Analysing Learner Strategies in Programming Using Clickstream Data

Daevesh Singh[a], Indrayani Nishane[b] and Ramkumar Rajendran[c]
IDP in Educational Technology, Indian Institute of Technology Bombay, Mumbai, India

Keywords: Digital Learning Environments, Programming Behaviour, Process Mining, PyGuru.

Abstract: Programming courses have high failure rates and to address this, it is crucial to better understand learning strategies associated with higher learning gains. Digital learning environments capture fine-grained data that offer valuable insights into learners' learning strategies. Although much research has been dedicated to analysing student programming behaviours in integrated development environments, it remains unclear how their reading and video-watching behaviours, which are used for knowledge acquisition, influence these programming behaviours. In this study, we aim to bridge this gap by analysing learners' actions in PyGuru, a learning environment for Python programming, using process mining techniques to capture their temporal learning behaviours. Our objective is to understand the behaviours associated with high and low-scoring learners. Study reveals that high-scoring learners execute codes more, indicating a correlation between execution actions and conceptual reinforcement and engaging in active video-watching behaviours, contributing to higher learning gains. Conversely, low-scoring learners tend to rely on trial and error techniques, neglecting content review after execution. Furthermore, despite the frequent use of the 'highlight' action, low-scoring learners fail to revisit highlighted content, suggesting a lack of comprehensive information processing. By uncovering such behaviours, we aim to shed light on effective strategies associated with higher performance, thereby helping instructors provide feedback to struggling learners.

1 INTRODUCTION

Programming is a valuable skill that is applicable to both academic and industry settings. This often generates strong interest in enrolling in computer science courses, especially those related to programming. The programming skill is considered difficult to learn and even more difficult to attain mastery. This leads to high dropout rates in programming courses (Bennedsen and Caspersen, 2019). In one of the big five open questions in computing education, Kim Bruce highlights how the dropout rates in introductory courses to programming can be minimized (Bruce, 2018). This has led to a lot of avenues for research in computing education. The use of learning analytics can help in improving learners' performance by identifying students in need and providing them support. However, the literature suggests that today's CS (Computer Science) classes still miss out on using diverse forms of Learning Analytics (Ihantola et al., 2015) to improve student performance somehow. This issue can be solved using digital learning environments that allow capturing of student data (Ihantola et al., 2015). These learning environments help us capture the interaction data of learners called click-stream data, which includes clicks on links, buttons, videos, or other elements of the digital learning environment and may also include the duration of time spent on each interaction. Click-stream data provides a detailed record of a student's activity within the learning environment and can be used to analyze patterns of behaviour, identify areas where learners may be struggling, and make data-driven decisions to provide support thereby improving their learning experience (Long and Siemens, 2014).

With the advent of educational data mining and learning analytics, it has become possible to use this fine-grained clickstream data to uncover patterns that would otherwise remain unobserved. Thereby enabling us to understand the learning process rather than just focusing on the learning outcome. Further, the need to use data mining techniques to examine fine-grained data is particularly crucial in the case of introductory programming courses due to the high failure and withdrawal rates in these courses(Yogev

[a] https://orcid.org/0000-0001-6610-3887
[b] https://orcid.org/0000-0002-1223-3528
[c] https://orcid.org/0000-0002-0411-1782

Singh, D., Nishane, I. and Rajendran, R.
Analysing Learner Strategies in Programming Using Clickstream Data.
DOI: 10.5220/0012636500003693
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024) - Volume 2*, pages 87-96
ISBN: 978-989-758-697-2; ISSN: 2184-5026
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

et al., 2018). Additionally, the amount of digital traces learners leave behind in programming is far more than in other domains, providing researchers a better opportunity to understand the learning process.

Existing research on understanding learners' behaviour in computer programming using learning analytics has focused on learners' interactions with the programming environment (Azcona et al., 2019), neglecting the strategies they employ while interacting with the course content. Therefore it is important to understand learners' interactions with the content to determine the reason behind differences in student performance or mastery. To address this lacuna created due to the focus on using data from isolated programming learning environments, we collected the clickstream data of learners from PyGuru, an online learning environment for Python programming. PyGuru allows learners to interact with the course content through reading content and videos and practising programming exercises. This allows us to uncover the relationships between actions performed by learners to access the content and programming behaviours to identify effective learning strategies that can improve learner performance. We have considered the temporal nature of the actions performed by the learners while analysing the click-stream data of the learners' interaction with PyGuru. We will refer to these sequences of actions the learners perform as learning strategies.

This motivated us to investigate the learner behaviour and their learning strategies while interacting with different learning environments elements like reading content, video content, and Interactive Development Environment (IDE). There are multiple mining techniques available to analyse the temporal data, such as Sequential Pattern mining (SPM), Differential Sequence Mining (DSM), Process Mining (PM), etc. SPM and DSM give us selective common and uncommon patterns from the log data. While PM gives us a visual depiction of the entire sequence of actions based on the complete log for all learners, enabling us to draw inferences about learner behaviours while they interact with different components of the learning environment.

In this study, we collected the click-stream data of learners to understand different learning strategies and their impact on performance. Our objective is to identify effective learning techniques that enhance learner performance by establishing connections between learners' actions to access content and their programming behaviour.

The results indicate that high-scoring learners significantly perform code execution, indicating a correlation between execution actions and conceptual reinforcement. Notably, high-scoring individuals engage in active video-watching behaviors, contributing to higher learning gains. Conversely, low-scoring learners tend to rely on trial and error techniques, neglecting content review after execution, potentially perpetuating misconceptions. Furthermore, despite frequent use of the 'highlight' action, low-scoring learners fail to revisit highlighted content, suggesting a lack of comprehensive information processing compared to their high-scoring counterparts.

The study's findings carry significant implications for educators, researchers, and system designers aiming to enhance educational outcomes in programming education. Since it was observed that certain actions like the execution of code and active-video watching behaviours are associated with higher learning gains, it is, therefore, crucial for educators to promote such actions. Also, researchers can leverage this information to explore targeted interventions, develop adaptive learning systems, and refine instructional approaches based on the observed correlations between actions and learning gains. These findings offer practical insights for educational stakeholders to tailor their approaches, interventions, and system designs, ultimately fostering more effective learning experiences and improving student outcomes in programming education.

The paper is organized as follows: In Section 2, we discuss related work, and in Section 3, we provide context for the data. Section 4 describes our methodology, including the prepossessing of click stream data, and we present our results in Section 5. Finally, we conclude Section 6.

2 BACKGROUND AND LITERATURE REVIEW

To analyze student strategies in programming, we will first present how learning analytics was used in computer programming, followed by different methods to analyze click-stream data, and finally present how process modelling was used to understand learners' learning process.

2.1 Learning Analytics in Computer Programming

The use of learning analytics to understand student behaviour in computer programming has been an area of research for several years, and researchers have collected and used different kinds of data to understand student behaviour. For instance, computer programming problem solving success (Guerra et al., 2014; Sosnovsky and Brusilovsky, 2015), help-seeking behaviour (Price et al., 2017), programming assignments progression (Piech et al., 2012), programming information seeking strategies (Lu and Hsiao, 2017), the use of hints (Rivers and Koedinger, 2017; Price et al., 2017), troubleshooting behaviours (Buffardi and Edwards, 2013), code snapshot process state (Carter et al., 2015), and generic Error Quotient measures (Carter et al., 2015). While extensive research has been done to understand learners' behaviour in computer programming using learning analytics, much of it has focused on learners' interactions with the programming environment, neglecting the strategies they employ while interacting with the course content. Understanding learners' interactions with the content is crucial for understanding the reasoning behind different learning levels.

To address this gap, in this paper, we explore process mining to obtain a comprehensive view of learners' different action sequences and better understand their learning behaviours.

2.2 Methods for Analysing Click-Stream Data

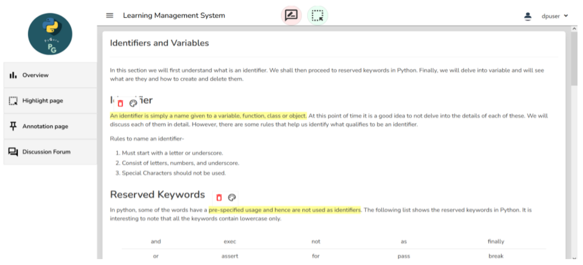
Understanding how learners behave in learning environments can be beneficial in identifying when scaffolding, such as personalized hints and positive encouragement, should be provided to create a more productive learning experience (Munshi et al., 2018). Analytics and mining schemes can be utilised to gain insight into learner strategies when using any digital learning environment (Saint et al., 2018). Analysing the temporal links between the actions of high and low performers can help identify the differences in their respective learning strategies (Rajendran et al., 2018). Several methods like DSM, PM are available for such analysis. DSM can identify less common but distinct behaviour patterns of different groups (Rajendran et al., 2018), while PM involves the analysis of temporally ordered action sequences performed by learners, which can be interpreted as their temporal problem-solving model (Günther and Van Der Aalst, 2007). PM can help visualize learner action sequences while interacting with the learning environment and provide insights into these sequences by visually depicting the interaction (Rajendran et al., 2018). Hence, in this work, we apply PM to understand learner strategies while interacting with PyGuru.

2.3 Analysing Click-Stream Data Using Process Mining

There are different techniques to analyse the click-stream data for insights about learners' learning strategies, as discussed in the previous subsection. Process mining (PM) is the technique that captures the temporal nature and the sequence of actions and showcases it pictorially. The temporal data consists of a sequence of events where each represents the learner's action in a digital learning environment. (Reimann et al., 2009; Winne and Nesbit, 2009). PM visually represents the temporal data in multiple forms such as Petri net or automata, etc (Günther and Van Der Aalst, 2007).

In previous work, (Sedrakyan et al., 2016) applied process mining techniques to interpret learners' cognitive learning processes by identifying novice modelling activities. The aim was to provide feedback which is process oriented rather than outcome based. Another interesting work by(Rajendran et al., 2018) visualized and explored temporal differences in learning behaviour sequences of high versus low-performing learners working on causal modelling tasks in Betty's Brain environment. Process models gave insights into different strategies deployed by the learners. Use of PM has also shown differences in learning interactions novices had when they interacted with a TELE to create a software conceptual designs (Nishane et al., 2021). This works highlighted that students who had higher learning interacted with design elements more frequently and created the designs compared to those who had lower learning. One study reported using PM to differentiate the behaviour of students based on their mindset, while they interacted with the learning environment (Nishane et al., 2023). PM showed the difference in the interaction pattern of learners with Fixed and Growth mindset as it is able to retain the sequential nature of actions along with the frequencies of the commonly occurring actions.

Learning Analytics (LA) holds a crucial role in the intersection of education and technology, objectives such as decoding learning strategies and comprehending how learners engage with programming environments, etc. needs more work for allowing researchers a deeper understanding of the learning computer programming. Employing methodologies like SPM, DSM, and PM, learner behaviour is systematically analysed in existing literature. Notably, PM serves as a valuable instrument for dissecting the temporal dimensions of learners' data, presenting a visual representation of their learning journey. In addressing existing research voids, our investigation centers



Identifiers and Variables

In this section we will first understand what is an identifier. We shall then proceed to reserved keywords in Python. Finally, we will delve into variable and will see what are they and how to create and delete them.

I ier

An identifier is simply a name given to a variable, function, class or object. At this point of time it is a good idea to not delve into the details of each of these. We will discuss each of them in detail. However, there are some rules that help us identify what qualifies to be an identifier.

Rules to name an identifier-

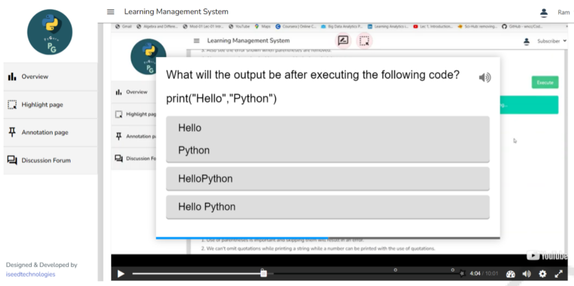
1. Must start with a letter or underscore.
2. Consist of letters, numbers, and underscore.
3. Special Characters should not be used.

Reserved Keywords

In python, some of the words have a pre-specified usage and hence are not used as identifiers. The following list shows the reserved keywords in Python. It is interesting to note that all the keywords contain lowercase only.

and	exec	not	as	finally
or	assert	for	pass	break

(a) Book Reader



What will the output be after executing the following code?

print("Hello","Python")

Hello
Python

HelloPython

Hello Python

1. Use of parentheses is important and skipping them will result in an error.
2. We can't omit quotations while printing a string while a number can be printed with the use of quotations.

(b) Video Player

Figure 1: (a) displays the book reader interface, featuring digital text. Users can highlight text with different colours and add comments with tags for the organization. (b) displays a screenshot of the interactive video-watching platform in PyGuru. The interface has basic video player controls (play, pause, seek, and speed enhancement) underneath it. It allows learners to respond to the embedded questions within the video.

on scrutinising learner conduct within programming learning settings, utilising PM as a key methodology. This approach is poised to unveil nuanced insights into the rationale behind distinct learning levels, thereby significantly contributing to the ongoing evolution of Learning Analytics.

3 LEARNING ENVIRONMENT: PyGuru

3.1 Description of Learning Environment

PyGuru [1] is a computer-based learning environment developed to teach and learn Python programming skills. PyGuru has four components: a book reader, video player, code editor, and discussion forum. This section describes each of these components.

1. **Book Reader.** PyGuru contains a book reader (shown in Fig 1 (a)) that allows the reader to highlight and annotate the text. Highlight in the digital context consists of selecting a text and colouring it. The annotating feature in the learning environment comprises selecting a text, commenting on that text, and providing a tag to that text.
2. **Video Player.** PyGuru has an interactive video watching platform (Fig. 1(b)). The learners can interact with the video using basic video player features like enhancing the speed of the video and performing other actions like play, pause and seek. Additionally, more advanced interactive features are embedded into the system, allowing the instructor to add questions within the video. The video automatically stops and waits for the learner's response.
3. **Code Editor.** Learning programming requires a code editor where learners can practise coding. PyGuru offers two kinds of code editors. The first kind of code editor (Fig 2(a)) is embedded into the book reader to facilitate learners to practice codes immediately after learning about the concept. This code editor has a coding window and an execute button. The second kind of code editor shown in (Fig 2(b)) is more advanced and used to assign programming questions to learners. It evaluates the learners' code against the test cases. The 'verify' button allows learners to check their program for errors and test cases before submitting. This code editor consists of four panels:
   (a) Instruction Panel -To provide details about the problem or algorithm like the inputs that the program will receive, the expected output, and an example to demonstrate the problem.
   (b) Input Panel - To provide the test cases for the problem, and the learners' code will be tested against these inputs.
   (c) Coding Panel - The coding panel is where the student is expected to write the code, and the instructor can also present some partial codes.
   (d) Output Panel - To display the output once the program is run. It will also provide information about the number of test cases passed and failed.

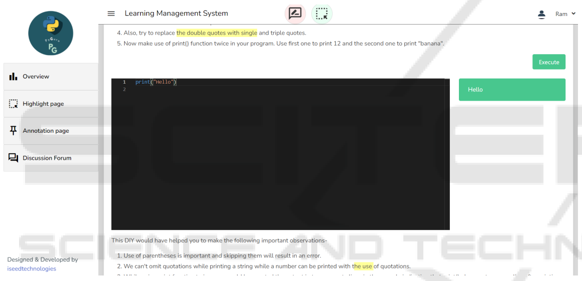
3.2 Log Data

In this subsection, we will first describe students' actions in PyGuru. The actions documented in Table 1 encompass diverse student actions within the learning environment.

[1]https://pyguru.personaltutoring.in/

Table 1: Description of the actions learner performs in the learning environment

Action	Description
Log_In	Logged into the learning environment
Log_Out	Logged out of the learning environment
Assessment	Assessment (Program IDE) page is viewed
Execute	Executing the code on embedded Code Editor
Verified	Executing the code on Advanced Code editor
Video_Player	Visiting the video Player
Reading	The course materials page is viewed
Video_Played	Video content is played
Highlight_view	The page where highlights are saved is viewed
Annotation_view	The page where annotations are saved is viewed
Paused	The video is paused
Highlighted	Highlighting features is used
Continue_Video	The video content is resumed after answering the in-video question
Seek	Navigation controls are used to seek specific sections within video
VQ_retry	In-video question is reattempted
view_VQ_Sol	The solution of the in-video questions is verified
See_Errors	The errors within the code notified by IDE is checked
Annotation	Annotated or tagged the content
VQ_opt_selected	The in-video question is attempted



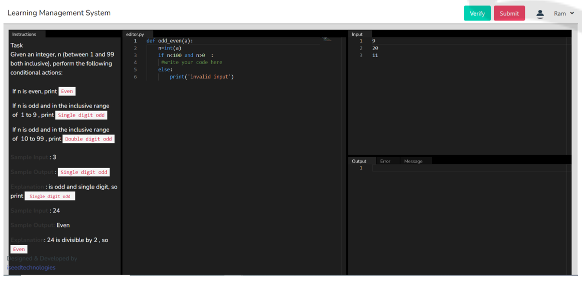
4. Also, try to replace the double quotes with single and triple quotes.
5. Now make use of print() function twice in your program. Use first one to print 12 and the second one to print "banana".

```
print("Hello")
```

This DIY would have helped you to make the following important observations-

1. Use of parentheses is important and skipping them will result in an error.
2. We can't omit quotations while printing a string while a number can be printed with the use of quotations.

(a) Embedded Code Editor



Task
Given an integer, n (between 1 and 99 both inclusive), perform the following conditional actions:

If n is even, print `Even`

If n is odd and in the inclusive range of 1 to 9 , print `Single digit odd`

If n is odd and in the inclusive range of 10 to 99 , print `Double digit odd`

Sample Input : 3

Sample Output : `Single digit odd`

Explanation : is odd and single digit, so print `Single digit odd`

Sample Input : 24

Sample Output : Even

Explanation : 24 is divisible by 2 , so `Even`

```
def odd_even(a):
    n=int(a)
    if n<100 and n>0  :
     #write your code here
    else:
        print('invalid input')
```

```
9
20
11
```

(b) Advanced code Editor

Figure 2: (a) Displays the code editor embedded in the book reader. (b) displays a screenshot of the second code editor used for formative assessment.

In book reader, the actions performed are reading, highlighting, annotation. Several actions focus on reading and text-related activities, such as 'Reading,' which denotes involvement in course materials, and 'Highlighted,' which involves highlighting features. 'Annotation' refers to the act of selecting a text and supplementing texts with user-generated comments or notes. These highlighted and annotated text can be collectively accessed on a separate page. These actions are referred to as 'Highlight_view' and 'Annotation_view'.

In the video player, the actions performed are played, paused, and seek. These actions pertain to initiating and managing multimedia content, with 'Video_Player' action defined as the act of visiting the video player and 'Video_Played' signifying engagement with educational video materials, while 'Paused' denotes the interruption of video playback. 'Seek' encompasses using navigation controls to locate specific sections within video content. 'Continue_Video' marks the resumption of video content after responding to in-video questions. Since the videos have in-video questions, the action 'VQ_response' corresponds to selecting one of the options in the in-video question. If the option selected is incorrect, learners can retry the question and the action 'VQ_retry' corresponds to this.

As mentioned earlier, PyGuru has two code editors. The 'Execute' action corresponds to executing the code in the embedded editor. The second kind of code editor is more advanced and the action 'Assessment' corresponds to using or accessing this code editor. The learners can also click the submit or verify button to check if their code is thriving on the given test cases. The learners can look at the 'See_Error' and 'Look_messages' tabs in the output panel for more information about the logical and syntax error.

Lastly, actions like ‘Log In’ and ‘Log Out’ encapsulate the initiation and termination of learning sessions by logging in and out of the learning environment.

4 METHODOLOGY

To investigate the difference in temporal actions between high and low-performing learners in computer programming, we apply PM models to the data collected from learners’ interaction data to seek the answer for the following research question.

How do learning strategies differ for high and low scoring learners for Python Programming?

We first describe the study design, data preprocessing and parameters set for process mining to answer this.

4.1 Participants

The study was conducted with 37 students who used PLE to learn Python programming in 2022. Out of which 18 were females. These students were aged between 18 to 19 years. These students were enrolled in a bachelor’s program at an Engineering Institute in an IT course. The researchers provided students with a demographic survey that involved questions like their name, age, gender, etc., at the beginning of the study. Further, they were also asked to report their prior experience with any other programming language. The data shows that most students were learning programming for the first time. Informed consent was obtained from the student, and the Institute Research Board (IRB) cleared the study. No monetary compensation was given to the students.

In addition, the students took a pre and post-test containing ten multiple-choice questions from Python basics (variables, operators, conditional statements, etc.). Students’ interaction with the system (clickstream) and the time stamp was captured (more details about log data are provided in the previous subsection). This study lasted four days, of which students’ interaction which system happened for two days and each day, the students interacted with the system for 3 hours.

4.2 Study Design

At the commencement of the study, a comprehensive procedural protocol was followed to ensure the informed participation of the students. Initially, on day one, the research objectives were explained, and the students were given consent forms, which they duly completed and signed, affirming their voluntary participation. Subsequently, a demographic survey was administered to gather pertinent background information. Following this, a pre-test was administered to assess the students’ baseline programming knowledge, comprising 10 multiple-choice questions that spanned various programming topics, including variables and conditional statements, among others. After the pre-test, students were introduced to the learning environment, which included a detailed demonstration of the system’s functionalities. They were informed of the diverse features available within the learning environment and were tasked to master Python programming using the system. The learning curriculum was structured into four distinct modules, each addressing specific programming concepts. The initial module encompassed topics such as print functions, input operations, and escape sequences, while the second module delved into identifiers, variables, fundamental data types, and operators. The third module concentrated on conditional statements, encompassing constructs such as ‘if,’ ‘elif’, and ‘else.’ The final module was dedicated to the comprehensive coverage of loops and control statements. Students were explicitly instructed to engage with the system for a duration of 3 hours per day over the next two days to facilitate their Python programming learning experience. On the fourth day, a post-test with a similar difficulty level as the pre-test was administered to gauge the extent of learning progression. Additionally, an engagement survey was distributed to the students for triangulation; however, it is important to note that the results are not included in the scope of this research endeavour. The study was done in a lab setup to control the conditions.

4.3 Data Preprocessing

Participants were categorized as high or low performers based on their performance in the post-test, with the median score being 4.5 out of a total of 10 marks. Those who scored 5.5 or higher (n=9) were grouped as “High”, while those who scored 3.5 or lower (n=16) were grouped as “Low”. Participants who scored between 3.5 and 5.5 were excluded to maintain a clear distinction between the two groups.

4.4 Process Mining

To visualize the temporal differences in the learning behaviour of high and low-scoring learners in PyGuru, we employ the fuzzy miner algorithm using the ProM tool (Günther and Van Der Aalst, 2007).

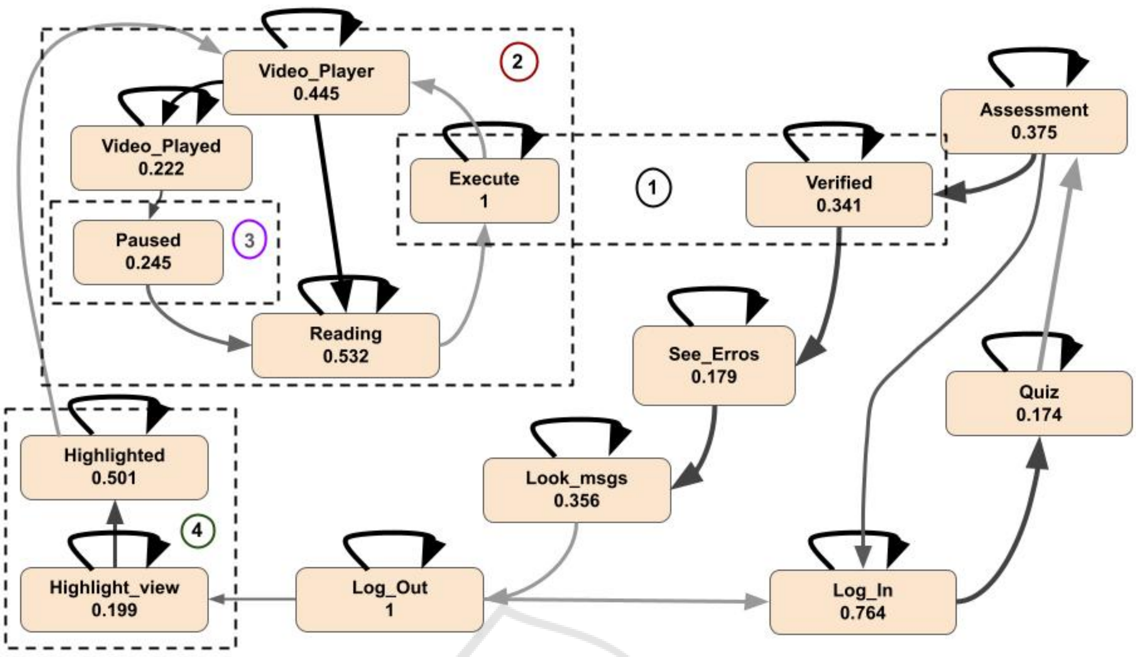


Figure 3: This figure represents the process model of high-scoring learners.

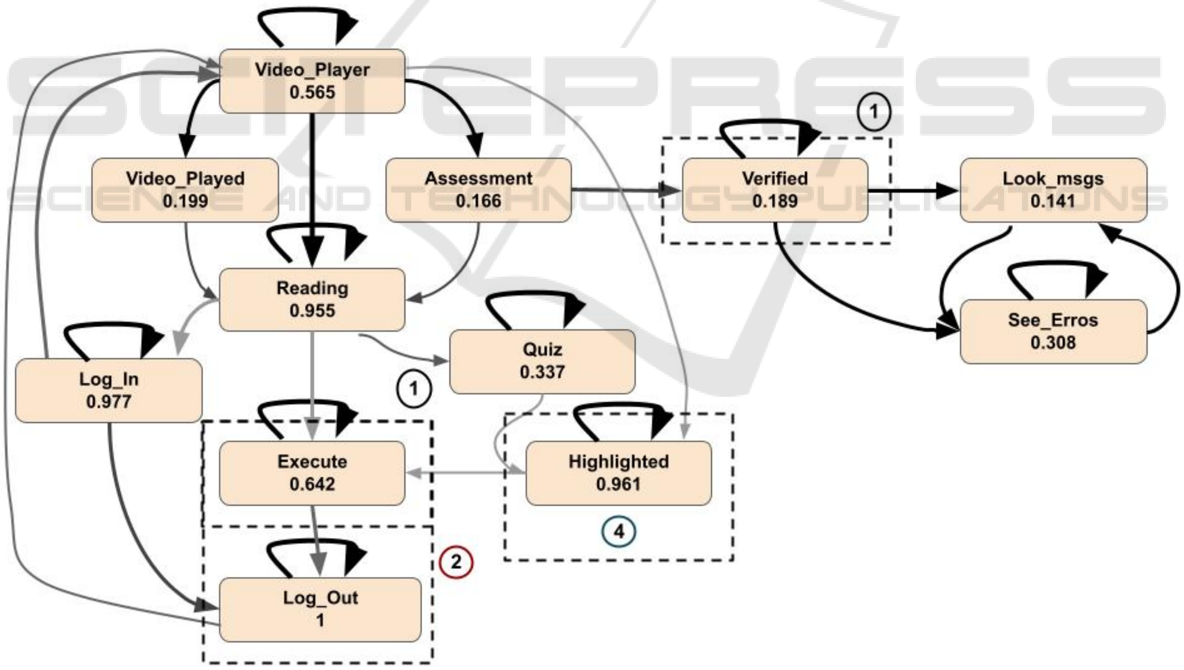


Figure 4: This figure represents the process model of low scoring learners.

ProM is an open-source process mining tool[2]. This tool provides process models of the temporal data, which are often very complex due to multiple events and transactions between them. The tool manipulates multiple parameters to change the degree of abstraction represented by certain key metrics to get a suitable abstraction. The next paragraph summarises the various parameters involved in obtaining the abstract view of the model. For more details, refer (Günther and Van Der Aalst, 2007)).

[2]www.promtools.org

The process model includes nodes that represent the events or actions, while the edges in the model represent the transitions between these events or actions. In all the PMs, the node represents the actions performed by learners, and the edge represents the transition from one activity to another. Each node has a significance value (between 0 and 1), and each edge has thickness indicating significance, while darkness depicts the correlation (Günther and Van Der Aalst, 2007). The abstraction of the model is done using these two key metrics, correlation and significance. Correlation measures the common occurrence of two events, i.e. if two events occur together more frequently will have a higher correlation. This metric is only used for nodes. On the other hand, significance is measured for both nodes and edges. It is defined as the relative relevance of a node or edge's occurrence with respect to all other occurrences. For instance, higher significance indicates that a particular node or edge occurs more frequently.

Using the above two metrics, the abstraction and simplification of the process model are done(Günther and Van Der Aalst, 2007), using the following three rules:

1. highly significant nodes are preserved as is;
2. less significant nodes that are highly correlated are aggregated and grouped into clusters; and
3. less significant nodes with low correlations to other nodes are dropped, thus creating more abstract forms of the model.

Now in the process mining tool, the abstraction is done by manipulating three parameters: node cutoff, edge cutoff, and utility ratio(ur). Node and edge cutoff is used to remove the nodes and edges having significance value(for nodes) and utility value (for edges) below the given threshold. The utility value (uv) of an edge is the convex combination of significance (s) and correlation value (cv) of an edge. In mathematical terms, it is defined as:

$$uv = ur * s + (1 - ur) * cv \tag{1}$$

To compare the process models of high and low learners, we retain all the nodes in the process model by keeping the node cutoff fixed at 0. We wanted the log conformance value to be above 80%, so we fixed the utility ratio as 0.5 and varied the edge cutoff value to achieve the desired log conformance. The significance metric is represented by each action node's numerical value (between 0 and 1). The thickness and darkness of the edges indicate the significance and correlation values associated with the edges, respectively.

5 RESULTS AND DISCUSSION

This section describes the result of the research question posed in the previous section. We first highlight the differences between the high and low-scoring learners using process models. We also present the discussion based on the results.

Figure 3 and Figure 4 show the process models of high and low-scoring groups, respectively. We find the following differences in the process models of these groups:

1. The significance of nodes like 'Verified' and 'Execute' is higher for high-scoring learners. These two actions correspond to executing the code in two different IDEs. They implement the concepts learned by reading the content or watching the course videos. A higher frequency of such actions might be one of the reasons for higher learning gains.
2. The high-scoring learners, after performing the 'Execute,' refers to the reading or video content, whereas the low-scoring learners opt for a trial and error technique and do not refer back to the content; therefore, after performing 'Execute,' a couple of times, they logout from the system. There are possibilities that the code executed might have produced an error or the output might be different from their expectations; in such cases, it is crucial to refer to the content to rectify the misconceptions. We see low-scoring learners do not refer back. As a result, they may continue to hold on to some of their misconceptions.
3. We also see the process model of high-scoring learners has a node 'pause', which is absent in low-scoring learners. The pause action corresponds to active video-watching behaviour (Dodson et al., 2018) and is linked to higher learning gains.
4. Another interesting thing to note is that the 'highlight' node, which corresponds to the action of highlighting the text, has a higher significance value in the process models of low scorers. The action highlight is usually linked to higher learning gains. This is since the act of deciding what to and what not to highlight itself denotes the deeper processing of the textual information as compared to simple reading (Yogev et al., 2018). However, the non-judicial use of highlighting indicates that the learners might not be processing all the information.
5. We also see that despite performing the highlight action relatively more times, the low-scoring learners never went to the highlight page to check

what they have highlighted, unlike high-scoring learners.

6 CONCLUSION AND LIMITATIONS

In conclusion, this research has analysed learners' strategies in programming using click-stream data from the PyGuru learning environment using process mining techniques. Our findings indicate that learners employ different strategies when interacting with the course content and programming IDE and these strategies are associated with different levels of learning.

One of the key insights from this study is that high-scoring learners execute their codes more often than low-scoring learners. These high-scoring learners also re-visit the content after code execution, possibly to identify the reasons for the error in the code or the unexpected output. We also see high-scoring learners employing active video-watching behaviour like pausing the video to absorb and reflect, which is missing in low-scoring learners. Another insight from this study is that although low-scoring learners use the highlight feature frequently, they do not visit the highlight page containing all the highlights learners have done.

We will now discuss the implications of this study. The following strategies were observed to be adopted by the learners. Each strategy is discussed for high and low-scoring learners.

Understanding the learning process: This study sheds light on the differences in the learning process of high and low-scoring learners. This information can be useful in developing learning strategies that can help learners achieve better outcomes and the desired level of mastery.

Importance of implementing concepts: The results indicate that implementing concepts through executing the code and referring back to the content after execution is crucial for high-scoring learners. This highlights the importance of hands-on experience in learning and reinforcing the concepts learned. However, this strategy was not observed for low-scoring learners. Low-scoring learners must be motivated to apply their learning and verify their understanding by cross-checking the content.

Active video-watching behaviour: The presence of the 'pause' node in the process model of high-scoring learners suggests that active video-watching behaviour is linked to higher learning gains. This can be useful information for educators to design video-based learning activities that engage learners actively.

Non-judicious highlighting: Low-scoring learners' non-judicious use of highlighting suggests that they might not be processing all the information. Educators can use this information to encourage learners to use highlighting more meaningfully. The fact that low-scoring learners never checked what they highlighted suggests that they might not be retaining the information highlighted. Educators can use this information to encourage learners to review their highlights regularly.

This research study offers valuable understanding regarding the variations in learning strategies used by high-scoring and low-scoring learners. This understanding can be utilized to develop learning systems that capitalize on effective learning strategies, resulting in improved learning outcomes. Also, it will enable instructors to provide the learners with targeted feedback and support to improve the overall learning experience for learners in programming education. The study has also shown that the process mining approach helps to view learners' temporal action sequences to better understand their learning behaviours.

We now highlight some of the limitations. Our study is based on data from a single online learning environment, PyGuru, and hence, the findings may not be generalised to other programming languages or learning environments. The sample size used in this study is relatively small, which is preventing us from asserting any claims.

To address concerns regarding the duration of student engagement, future iterations of this study should consider extending the duration of interaction to more closely align with the comprehensive nature of programming topics, which typically demand more extensive periods of study. This adjustment will afford a more accurate representation of the learning process and its associated dynamics.

Immediate future work may include conducting similar studies using a larger sample size. Our work can be extended by using other educational data mining techniques and evaluating the impact of the feedback and support provided to learners using the insights gained from this study. The study can be replicated in other programming languages and learning environments.

REFERENCES

Azcona, D., Hsiao, I.-H., and Smeaton, A. F. (2019). Detecting students-at-risk in computer programming classes with learning analytics from students' digital footprints. *User Modeling and User-Adapted Interaction*, 29:759–788.

Bennedsen, J. and Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM inroads*, 10(2):30–36.

Bruce, K. B. (2018). Five big open questions in computing education. *ACM Inroads*, 9(4):77–80.

Buffardi, K. and Edwards, S. H. (2013). Effective and ineffective software testing behaviors by novice programmers. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 83–90.

Carter, A. S., Hundhausen, C. D., and Adesope, O. (2015). 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*, pages 141–150.

Dodson, S., Roll, I., Fong, M., Yoon, D., Harandi, N. M., and Fels, S. (2018). An active viewing framework for video-based learning. In *Proceedings of the fifth annual ACM conference on learning at scale*, pages 1–4.

Guerra, J., Sahebi, S., Lin, Y.-R., and Brusilovsky, P. (2014). The problem solving genome: Analyzing sequential patterns of student work with parameterized exercises.

Günther, C. W. and Van Der Aalst, W. M. (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer.

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., et al. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 41–63.

Long, P. and Siemens, G. (2014). Penetrating the fog: analytics in learning and education. *Italian Journal of Educational Technology*, 22(3):132–137.

Lu, Y. and Hsiao, I.-H. (2017). Personalized information seeking assistant (pisa): from programming information seeking to learning. *Information Retrieval Journal*, 20:433–455.

Munshi, A., Rajendran, R., Ocumpaugh, J., Biswas, G., Baker, R. S., and Paquette, L. (2018). Modeling learners' cognitive and affective states to scaffold srl in open-ended learning environments. In *Proceedings of the 26th conference on user modeling, adaptation and personalization*, pages 131–138.

Nishane, I., Sabanwar, V., Lakshmi, T., Singh, D., and Rajendran, R. (2021). Learning about learners: Understanding learner behaviours in software conceptual design tele. In *2021 International Conference on Advanced Learning Technologies (ICALT)*, pages 297–301. IEEE.

Nishane, I., Singh, D., Rajendran, R., and Sridhar, I. (2023). Does learner mindset matter while learning programming in a computer-based learning environment? In *2023 International Conference on Technology for Education (T4E)*. IEEE.

Piech, C., Sahami, M., Koller, D., Cooper, S., and Blikstein, P. (2012). Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160.

Price, T. W., Zhi, R., and Barnes, T. (2017). Hint generation under uncertainty: The effect of hint quality on help-seeking behavior. In *Artificial Intelligence in Education: 18th International Conference, AIED 2017, Wuhan, China, June 28–July 1, 2017, Proceedings 18*, pages 311–322. Springer.

Rajendran, R., Munshi, A., Emara, M., and Biswas, G. (2018). A temporal model of learner behaviors in oeles using process mining. In *Proceedings of ICCE*, pages 276–285.

Reimann, P., Frerejean, J., and Thompson, K. (2009). Using process mining to identify models of group decision making in chat data.

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

Saint, J., Gašević, D., and Pardo, A. (2018). Detecting learning strategies through process mining. In *European conference on technology enhanced learning*, pages 385–398. Springer.

Sedrakyan, G., De Weerdt, J., and Snoeck, M. (2016). Process-mining enabled feedback:"tell me what i did wrong" vs."tell me how to do it right". *Computers in human behavior*, 57:352–376.

Sosnovsky, S. and Brusilovsky, P. (2015). Evaluation of topic-based adaptation and student modeling in quizguide. *User Modeling and User-Adapted Interaction*, 25:371–424.

Winne, P. H. and Nesbit, J. C. (2009). Supporting self-regulated learning with cognitive tools. In *Handbook of metacognition in education*, pages 259–277. Routledge.

Yogev, E., Gal, K., Karger, D., Facciotti, M. T., and Igo, M. (2018). Classifying and visualizing students' cognitive engagement in course readings. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, pages 1–10.