondary data analysis. Finally, we discuss the implications of these findings for future research on programming log data.

Keywords

Programming log data, data features, early performance prediction, correlation analysis,

1. Introduction

Programming log data is increasingly collected and used in Computing Education Research (CER) because it can capture student dynamic behaviors over time, such as the number of attempts a student makes on an exercise, or the time spent on a particular task. These are *features* that researchers rely on as a proxy for behaviors that instructors care about, such as student performance, learning, or struggle. Accordingly, researchers often evaluate the ability of log data features to predict outcomes that they care about. For example, prior work used the number of submissions a student takes to solve a problem, or the time they take to do so, as a measure of student performance, and found these features predictive of students' course grades or final exam scores [1, 2, 3, 4]. Inspired by this work, a large body of research has developed automated models using programming log data to predict students' final performance *early* in the course, and therefore, facilitate their success [2, 5, 1]. However, this line of research has two main limitations.

First, prior work almost exclusively evaluates or validates the predictive power of programming features within a single course *context* (i.e., one programming environment, one assessment design, one course structure), which limits the generalizability of their findings. In other words, it is unknown whether features predictive of student performance in one context will have the same predictive power in another context, making these predictive models rarely used in practice. Second, data features are defined and computed differently across papers, depending on the classroom context and the structure of log data, making reproducibility a challenge. Computing even simple features (e.g., How much time did a student spend programming?) requires making many *design choices* (e.g., Should large idle gaps be ignored? If so, how large? What about problems a student did not attempt?), and therefore, it is unclear how such analysis could be applied consistently across multiple datasets with various contexts.

CSEDM'26: 10th Educational Data Mining in Computer Science Education (CSEDM) Workshop, June 28, 2026, Seoul, South Korea

✉ samarwan@ncsu.edu (S. Marwan); twprice@ncsu.edu (T. W. Price)

🌐 <https://www.samihamarwan.com/> (S. Marwan); <https://csc.ncsu.edu/people/twprice/> (T. W. Price)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

To tackle this gap, **the first goal** of this paper is to rigorously evaluate the predictive validity of commonly used features, and to understand how their predictive power varies across course contexts. In particular, we want to investigate the usefulness of these features *early* in the course, when identifying at-risk students is most useful and pedagogically valuable. Moreover, some of these features are easy to calculate from a gradebook (e.g., score, number of attempts, how early a student submitted), and some require detailed logging (e.g. calculating student’s idle time on a given task). Therefore, our **second goal** is to help instructors know whether simple features are more or less powerful than ones that require additional logging or analysis.

In this paper, we ask the following research questions: (1) Which features, derived from students’ programming log data early in the course, are most predictive of students’ final course outcomes? (2) How consistently do these features perform across datasets? To answer these questions, in this work, we analyze four datasets, some with several semesters of data, in various educational contexts to investigate which data features can serve as early predictors of students’ exam performance across contexts. Our analysis revealed the potential of *Score* and *Time* features as reliable predictors of student performance *across* datasets. We also found that while some features consistently predicted student performance across different *semesters* within the same dataset, these features did not generalize *across datasets* with different contexts and student populations. This finding adds nuance to prior studies that claim certain log data features are more predictive over others, which suggests that such conclusions may not be applicable on other datasets.

Overall, this work contributes the following:

1. A multi-context analysis that investigates which features extracted from log data can predict student performance early in the semester. This analysis also evaluates the generalizability and robustness of these features across datasets.
2. Set of recommendations to help researchers standardize the computation of common programming log data features across diverse datasets. This contribution can also facilitate the *replication* of prior studies, support secondary data analysis, and promote methodological consistency in CER.

2. Literature Review

Student Performance Prediction Using Machine Learning (ML): Predicting students’ performance in programming using ML models is one of the oldest and well-studied problems in the CER community [3, 6, 7, 1, 2]. For example, Deep Learning (DL) models, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) on sequences of student submissions or time-ordered events can achieve high predictive accuracy [8, 9, 10]. However, only a limited number of studies incorporate explainable ML techniques (e.g., SHAP) to interpret the model outputs [2]. This limitation also makes these models difficult to communicate to instructors, limiting its practical use in classrooms. In addition, DL models can also handle non-tabular data by directly analyzing program code or time series data [11, 8, 9]. However, in this work, we focus on evaluating the utility and predictive power of explicit, named features that are either logged in or could be extracted from log data, such as the *StartTime*, which we will discuss in detail below.

Another research direction uses static variables to predict student outcomes, such as prior programming experience [3], or prior academic experience [12, 3]. However, many studies found weak correlations between these variables and student success [3]. In addition, it is inconvenient to use because it adds an additional burden to instructors or students to process lengthy surveys [3]. Other work uses **dynamic** features captured from programming log data (such as number of errors, number of attempts, or time spent) [3, 13, 2, 14]. In a review comparing static features and dynamic features in predicting student performance, Watson et al. found strong and significant correlations between student programming behavior and performance; however, they found no static variable, except for self-efficacy, that strongly correlate with students’ performance [3]. Many studies focus on using programming behavior features (i.e. dynamic features) to *identify* at-risk students *earlier* in the course,

so that instructors can provide timely interventions and facilitate their success [1, 15, 16, 17]. This is the focus of this work, and below we review prior work in this line of research.

Dynamic Data Features: Prior work explored four main types of data features extracted from programming log data, where each can include many features, to predict early student performance. First, *Score features*, such as the number of student submissions/attempts per problem, or maximum scores on problems, which have been shown to be a good predictor of student success [2, 18]. For example, Hoq et al. found that the average score of student assignment submissions and the number of attempted problems can predict students’ final exam scores early in the course [2]. Second, *Error features*, such as the Error Quotient (EQ), or Repeated Error Density (RED). These data features are fundamental in understanding student struggles in programming; however, they have varying accuracy in predicting student performance [4, 19, 5, 13]. Third, *Time features*, such as student total time spent on a problem, student idle time, or time until a student has a correct attempt [2, 20]. These time features offer valuable insights into students’ engagement, study habits, and course performance [3, 20, 21]. For example, Lee et al. found that students who submit their homework late are more likely to poorly perform in their exams [22]. Fourth, *Code features*, such as number of comments or lines of student code. Code features are used to analyze the quality of student-generated code and explore student struggle, and problem-solving approaches [23, 24, 2]. For example, Kerschbaumer found that the number of comments and the complexity of the source code has a positive influence on students’ success, whereas Hoq et al. found that low EditDistance in early assignments predicts poor performance in student final exams in a CS1 Java course [2].

Application of Data Predictors and Generalizability: While researchers have put a lot of effort into exploring how these data metrics can predict student performance, this prior work is often limited to comparing a single *type* of feature (e.g., *Error features* [13] or *Time features* [20]), making it unclear how these various approaches compare. To our knowledge, only a few studies evaluated a small subset of these metrics. For example, Watson et al. found that Watwin score and the percentage of lab time spent on resolving errors are strongly correlated with student course performance [3]. Other prior work suggested the use of multiple data sources to gain an accurate picture of students’ learning [20, 7, 25].

In addition, prior work found that some log data features can predict student performance when applied to a single course context. However, these data-driven approaches may lack generalizability across different programming courses due to variations in course structures and intermediate assessments. For example, Carter et al. used the *Error Quotient* and *Watwin Score* to predict student performance and found that they accounted for 3% and 12% of the variance of students’ final grades, while in a prior study they accounted for 18% and 36%, which is a significant change [4]. Carter’s et al. suggested these inconsistencies might be due to the difference in programming languages and the assessment weights in both classrooms. Another reason for inconsistent findings is that data features are often defined and computed *differently* across studies [4, 13], depending on classroom context and log data structure, which makes reproducibility a challenge. In this paper, we address these gaps by evaluating two types of data features across several datasets, to predict student course performance. We also provide standardized definitions, and recommendations on how to calculate data features to promote consistency in future CER research.

3. Methods

3.1. Datasets

In this work we analyzed four datasets, consisting of 8 separate semesters of data, as shown in Table 1. All datasets are from introductory CS1 courses and capture students practicing programming in Java or Python in a learning environment as part of a formal university course. However, these datasets have different institutional settings (e.g. R1 and R2 universities, and the US Air Force Academy), as well as different course context (e.g. course structure, assessments, and grading policy) which we describe below. All datasets capture changes to the students’ code over time, though they vary in granularity (submission-level, edit-level, keystroke-level), and they are available for research use, publicly, or by

request. Importantly, all datasets include a summative outcome measure for students: either a final course grade, final exam grade, or exam programming problems, allowing us to evaluate the predictive performance of programming log features for these outcome measures. Although these outcome measures vary across datasets, correlation analyses revealed they are highly correlated (e.g. final exam grade is correlated with final course grade). Therefore, we hypothesized that these different outcome measures remain valid for addressing our research questions. The datasets used the *ProgSnap2* format (one, FalconCode dataset, was converted to this format) [13], but they had differences in which events and attributes were captured. Table 1 summarizes datasets properties, and more details can be found in the cited sources citations.

CodeWorkout (CWO) [26, 27]: This dataset is collected from a CS1 Java course at a large, R1 public university in the United States. It contains data from two semesters where students worked on small programming exercises and were allowed to submit assignments at any time. This dataset lacked defined deadlines, and students were allowed to submit assignments in any order and at any time. This dataset also includes students’ final exam scores.

FalconCode (Fal) [28]: This dataset is collected from three semesters in a CS1 Python course for undergraduate students at the United States Air Force Academy. Authors of this data log students’ code submissions as they test their programs and submit them for grading. This dataset does not include student final course grades or exam scores; however, students complete several programming problems *labeled as exams* throughout the semester. To calculate a final exam score for each student, we summed up the highest achieved score for each exam-labeled programming question.

CodeBench (CB) [29, 30]: This dataset was collected from a CS1 Python course. In this course students solve programming problems using CodeBench [29], which is an Online Judge platform that logs fine-grained data and is developed by the Institute of Computing of the Federal University of Amazonas, Brazil. CodeBench logs all actions performed by students (such as keystrokes and code submissions). This data also includes student scores on exam coding exercises.

Edwards Keystroke Datasets (Ed) [31, 32]. This keystroke dataset was collected from research studies that took place in two semesters in a CS1 Python course at a mid-sized public university in the United States. This course was open for major and non-major students. In this dataset, there were multiple submissions per student; however, only the last submission was graded and therefore we only use the final submission in our analyses. This dataset also includes students’ final course grades.

Table 1
Datasets Description

Programming Context	Grade Predicted	Language	Submissions	Errors	Semester	Students Analyzed	Problems: Early/Total
CodeWorkout [26, 27] - Public U.S. R1	Final Exam	Java	Manually	Compile	S19	341	20/50
					F19	494	20/60
FalconCode [28] - Small U.S. Air Force Academy	Exam Problems	Python	Manually	None	S22	482	64/120
					F21	621	44/130
					S21	444	40/143
CodeBench [29, 30] - Public Brazilian University	Final Grade	Python	Manually	Syntax + Runtime	F24	212	88/280
Edwards [31] - Public U.S. R2	Final Grade	Python	Final Submission	None	S19	206	2/5
					F21	44	2/8

3.2. Data Features

Derived from literature, we calculated *Score* features and *Time* features across all datasets. There are three other types of features prevalent in the literature: compiler error features such as *Error Quotient* [13], keystroke features which analyze very fine-grained keystroke-level actions [31, 20], and code features such as *Cyclomatic Complexity* [24]. We excluded compiler error metrics due to space

constraints and because they are not supported across all datasets. However, preliminary analyses on supported datasets yielded weak correlations with student performance, aligning with prior work [13, 33]. We also excluded keystroke features because half of the datasets lacked keystroke-level logging. Finally, we left analysis of code features for future work because their language-specific nature requires a separate, comparative evaluation.

Each dataset has a different context (i.e. classroom structure and programming language), so calculating features from the log data requires making important implementation decisions. We explain each feature followed by the design choices we made (if applicable) for calculating each. As described in more detail in Section 3.3, we calculated each of these features for each student-problem pair (i.e., one student’s work on one problem), and then aggregated the results across problems to get a single value per feature per student, similar to prior work [2, 13, 18].

3.2.1. Score Features

Score features are calculated from the scores students receive on programming tasks (usually from test cases) when they *submit* their work for feedback. Our datasets varied in how a “submission for feedback” was detected, as shown in the “Submissions” column of Table 1. In the **CWO**, **CB** and **Fal** contexts, students manually submitted their work to receive unit test feedback as many times as they wanted. In the **Ed** context, students submitted their code only when finished, which was graded by teaching staff, and we used those grades as their scores¹.

We calculated the following *Score* features for each student’s work on each problem: (1) *Attempts*: The number of times the student submitted their code for feedback on a given problem, regardless of their score. Prior work used this feature as a measure of struggle and/or effort on a problem, as the student requires more feedback to succeed [34, 30, 20]. (2) *MaxScore*: The highest score (0-1) achieved across all submissions for a given problem, and is generally equivalent to the student’s grade on that problem. These features represent information that instructors already have access to, making them commonly used features in predicting students’ performance [15, 20, 2, 22, 30, 34], and therefore, they serve as a good baseline against which to compare other features.

We also calculated the following three features across all problems: (3) *FirstCorrect*: The proportion of all *attempted* problems that the student completed correctly (i.e. passed all test cases) on their *first* attempt, which is a common practice in student modeling to give the most accurate measure of student performance [11, 35]. (4) *EverCorrect*: The proportion of all *attempted* problems that the student ever completed correctly. In prior work this feature is called: *CorrectSubmissions*, or *Correctness* [2, 30, 17]. While this feature indicates students’ willingness to complete their work, which is a behavior that can reflect motivation [36, 37], other research has found that it is not a strong predictor of student performance [2, 17]. (5) *ProblemsStarted*: The number of problems that a student started (regardless of their score). Since all other features are *averaged* across only the problems that a student started, this feature captures how many problems the student started, which is a common high-level feature used in predictive models [22, 2].

3.2.2. Time Features

Time features capture student engagement with a problem, and have been used frequently in prior work [20]. All datasets provided timestamps, though the granularity of logging varied (e.g., one log per submission vs one log per edit). Therefore, we set different time thresholds based on the dataset time distribution. We explain this in detail below. As before, all the five time features below were calculated for each student’s work on each problem. (1) *StartTime*: The time of the first log entry for the student’s work on a given problem. (2) *FirstCorrectTime*: The time of the first fully correct submission (i.e. passing all test cases) for a given problem. It is undefined if the student submission never passed all test cases. Prior work used these features to capture procrastination behaviors and

¹In the **Ed** dataset, it is worth noting that we only analyze the final graded attempt, and therefore, features related to the number of attempts carry no significance in the analyses because there is exactly one submission per problem.

examine the effect of starting early on a task on student performance [38, 39, 17]. For example, Edwards et al. found that students who achieved high scores tended to start their assignments earlier [38]. (3) *ActiveTime*: The time the student spent *actively* working on this problem, ignoring gaps that may indicate disengagement or breaks. This is a common feature used in predictive models of student performance [20, 39, 18, 2]. (4) *PassiveTime*: The amount of time the student spent in shorter gaps in programming activity (with no log data) on this problem. While it has been previously used as a measure of disengagement [21, 40], it could also indicate that a student is taking a long time to think, tracing their code, or reading instructions. (5) *TotalTime*: The sum of *ActiveTime* and *PassiveTime*, since some papers do not distinguish between these two (e.g. [20]).

To calculate *ActiveTime* and *PassiveTime*, we require a threshold of inactivity that separates active work from a passive time, taking into account the granularity of the log data. For example, a 5-minute gap between submissions is normal, but a 5-minute gap between keystrokes suggests some break in activity. We should also exclude large breaks in work (e.g., a full day) from both of these measures. To determine these thresholds, we looked at both prior work and the distribution of time gaps for each dataset. For keystroke-level datasets (**Ed**, **CB**), no activity for 30 seconds (98th+ percentile) was considered passive time, and no activity for 3 minutes (99th+ percentile) was considered a break (similar to prior work [20]). For submission-level datasets, 5 min of inactivity (90th+ percentile) was considered passive time (consistent with prior work [41]), and 30 min of inactivity (95th+ percentile) was considered a break.

3.3. Data Analysis

As detailed earlier, prior work uses log data features as a proxy for meaningful student behaviors (e.g., procrastination, efficiency). While we cannot test these hypotheses directly, we can test them *indirectly* by evaluating how well these features predict final student course outcomes, which are related to *meaningful* student behaviors. Below we describe how we extracted and analyzed features from diverse datasets from different contexts.

Early Threshold: Specifically, we focus on using *early* log data in the course to predict student final exam scores or course grades. We focus on early data because this is the most practically useful time to identify struggling students. To do so, for each dataset, we manually identified a time cutoff for “early” log data that would be used for prediction. If assignment due dates were available or obvious from student activity, we identified the first ~30% of assignments and used the due date of the last assignment as the cutoff. Otherwise, we plotted the distribution of logs for each assignment and identified the most prominent gap in activity between assignments between 20% and 40% of the way through the course. We removed activity for any problem that contributes less than 50% of log activity for that problem occurred inside the time window.

Applying and Aggregating Metrics: In our analysis, we first grouped all data by student and problem, and then calculated each feature for a given student’s behavior within each problem (e.g., calculating the *MaxScore* one student received on one problem). We then calculated the z-score to normalize each feature within each problem, such that the feature value represents the student’s behavior *relative* to each of their peers on that problem. A z-score of 0 on a problem means the student’s performance (e.g. *MaxScore*) was average, and 1 indicates 1 standard deviation above average. This normalization allows easier comparison and aggregation across problems because z-scores have a similar scale for every problem, while raw values may have different scales on each problem (e.g., one problem took 5 minutes on average, while another took 30 minutes). We then averaged these z-scores for each student across all problems to give a single feature value per student, which can then be correlated with their grade. Specifically, the z-score normalization was applied across students at the problem level, rather than across problems. This average *ignores* problems a student did not attempt, which can sometimes change the meaning of the feature (e.g., a student can do only 10% of problems perfectly and achieve an average *MaxScore* of 1). We also computed sum-aggregated features but the results were quite similar, so we report only mean-aggregated features. The only exception is *ProblemsStarted*, which is the sum of all problems attempted.

Correlation Analysis: To measure how data features can predict student performance we used Spearman correlation, which is a rank-based measure of correlation between -1 and 1. As in prior work [20, 19], we used the Spearman correlation since the distributions of data features and students’ grades contain extreme values, which have an outsized effect on Pearson correlation. Rank correlation is still effective at detecting linear relationships, but it is less sensitive to these outliers, and can also capture non-linear, monotonic relationships between variables. One limitation of correlation analysis is that it does not consider the value of features in *combination*, which is explored in prior work using ML feature importance approaches (e.g. SHAP [42, 2]). We aim to explore these approaches in future work; however, our preliminary analysis suggests that only two of our datasets (**CB** and **Fal**) can produce high-performing predictive models, and feature importance approaches can be quite misleading with low-quality models. We therefore report only correlational analysis in this work.

4. Results

RQ1: Which features derived from students’ programming log data early in the course are most predictive of students’ course outcomes? Figure 1 shows the Spearman correlation between each data feature and students’ outcome measure (final grade, or final exam score) across the four datasets. Strong correlations have a large magnitude (positive or negative) and a darker color. We found that many of the strongest correlations come from *Score* metrics, such as MaxScore, which has significant correlations on all but one dataset. Importantly, this measure is essentially the student’s grade on the assignment and is already available to instructors without log data. *Time* metrics have some predictive power: students who start problems earlier (lower StartTime) consistently perform better in the course, which aligns with prior work on procrastination [43, 38]. **Overall, our results show that student scores on assignments are the most reliable predictive feature for student outcomes**, and maybe other features can add additional predictive power, which we plan to investigate in future work.

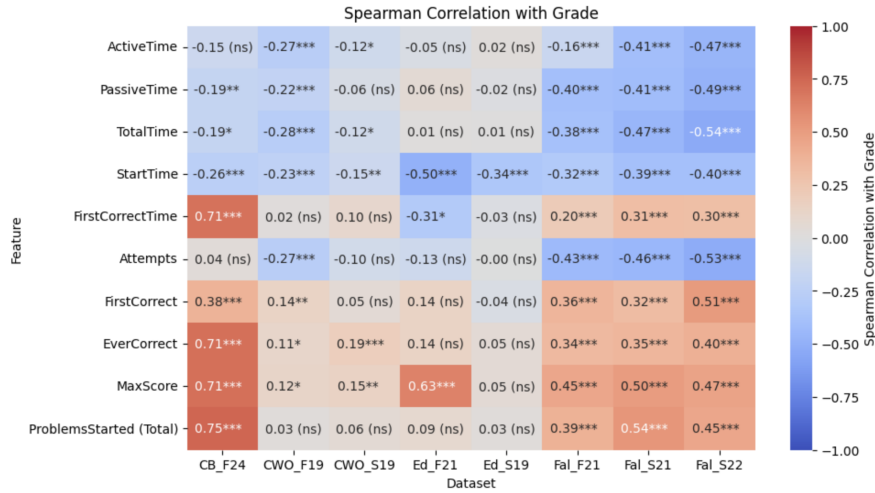


Figure 1: Spearman Correlations between each data feature and exam scores, across all datasets. * means significant value.

RQ2: How consistently do these features perform across datasets? Our results show clear inconsistencies in the predictive power of features across datasets. We did find that most features are consistently predictive *across semesters within* a given dataset and programming context (e.g., PassiveTime in **Fal**), but not across datasets. This suggests that while it may be possible to interpret the meaning of log features and use them for prediction within a *given programming context*, as has been done successfully in prior work [2, 7, 3], it is quite difficult to make *generalizable* claims about their meaning or importance. Two exceptions seem to be MaxScore, which our data suggest is a reliable, direct measure of performance, and the strong negative correlation of StartTime with outcomes, which has

a strong theoretical grounding in procrastination literature [43]. However, even then, the magnitude of correlation varies meaningfully across contexts. **Overall, our results show little consistency between features in predicting student performance across datasets, but some consistency within semesters of a given dataset.**

5. Discussion & Implications

Overall, we found that `MaxScore` and `StartTime` features have the most variation to predict performance across datasets. However, we found these features are highly predictive for some datasets (e.g. **Ed F21**), but not for others (e.g. **CWO**). A few factors likely affect this. First, **What data is included?** The **CWO** dataset only includes a subset of the programming assignments that students worked on, including short practice assignments, but not larger projects. These easier exercises, which nearly all students complete successfully, may not be as predictive of students' mastery of course learning objectives, leading to lower correlations with score features. By contrast, the **CB** and **Fal** datasets included exam programming problems as well as other practice problems, making early scores much more informative of students' course performance. Second, **What outcome measure is used?** The **CWO** datasets included only students' final exam grades, which we used as the outcome measure. These more formal exams may test related but different skills than what students are demonstrating in practice problems, and as a result these practice problems are less predictive. By contrast, final course grades (used in **CB** and **Ed**), which often include ample credit for programming practice, and programming exams are likely better predicted by other programming practice. Third, **Are there any additional interventions?** The **Ed S19** and **F21** datasets have quite different results, which is an exception to the general finding that features are consistently predictive across semesters of the same course. One possible explanation is that in the **F21** dataset students completed their assignments in an IDE that provided hints and simple code completion [31], which may have improved the practice exercises' ability to differentiate between high- and low-performing students.

Implications: Our work has a number of implications for research.

1. Our work strongly underscores the importance of evaluating predictive analytics on multiple datasets from diverse course contexts. This can highlight consistent results that are broadly adoptable, e.g. that the association between procrastination (starting an assignment late) and student outcomes is quite robust. It can also highlight inconsistent results that should be generalized cautiously, e.g. the total number of `ProblemsStarted` may be highly predictive of students' final scores, or hardly predictive at all, probably depending on the course context.
2. This work was only possible because of the availability of public datasets, highlighting the importance of data sharing. However, we also argue that including more details about course context with public datasets is deeply important to enable informed secondary analysis. Without more contextual information, we are forced to make inferences about what differences between datasets are most likely to explain the differences in our results. While we are deeply grateful for the documentation already provided by dataset authors, additional standardized documentation, such as the one proposed in prior work [44], would help inform this type of analysis.
3. We recommend caution in assigning *meaning* to specific log data features, as they are inherently dependent on the course context. For example, `ActiveTime` is sometimes interpreted in prior work as a measure of performance (i.e. less time means higher performance), and we see support for that interpretation in **Fal** datasets. However, spending more time on tasks can also indicate increased engagement with the task [21], or dozen other things, which is why its impact might not hold in other datasets (eg. **Ed**, **CWO**).
4. In the absence of strong results for other log data features, this work suggests the need for more research into how to best leverage programming log data. Perhaps we should pursue more advanced predictive features that outperform *Score* features, but our results suggest that early performance prediction may not be the most effective use-case for log data analysis.

5. Our results demonstrate the importance of accounting for dataset contexts, opening up new research directions to explore how these contextual variations impact the predictive power of programming log features. Specifically, future studies can evaluate how different contextual variables correlate with log-data features and change their utility across diverse student populations.
6. Finally, we suggest that even very simple features, derived from a course gradebook (e.g. average score on programming practice) can be an effective predictor of student outcomes. However, this raises questions about the efficacy and utility of programming log features for prediction, suggesting the need for more innovation in this area, and the advantage in keeping this simple.

Limitations This work has three main limitations. First, the datasets used have different outcome measures. While this is due to the different contexts of each dataset, these measures are still valid indicators of students' performance outcomes. Second, the limited available contextual information of datasets limited us to fully interpret the differences in the results. Finally, because we tried to standardize the calculation of all features across datasets, with varying contexts and structures, our implementations may not exactly match those used in prior studies, which limits the extent to which our findings can be generalized.

6. Conclusion & Future Work

This study defined and calculated *Score* metrics and *Time* metrics across four datasets with different contexts to explore their predictive power of identifying student performance in exams using early student programming data. Our results clarify how multi-context analysis can affect the effectiveness of data features in predicting student performance. We found that student scores are the more reliable feature to predict student performance; however, it is not consistent across all datasets. We also found some features that are consistently predictive across semesters within a given dataset. In addition, in this work we contribute suggestions to standardize computation of different log-data features to facilitate secondary data analysis and consistency of data feature calculations for the CER community. In the future, we plan to explore additional datasets and calculate Code features. We also aim to investigate the predictive power of multiple features together using machine learning feature importance approaches.

Acknowledgments

This material is based upon work supported by the National Science Foundation under grant #2238108 and #2213792.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Grammar and spelling check, and Paraphrase. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

- [1] C. Romero, S. Ventura, Guest editorial: Special issue on early prediction and supporting of learning performance, *IEEE Transactions on Learning Technologies* 12 (2019) 145–147.
- [2] M. Hoq, P. Brusilovsky, B. Akram, et al., Explaining explainability: Early performance prediction with student programming pattern profiling, *Journal of Educational Data Mining* 16 (2024) 115–148.
- [3] C. Watson, F. W. Li, J. L. Godwin, No tests required: comparing traditional and dynamic predictors of programming success, in: *Proceedings of the 45th ACM technical symposium on CS education*, 2014, pp. 469–474.

- [4] A. S. Carter, C. D. Hundhausen, O. Adesope, The normalized programming state model: Predicting student performance in computing courses based on programming behavior, in: Proceedings of the eleventh annual international conference on international computing education research, 2015, pp. 141–150.
- [5] C. Watson, F. W. Li, J. L. Godwin, Predicting performance in an introductory programming course by logging and analyzing student programming behavior, in: 2013 IEEE 13th international conference on advanced learning technologies, IEEE, 2013, pp. 319–323.
- [6] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, et al., Educational data mining and learning analytics in programming: Literature review and case studies, Proceedings of the 2015 ITiCSE on working group reports (2015) 41–63.
- [7] N. Gitinabard, Y. Xu, S. Heckman, T. Barnes, C. F. Lynch, How widely can prediction models be generalized? performance prediction in blended courses, IEEE Transactions on Learning Technologies 12 (2019) 184–197.
- [8] Y. Mao, One minute is enough: Early prediction of student success and event-level difficulty during novice programming tasks, in: In: Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019), 2019.
- [9] Y. Mao, Y. Shi, S. Marwan, T. W. Price, T. Barnes, M. Chi, Knowing both when and where: Temporal-astnn for early prediction of student success in novice programming tasks, in: In Proceedings of the 14th International Conference on Educational Data Mining (EDM) 2021, 2021.
- [10] Y.-H. V. Chiang, Y.-R. Lin, N.-S. Chen, Using deep learning models to predict student performance in introductory computer programming courses, in: 2022 International Conference on Advanced Learning Technologies (ICALT), IEEE, 2022, pp. 180–182.
- [11] Y. Shi, M. Chi, T. Barnes, T. Price, Code-dkt: A code-based knowledge tracing model for programming tasks, arXiv preprint arXiv:2206.03545 (2022).
- [12] D. Jeske, C. Stamov-Roßnagel, J. Backhaus, Learner characteristics predict performance and confidence in e-learning: An analysis of user behaviour and self-evaluation, Journal of Interactive Learning Research 25 (2014) 509–529.
- [13] T. W. Price, D. Hovemeyer, K. Rivers, G. Gao, et al., Progsnap2: A flexible format for programming process data, in: Proceedings of the 2020 ACM Conference on Innovation and Technology in CS Education, 2020, pp. 356–362.
- [14] E. Liu, I. Koprinska, K. Yacef, Early prediction of student performance in online programming courses, in: International Conference on Artificial Intelligence in Education, Springer, 2023, pp. 365–371.
- [15] J. Llanos, V. A. Bucheli, F. Restrepo-Calle, Early prediction of student performance in cs1 programming courses, PeerJ Computer Science 9 (2023) e1655.
- [16] F. D. Pereira, E. Oliveira, A. Cristea, D. Fernandes, L. Silva, G. Aguiar, A. Alamri, M. Alshehri, Early dropout prediction for programming courses supported by online judges, in: Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25–29, 2019, Proceedings, Part II 20, Springer, 2019, pp. 67–72.
- [17] F. D. Pereira, S. C. Fonseca, E. H. Oliveira, et al., Explaining individual and collective programming students’ behavior by interpreting a black-box predictive model, IEEE Access 9 (2021) 117097–117119.
- [18] G. Gao, A. Leon, A. Jetten, J. Turner, H. Almoubayyed, S. Fancsali, E. Brunskill, Predicting long-term student outcomes from short-term edtech log data, in: Proceedings of the 15th international learning analytics and knowledge conference, 2025, pp. 631–641.
- [19] M. C. Jadud, Methods and tools for exploring novice compilation behaviour, in: Proceedings of the second international workshop on Computing education research, 2006, pp. 73–84.
- [20] J. Leinonen, L. Leppänen, P. Ihantola, A. Hellas, Comparison of time metrics in programming, in: Proceedings of the 2017 acm conference on international computing education research, 2017, pp. 200–208.
- [21] S. Marwan, T. W. Price, M. Chi, T. Barnes, Immediate data-driven positive feedback increases engagement on programming homework for novices., in: CSEDM@ EDM, 2020.

- [22] N. T. S. Lee, O. Kurniawan, Predicting at-risk students for an introductory programming course: A pilot study, *ASCILITE Publications* (2019) 178–185.
- [23] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, C. Soubervielle-Montalvo, Source code metrics: A systematic mapping study, *Journal of Systems and Software* 128 (2017) 164–197.
- [24] D. Kerschbaumer, The code to success: Developing a student source code analysis tool to predict course performance and identify key concepts in programming education, 2023.
- [25] G. Kostopoulos, S. Karlos, S. Kotsiantis, Multiview learning for early prognosis of academic performance: A case study, *IEEE Transactions on Learning Technologies* 12 (2019) 212–224.
- [26] E. . Murali, Codeworkout dataset, 2017. URL: <https://sites.google.com/ncsu.edu/csedm-dc-2021/dataset?authuser=0>.
- [27] S. H. Edwards, K. P. Murali, Codeworkout: short programming exercises with built-in data collection, in: *Proceedings of the 2017 ACM conference on innovation and technology in CS education*, 2017, pp. 188–193.
- [28] A. de Freitas, J. Coffman, M. de Freitas, J. Wilson, T. Weingart, Falconcode: A multiyear dataset of python code samples from an introductory computer science course, in: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 2023, pp. 938–944.
- [29] A. Gunawardena, Codebench dataset, 2024. URL: <https://codebench.icomp.ufam.edu.br/dataset/>.
- [30] F. D. Pereira, E. H. Oliveira, D. B. Oliveira, et al., Using learning analytics in the amazonas: understanding students’ behaviour in introductory programming, *British journal of educational technology* 51 (2020) 955–972.
- [31] J. Edwards, K. Hart, R. Shrestha, et al., Review of csedm data and introduction of two public cs1 keystroke datasets, *Journal of Educational Data Mining* 15 (2023) 1–31.
- [32] J. Edwards, J. Ditton, D. Trninic, H. Swanson, S. Sullivan, C. Mano, Syntax exercises in cs1, in: *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 2020, pp. 216–226.
- [33] A. Petersen, J. Spacco, A. Vihavainen, An exploration of error quotient in multiple contexts, in: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, 2015, pp. 77–86.
- [34] A. Ahadi, R. Lister, A. Vihavainen, On the number of attempts students made on some online programming exercises during semester and their subsequent performance on final exam questions, in: *Proceedings of the 2016 ACM conference on innovation and technology in CS education*, 2016, pp. 218–223.
- [35] T. Gervet, K. Koedinger, J. Schneider, T. Mitchell, et al., When is deep learning the best approach to knowledge tracing?, *Journal of Educational Data Mining* 12 (2020) 31–54.
- [36] S. Marwan, B. Akram, T. Barnes, T. W. Price, Adaptive immediate feedback for block-based programming: Design and evaluation, *IEEE Transactions on Learning Technologies* 15 (2022) 406–420.
- [37] Z. Gao, B. Erickson, Y. Xu, C. Lynch, S. Heckman, T. Barnes, You asked, now what? modeling students’ help-seeking and coding actions from request to resolution., *Journal of Educational Data Mining* 14 (2022) 109–131.
- [38] S. H. Edwards, J. Snyder, M. A. Pérez-Quñones, A. Allevato, D. Kim, B. Tretola, Comparing effective and ineffective behaviors of student programmers, in: *Proceedings of the fifth international workshop on Computing education research workshop*, 2009, pp. 3–14.
- [39] J. Leinonen, F. E. V. Castro, A. Hellas, Time-on-task metrics for predicting performance, *ACM Inroads* 13 (2022) 42–49.
- [40] S. Marwan, G. Gao, S. Fisk, T. W. Price, T. Barnes, Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science, in: *Proceedings of the 2020 ACM conference on international computing education research*, 2020, pp. 194–203.
- [41] K. Hart, C. M. Warren, J. Edwards, Accurate estimation of time-on-task while programming, in: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 2023, pp. 708–714.
- [42] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, *Advances in*

neural information processing systems 30 (2017).

- [43] S. H. Cormack, L. A. Eagle, M. S. Davies, A large-scale test of the relationship between procrastination and performance using learning analytics, *Assessment & Evaluation in Higher Education* 45 (2020) 1046–1059.
- [44] S. Marwan, A. C. Bart, T. W. Price, A specification for cs education dataset documentation, in: *In the Educational Data Mining in Computer Science Education (CSEDM) 9th Workshop*, 2025.