Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)
18/2 (2025), 167-180. DOI: http://dx.doi.org/10.21609/jiki.v18i2.1487

# Analysis of Coding Stress Impact on Students Programming Skills with Random Forest and C4.5 Algorithms

M. Akiyasul Azkiya, Yahya Nur Ifriza

Information Systems, Faculty of Mathematics and Natural Sciences, Universitas Negeri Semarang, Gunungpati, Semarang, 50229, Indonesia

*E-mail: akiyasulazk@students.unnes.ac.id, yahyanurifriza@mail.unnes.ac.id*

**Abstract**

Students' stress often impedes their advancement in programming, which demands logical reasoning, an understanding of algorithms, and a firm grasp of basic concepts. This research intends to pinpoint the elements that affect students' programming abilities, explore their connection to stress levels, and assess the effectiveness of the Random Forest and C4.5 algorithms in classifying data. Information was gathered through an online questionnaire involving 744 students in 2024 at various leading universities in Islamabad, Pakistan. The dataset used in this study was sourced from Kaggle, which provides insights into factors affecting students' programming performance and stress levels. The analysis utilized a Confusion Matrix and evaluation metrics like accuracy, precision, recall, and F1-Score. The analysis results indicate that the C4.5 algorithm has a higher accuracy of 68.04% compared to Random Forest, which achieved 65.54%. Additionally, C4.5 outperforms Random Forest in terms of precision, scoring 71.7% versus 65.2%. However, in terms of recall, Random Forest performs better with a score of 66.3%, while C4.5 only reaches 59.6%. This study confirms that interest in programming, debugging skills, mathematical and analytical abilities, and perceptions of programming significantly impact students' performance and stress levels. Students with strong logical abilities and adequate support demonstrate better performance and lower stress levels, whereas those with weak technical skills and negative perceptions are more vulnerable to stress, which adversely affects their performance. These findings emphasize the importance of creating a positive learning environment through interactive methods, structured problem-solving, and additional support.

**Keywords:** *C4.5 Algorithm, Programming Skills, Random Forest, Stress Management, Student Stress*

## 1. Introduction

In the digital era, programming skills have emerged as a primary competency essential across various industrial sectors, including data analysis, process automation, and technology-based application development [1], [2]. Programming is becoming a fundamental skill akin to reading and writing, with projections indicating a 90% increase in demand for advanced programming skills between 2016 and 2030 [3]. Universities and institutions are crucial in bridging the gap between academic education and industry needs by aligning programming curricula with professional standards [4]. This includes teaching algorithmic thinking and computational skills, which form the foundation of programming [5]. Despite the growing urgency to master programming, learning it poses unique student challenges. One of the primary challenges is the psychological pressure experienced during the learning process, commonly known as coding stress [6]. Students often struggle to understand programming concepts, logical thinking, and syntax usage [7].

Academic pressure can result in a decrease in academic performance, as it influences students' motivation. This situation arises from the excessive expectations placed on students, which can impede their academic success [8], [9]. Coding stress is important because it affects students' learning experiences and academic achievements. Studies on brain activity indicate that students endure considerable mental strain while engaging in programming tasks, although this stress is not always directly correlated with their grades or outcomes [10]. Elements like learning styles, mindsets, emotions, and task difficulty also shape students' learning experiences and performance in coding endeavors [11]. Emotions are vital in mastering intricate tasks like programming. Negative feelings, such as confusion, boredom, and frustration, can adversely impact learning outcomes in the short and long term [12].

On the other hand, positive feelings such as joy and enthusiasm can significantly improve attitudes towards programming and accelerate knowledge acquisition [13]. The intensity of stress can fluctuate considerably based on the cognitive

demands of the task. In programming, tasks with greater complexity can provoke higher stress levels, influencing performance [14]. Programmers frequently encounter high levels of job-related stress, displaying symptoms such as fatigue, tension, and a diminished capacity to handle pressure [15].

Programming anxiety can be a considerable challenge due to various factors, including insufficient prior experience, heavy workloads, and the intrinsic difficulties associated with coding tasks [16]. Learning environments, socio-psychological influences, physical conditions, and cognitive capacities have been recognized as significant factors contributing to stress in educational settings [17]. To promote learning effectiveness and alleviate stress, educational institutions must consider these aspects when developing learning environments. Furthermore, it is essential to comprehend how anxiety impacts programming education, as it can hinder students' ability to tackle challenges and potentially result in elevated failure rates in computer science programs [16]. Consequently, recognizing and addressing coding stress is critical to enhancing educational success in programming.

In recent years, initiatives aimed at understanding coding stress and its implications for programming education have captured the interest of researchers. Multiple strategies have been employed to identify and assess the levels of stress experienced by students; however, research examining stress analysis during the programming learning process remains limited. The most prevalent methods used include psychological surveys, interviews, and various qualitative approaches to investigate individuals' subjective experiences. Although these techniques are useful for gaining insights into students' emotional states, they exhibit notable drawbacks. Traditional pre-and-post surveys might overlook fluctuations in students' emotions over time, which could result in an inaccurate depiction of anxiety levels [18]. The Experience Sampling Method (ESM) provides a more detailed approach by capturing real-time emotional changes, offering improved predictive potential for academic outcomes [18]. The widespread reliance on student surveys raises concerns about the reliability and validity of data in decision-making processes [19]. To overcome these shortcomings, researchers should consider adopting diverse methods, including advanced technologies, to obtain a more thorough understanding of students' emotional and psychological experiences [19], [20].

One rising concern is the insufficient integration of stress assessment with direct analysis of programming abilities. Current research often differentiates stress evaluation from programming performance measurement, creating obstacles in identifying correlations between the two. Certain studies have examined methods to assess cognitive load and stress in software development employing modern technologies. For example, EEG has become a favored technique for observing cognitive load during coding tasks [21]. Moreover, wearable devices have been utilized to measure stress in practical programming situations, effectively differentiating between high-stress instances and relaxed moments [22]. Nevertheless, most research segregates stress evaluation from programming skills assessment [23]. Consequently, the outcomes frequently vary among individuals, complicating the effort to develop universally applicable models. The incorporation of data science and machine learning methods in this area remains limited despite their considerable potential to offer more objective and precise insights [24]. Therefore, this study intends to address this gap by employing advanced analytical algorithms to comprehend the intricate relationship between coding stress and programming performance.

This study suggests a machine learning-driven methodology utilizing Random Forest and C4.5 algorithms to tackle the identified issues. These algorithms were chosen for their exceptional capabilities in analyzing complex datasets and recognizing significant patterns quantitatively. Random Forest is a well-known ensemble learning method noted for its reliability in processing large datasets with many variables. It demonstrates improved predictive performance for first-year computational science classes compared to traditional Decision Tree algorithms [25]. This algorithm performs thorough analysis by aggregating outputs from several decision trees, resulting in more accurate and less biased predictions. Random Forest excels at managing extensive datasets with numerous variables, employing feature selection techniques such as varImp(), Boruta, and Recursive Feature Elimination (RFE) [26]. Recent research has examined Random Forest's effectiveness in predicting students' stress levels. These models incorporate various elements, such as sleep quality, depression rates, academic achievement, and involvement in extracurricular activities, to gauge stress levels among students [27]. Random Forest has shown remarkable accuracy, achieving 95% [27] and 94.73% [28]. In this research, Random Forest is used to identify crucial factors that affect coding stress and students' programming performance and evaluate the significance of the relationships among these variables.

Conversely, the C4.5 algorithm was selected for

its capacity to produce easily understandable decision trees. This algorithm also offers a substantial theoretical basis to ascertain the most effective methods for practical applications [29]. The benefits of C4.5 include its capability to manage categorical data, resolve overfitting challenges, handle datasets with missing values, and perform better than the ID3 algorithm, even when dealing with an equal number of attributes [30]. The algorithm furnishes analytical results and straightforwardly presents models, easing the interpretation of research conclusions. C4.5 has been employed in various educational settings to analyze and visualize trends in student data. It has been utilized to illustrate patterns in student admissions to engineering and computer science programs by forming decision trees based on factors like test scores and interviews [31]. The algorithm has also been used to depict the evolution of programming skills by evaluating students' source code over time [32]. The C4.5 classification algorithm has shown its ability to forecast student performance with an accuracy rate of 71.9%, providing notable advantages for students, educators, and academic institutions in enhancing understanding and improving academic outcomes [33]. By using C4.5, this study seeks to visually represent the connection between coding stress levels and programming skills in a more transparent and more informative manner.

Both algorithms have effectively revealed connections between complex variables and academic results across different educational settings. Utilizing these methods to analyze educational data helps researchers identify patterns that traditional techniques, such as descriptive statistics or linear regression, may overlook due to their inability to handle intricate variable interactions. This research offers insights into the impact of coding-related stress on students' programming skills while proposing strategies to enhance learning outcomes. It also addresses a gap in the existing literature concerning the quantitative assessment of stress, employing machine learning algorithms like Random Forest and C4.5 to analyze complex datasets objectively. The results pave the way for future research, including examining additional factors that affect programming abilities or creating predictive tools for early detection of academic risks, ultimately enhancing the quality of programming education across various educational levels and professional training programs.

## 2. Research Method

This research was conducted through several structured stages to achieve the main objective, which is to analyze the relationship between coding stress and students programming abilities using the Random Forest and C4.5 algorithms. This approach was designed to ensure that each research step is carried out systematically and supported by a valid scientific framework. The stages of this research are explained as follows, as illustrated in Figure 1.

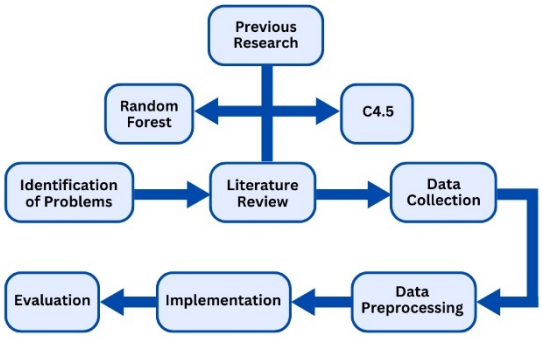


**Figure 1.** Research stage.

### *Identification of Problem*

In the initial stage of problem identification, this study focuses on the phenomenon of coding stress experienced by students and its impact on their programming abilities. Coding stress is defined as a psychological condition characterized by anxiety, confusion, and uncertainty when dealing with programming tasks. This condition can hinder logical thinking, reduce concentration, and impair the ability to effectively comprehend and solve complex programming problems.

This research explicitly explores how coding stress relates to several key factors that shape students' programming skills. These factors include proficiency in programming languages, debugging capabilities, understanding fundamental programming concepts, and psychological aspects such as learning motivation and coding-related anxiety. By examining the interrelationships among these variables, the study aims to determine how coding stress contributes to either the deterioration or improvement of students' programming performance.

As part of the problem identification process, a literature review was conducted to identify gaps in existing measurement approaches, particularly in assessing stress levels and their effects on programming performance. Many previous studies have not measured the effects of coding stress quantitatively while considering relevant variables. Therefore, this research seeks to address that gap through a data-driven approach.

The dataset employed in this study comprises various variables directly related to students' programming capabilities. These include the programming languages they have mastered, levels of debugging skills, understanding of basic

programming principles, and their learning motivation. In addition, variables such as coding anxiety, frequency of practice, task difficulty, perceptions of teaching quality, and the availability of additional support are also considered. These elements are interrelated and will be comprehensively analyzed to demonstrate the significant impact of coding stress on students' programming performance. By gaining deeper insight into these relationships, the findings of this study are expected to contribute to the development of more effective and psychologically responsive programming education strategies.

***Literature Review***

The literature review in this study is structured to deepen the understanding of the phenomenon of coding stress experienced by students and its relationship with their programming abilities. Coding stress often arises from the pressure to complete assignments within tight deadlines, difficulties understanding abstract programming concepts, and the fear of failure in writing or executing programs. These situations can trigger feelings of anxiety, frustration, and a loss of motivation to learn, particularly among novice students.

Several previous studies have shown that high levels of anxiety can disrupt students' logical thinking, problem-solving skills, and accuracy in writing and debugging code. This condition directly affects academic performance, learning quality, and the achievement of learning outcomes in programming courses. However, despite the growing body of research highlighting stress in educational settings, quantitative approaches remain lacking that specifically connect stress levels to students' programming performance comprehensively.

In this study, the literature review also highlights the importance of applying machine learning algorithms as a novel approach to analyzing the relationship between stress and programming abilities. Two algorithms emphasized in this study are Random Forest and C4.5, which fall under decision tree-based classification methods.

The Random Forest algorithm is an ensemble learning technique that generates multiple decision trees from different subsets of the data. Then, it combines their outputs to obtain more accurate and stable predictions. This algorithm is well-suited for handling datasets with numerous interrelated variables, such as students' psychological and academic factors. With its ability to reduce overfitting, Random Forest is a reliable tool for evaluating complex patterns between stress and programming performance.

On the other hand, the C4.5 algorithm is one of the most widely used decision tree algorithms in predictive model development. C4.5 divides the data based on the most informative attribute values using the gain ratio and constructs a tree structure capable of mapping stress-related variables into specific performance categories. Through this model, researchers can gain more precise insights into which factors most significantly influence students' programming performance.

***Data Collection***

This study utilizes a dataset from the open-source Kaggle repository titled "Behavioral Analysis of Programming Students" [34]. The dataset was developed through a survey distributed to various leading universities in Islamabad, Pakistan, to identify factors contributing to weak programming skills among computing students. Additionally, this study explores psychological and social aspects that influence learning outcomes, such as social comparison, jealousy, anxiety, and stress.

The survey involved 744 students from various undergraduate and postgraduate programs in computing, providing a diverse overview of academic backgrounds and programming proficiency levels. The respondents in this dataset come from 12 identified universities, namely Air University, FAST University, Bahria University, Fatima Jinnah Women University, FUUAST (Federal Urdu University of Arts, Science & Technology) Islamabad, Hamdard University Islamabad, Quaid-i-Azam University (QAU), COMSATS University, Islamic International University Islamabad (IIUI), Foundation University, NUST (National University of Sciences and Technology), and NUML (National University of Modern Languages).

Additionally, the respondents are enrolled in various undergraduate and master's programs, particularly in technology, data, and engineering fields, such as BS AI (Artificial Intelligence), BS DS (Data Science), BS CS (Computer Science), BS IT (Information Technology), BS SE (Software Engineering), BS CYS (Cybersecurity), BS CGD (Computer Game Development), BS CE (Computer Engineering), BS EE (Electrical Engineering), and the master's program MS DS (Master of Science in Data Science). Regarding the semester distribution recorded in the dataset, most students are in the third semester, followed by the fifth and first semesters with significant numbers. Meanwhile, students in the seventh

**Table 1.** Behavioral analysis of programming students dataset.

| Variable Name | Description | Type |
|---|---|---|
| Proficient Language | Primary programming language mastered. | Object |
| Extra Challenges | Participation in additional coding challenges. | Object |
| Learning New Concepts | Methods for learning new programming concepts. | Object |
| Memorize Solutions | Tendency to memorize solutions. | Object |
| Debugging Skill | Ability to fix coding errors. | Float |
| Problem Identification | Capability to identify programming issues. | Float |
| Logic Building | Skill in constructing programming logic. | Float |
| Syntax Memory | Ability to recall programming syntax. | Float |
| Coding Interest | Interest in programming. | Float |
| Career Importance | Relevance of coding to career goals. | Int |
| Dry Running Skills | Manual code execution to identify bugs. | Float |
| Hard Coding Skills | Advanced coding abilities. | Float |
| Math and Analytical Skills | Proficiency in math and analytical problem-solving. | Float |
| Logical Puzzles | Skill in solving logical puzzles. | Float |
| Programming Fundamentals | Understanding basic programming concepts. | Float |
| OOP Proficiency | Competence in Object-Oriented Programming (OOP). | Float |
| Database Proficiency | Expertise in databases. | Float |
| Data Structures Proficiency | Knowledge of data structures. | Float |
| Algorithm Proficiency | Skill in designing and using algorithms. | Float |
| Coding Challenges | Experience in solving coding challenges. | Object |
| Problem-Solving Approach | Approach to solving programming problems. | Object |
| Coding Anxiety | Anxiety experienced during coding. | Object |
| Practice Frequency | Frequency of coding practice. | Object |
| Extra Learning | Learning outside academic materials. | Object |
| Assignment Difficulty | Perceived difficulty of coding assignments. | Object |
| Assignment Solutions | Approach to completing coding assignments. | Object |
| Copy-Paste Shame | Embarrassment about copying solutions. | Object |
| Coding Perception | Perception of coding activities. | Object |
| Extra Help | Frequency of seeking additional help. | Object |
| More Practice Requests | Need for additional coding practice. | Object |
| Mood Impact | Impact of coding on mood. | Object |
| Learning Style | Preferred style of learning programming. | Object |
| Motivation | Motivation for learning programming. | Object |
| Instructor Delivery | Assessment of instructors' delivery of material. | Object |
| Exam Preparation | Methods for preparing for coding exams. | Object |
| Procrastination | Habit of delaying coding tasks. | Object |
| Difficult Problem Action | Strategies for tackling difficult problems. | Object |
| Problem Approach | Strategies for initiating problem-solving. | Object |
| External Factors | External factors influencing coding. | Object |
| Skill Improvement Belief | Belief in the ability to improve coding skills. | Object |
| Extra Projects | Involvement in additional projects. | Object |
| Coding Stress | Level of stress experienced during coding. | Object |

semester appear in smaller numbers, and those in the second, fourth, sixth, and eighth semesters have relatively lower distributions than others. An overview of significant dataset variables is provided in Table 1.

The dataset used in this study consists of 59 initial features covering various aspects related to students' programming abilities and psychological factors. After the feature selection process, 43 features were selected for further analysis, while the remaining 16 were excluded. The feature selection was conducted carefully, considering their relevance to the research objective, which is to analyze the impact of stress on students' programming abilities.

***Data Preprocessing***

In the data preprocessing phase, the initial step is data cleaning to ensure the dataset is high quality and prepared for analysis. Data cleaning involves identifying and managing irrelevant or duplicate data, which can interfere with the analysis process and diminish model accuracy. The effectiveness of machine learning models is significantly influenced by data cleaning, with various impacts on classification tasks based on the cleaning technique and the data inaccuracies present. Moreover, missing data is a prevalent challenge in datasets that must be meticulously addressed. There are several methods to tackle missing data, such as imputing values using the mean, median, or mode for numerical variables or employing other strategies suited to the nature of the missing data, such as deleting particular rows or columns if the absence of data is substantial.

The subsequent step is normalization once the missing data has been dealt with. Normalization aims to standardize the scale of variables with varying value ranges. This process enhances the

accuracy of classification and the quality of clustering outcomes [35]. It is vital to ensure that machine learning algorithms, such as Random Forest and C4.5, function effectively without being impacted by variations in feature scales. After completing these preprocessing tasks, the dataset that has been cleaned, normalized, and encoded will be prepared for analysis using the Random Forest and C4.5 algorithms. The dataset used in this research initially experienced class imbalance, particularly in the distribution of student stress levels, where there was a significant difference in the amount of data between the low, moderate, and high-stress categories. This imbalance can cause machine learning models to learn biased patterns, tending to predict the majority class more accurately than the minority class. To address this issue, this study applies to the Synthetic Minority Over-sampling Technique (SMOTE). This creates a new sample through linear interpolation between minority class examples and their K nearest neighbors [36]. This method balances the class distribution and improves the model's ability to recognize patterns from previously underrepresented groups.

***Implementation***

In the implementation phase, once the data has been adequately prepared and preprocessed, the next stage is to apply machine learning algorithms to examine the connection between coding stress levels and students' programming skills. RapidMiner is a powerful and trustworthy tool for machine learning analysis, presenting several benefits over other platforms like Python. Research indicates that RapidMiner excels in accuracy and efficiency in execution time for supervised learning algorithms, such as Decision Trees, Support Vector Machines, and Neural Networks [37]. Additionally, RapidMiner has shown itself to be optimal for various classification tasks, with algorithms like Random Forest and ID3 achieving high accuracy rates, even in scenarios such as airplane-type classification [38]. Another notable benefit is RapidMiner's automated modeling capability, which is advantageous for various business intelligence applications, especially in price prediction [39].

For the analysis in this study, two algorithms are utilized: Random Forest and C4.5. Random Forest is a collective algorithm combining multiple decision trees to improve prediction accuracy. Each decision tree is created by selecting random subsets of features and training data, resulting in a more robust model and reducing the risk of overfitting. Predictions are generated by taking a majority vote from all the trees in the forest to determine the final result. The foundational formula for forming a decision tree in Random Forest is demonstrated in Formula 1.

$$Entropy\ (S)\ = -\sum_{j=i}^{k} \frac{[Si]}{[S]} *\ log_2\ (\frac{[Si]}{[S]}) \quad (1)$$

In this formula, $\boldsymbol{S}$ denotes the entire dataset being analyzed, and $\boldsymbol{k}$ represents the number of classes or categories within $\boldsymbol{S}$. Each $S_i$ is a subset of $\boldsymbol{S}$ that belongs to a specific class, and $\frac{[\boldsymbol{Si}]}{[\boldsymbol{S}]}$ indicates the proportion of data in that class. This entropy measure evaluates the level of disorder or uncertainty in the dataset before it is split by the decision tree.

The C4.5 algorithm, another decision tree method, constructs trees by selecting attributes that yield the highest information gain ratio at each step. C4.5 can effectively handle both categorical and numerical data. Like Random Forest, the process begins by calculating entropy to assess the diversity in the dataset. The entropy formula used in C4.5 maintains the same structure as that employed in Random Forest, shown in Formula 2.

$$Entropy\ (S)\ = -\sum_{j=i}^{n} \frac{|Si|}{|S|} *\ log_2\ (\frac{|Si|}{|S|}) \quad (2)$$

Here, $\boldsymbol{n}$ denotes the number of partitions or subsets formed based on a specific attribute. Each fraction $\frac{[\boldsymbol{Si}]}{[\boldsymbol{S}]}$ represents the proportion of instances in the $\boldsymbol{i-th}$ partition. This measure is essential for understanding the variation within data segments. Once entropy is calculated, the gain value for each attribute is determined using formula 3.

$$Gain\ (S, A) = Entropy(S) - \sum_{i=1}^{n} \frac{|Si|}{|S|} * Entropy\ (S_i) \quad (3)$$

In this equation, $\boldsymbol{A}$ refers to the attribute being evaluated, $\boldsymbol{Si}$ represents the subset of $\boldsymbol{S}$ resulting from splitting based on attribute $\boldsymbol{A}$, and $\frac{[\boldsymbol{Si}]}{[\boldsymbol{S}]}$ indicates the proportion of instances in that subset. The gain value reflects the reduction in entropy achieved by partitioning the data using a particular attribute. The attribute with the highest gain is selected as the root node, and the process is repeated recursively to construct an optimal decision tree for classification.

***Evaluation***

During the evaluation phase, the models produced from applying the Random Forest and C4.5 algorithms will be assessed to evaluate their effectiveness in predicting the correlation between coding stress and students' programming skills. Implementing sound model evaluation practices, model selection, and algorithm selection methods

is essential to guarantee the accuracy and dependability of machine learning models [40]. Several performance metrics, including accuracy, precision, recall, and F1-score, will be key indicators to evaluate the effectiveness of the classification model. Accuracy measures how frequently the model produces correct predictions compared to the total number of predictions made, as illustrated in Formula 4.

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad (4)$$

Precision assesses the proportion of correct optimistic predictions the model makes against the total positive predictions it generates. This metric is significant for evaluating the model's capacity to minimize errors when generating positive outcomes, as defined in Formula 5.

$$precision = \frac{TP}{TP+FP} \quad (5)$$

Recall gauges the model's capability to identify all positive cases within the dataset. This is vital to ensure that no positive cases that need to be recognized are overlooked, as stated in Formula 6.

$$recall = \frac{TP}{TP+FN} \quad (6)$$

The F1-score represents the harmonic mean of precision and recall, balancing the two. This metric is particularly valuable in scenarios where there is an imbalance between the quantity of positive and negative data points, as defined by Formula 7.

$$F1-score = 2\,\frac{precision * recall}{precision + recall} \quad (7)$$

True Positive (***TP***), True Negative (***TN***), False Positive (***FP***), and False Negative (***FN***) are essential metrics in evaluating the performance of predictive models. A True Positive occurs when the model correctly predicts a positive outcome, while a True Negative refers to a correct prediction of a negative outcome. In contrast, a False Positive happens when the model incorrectly predicts a positive result for a negative case, and a False Negative occurs when the model predicts a negative result for a case that is positive. These metrics are crucial for assessing the accuracy and overall effectiveness of a classification model.

## 3. Result and Analysis

In this investigation, data analysis was performed utilizing the Random Forest and C4.5 algorithms, executed through RapidMiner software. The analytical procedure commenced with a data preprocessing phase involving cleaning, normalization, and transformation to meet the analysis standards. Subsequently, the prepared data was divided into two primary subsets: training data (80%) and testing data (20%) [41]. This ratio was selected to ensure that the model had ample data for training while retaining enough for a fair assessment of its performance. During the execution phase, a systematic workflow was developed in RapidMiner to delineate each stage of the analysis process. Figure 2 shows the process from preprocessing to model evaluation.

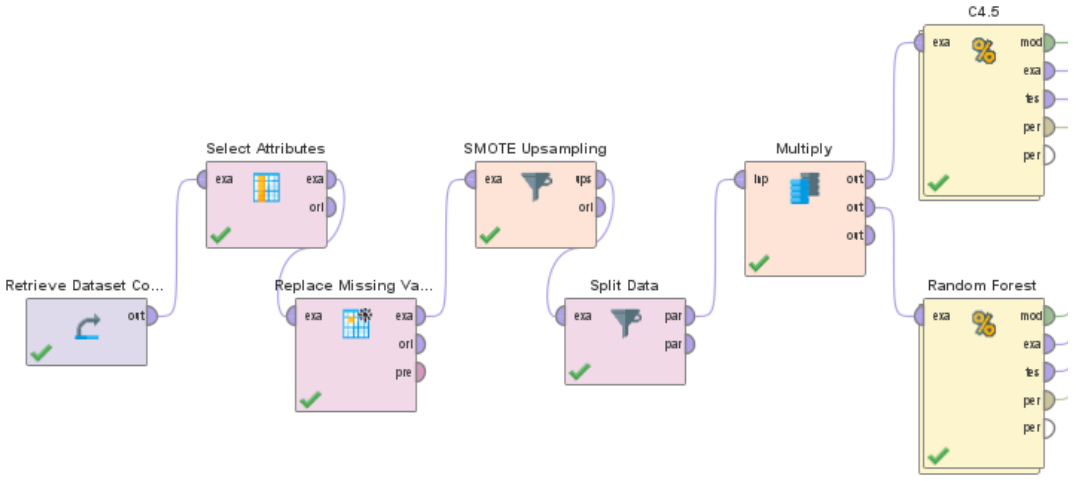


**Figure 2.** Design process for calculation in RapidMiner using Random Forest and C4.5 algorithms.

The approach taken with the Random Forest algorithm offers a thorough understanding of the patterns within the analyzed data. This algorithm employs a combination of multiple decision trees to uncover the relationships between coding stress levels and various factors impacting students' programming competencies. The visual depiction of the results from this analysis is shown in Figure 3.

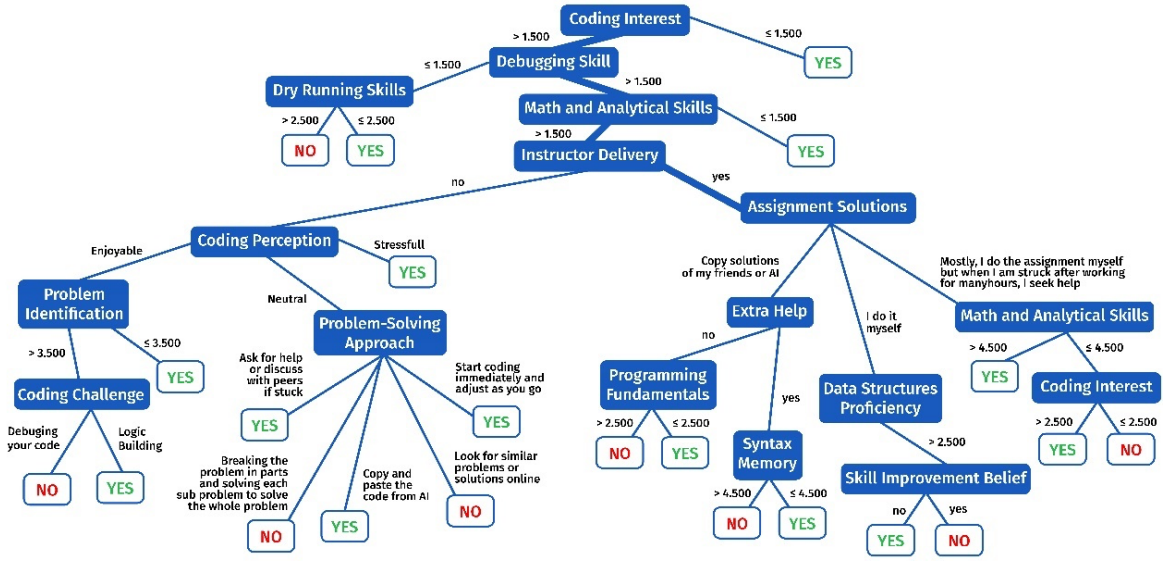


**Figure 3.** Decision tree model generated using the Random Forest algorithm.

According to Figure 3, the results of the analysis using the Random Forest algorithm in this study indicate that stress during coding activities significantly affects students' programming skills. The stress experienced during the learning process of programming not only impacts students' psychological aspects but also influences their problem-solving patterns, debugging abilities, and confidence in completing programming tasks. Based on the decision tree generated, it was found that interest in coding (Coding Interest) serves as the initial factor that differentiates students with high and low programming skills. Students interested in coding are more likely to develop their programming skills. In contrast, those with low interest tend to experience stress more easily when facing challenges in the coding process.

The second factor identified in this analysis is debugging skills. Students with debugging skills higher than 1.5 tend to cope better with stress than those with lower debugging skills. This finding suggests that debugging skills are essential for identifying and fixing errors in code but also play a role in reducing anxiety caused by uncertainties during programming. Adequate debugging skills give students greater confidence when encountering errors or bugs, thus reducing mental pressure compared to students with lower debugging abilities. Additionally, mathematical and analytical skills (Math and Analytical Skills) are crucial in determining students' stress levels. Students with mathematical and analytical skills above 1.5 tend to have better problem-solving abilities. In this context, analytical skills help students break down problems into smaller, more manageable parts, thereby reducing stress arising from the complexity of programming tasks. Conversely, students with lower mathematical skills are more prone to stress due to difficulties in understanding logical patterns and code structures.

Perception of coding (Coding Perception) emerges as a direct indicator of students' stress levels. Students who perceive coding as enjoyable tend to have better programming abilities than those with a neutral perception or those who find coding stressful. Students who enjoy coding are more inclined to use logic-based problem-solving strategies and engage in peer discussions. Meanwhile, students who view coding as stressful are likelier to adopt less effective problem-solving strategies, such as copying code from AI or peers without understanding the underlying concepts. Stress during coding is also reflected in the problem-solving strategies chosen by students. Stressed students tend to adopt a "Start coding immediately and adjust as you go" approach, where they begin writing code without proper planning. This pattern indicates time pressure or anxiety that drives students to produce output quickly without considering the structure and conceptual understanding of the program. In the long run, this strategy exacerbates stress by increasing the likelihood of coding errors and prolonging the debugging process.

Beyond internal student factors, this analysis also highlights the importance of instructional delivery (Instructor Delivery). Students who perceive the instructional delivery as ineffective are more susceptible to stress, as evidenced by an increased tendency to copy assignments from peers or external sources without a deep understanding. On the other hand, students who are satisfied with the instructional delivery exhibit greater confidence in completing programming tasks

independently. This finding underscores the importance of interactive, clear, and contextually relevant teaching approaches to help reduce students' stress levels. Additional support (Extra Help) in learning programming is also found to be closely related to stress levels. Students who receive supplementary assistance are generally better at overcoming challenges encountered during coding. This support may include additional explanations from lecturers or teaching assistants, peer discussions, or the use of relevant learning resources. In contrast, students without additional support tend to experience higher stress levels, particularly when facing complex problems that require in-depth understanding.

Problem-solving abilities related to proficiency in data structures also reduce students' stress levels. Students with good data structure knowledge find navigating and comprehending their code easier. The study revealed that students with data structure skills above 2.5 are less likely to experience stress than those with lower skills. Complex data structures often become a source of stress, especially when students encounter algorithmic and code efficiency issues.

Lastly, the analysis indicates that students with low coding interests (Coding Interest ≤ 1.5) tend to experience higher stress levels and demonstrate lower programming performance. This phenomenon can be attributed to low interest and factors such as reduced learning motivation, difficulties grasping fundamental concepts, and unpreparedness to face programming challenges. Therefore, increasing students' interest in programming through engaging and practical learning approaches is crucial in mitigating stress and enhancing programming skills. Overall, the findings of this study illustrate that stress in programming education is not merely a psychological issue but is closely related to technical factors such as debugging skills, problem-solving strategies, and data structure proficiency. Learning approaches that emphasize a strong conceptual foundation, provide adequate support, and foster a conducive learning environment can significantly help reduce students' stress levels and improve their programming skills.

The results from the analysis utilizing the C4.5 algorithm provide detailed insights into the patterns found in the examined data. This algorithm produces a decision tree demonstrating the relationships among independent variables, including coding stress levels and other aspects, as depicted in Figure 4.

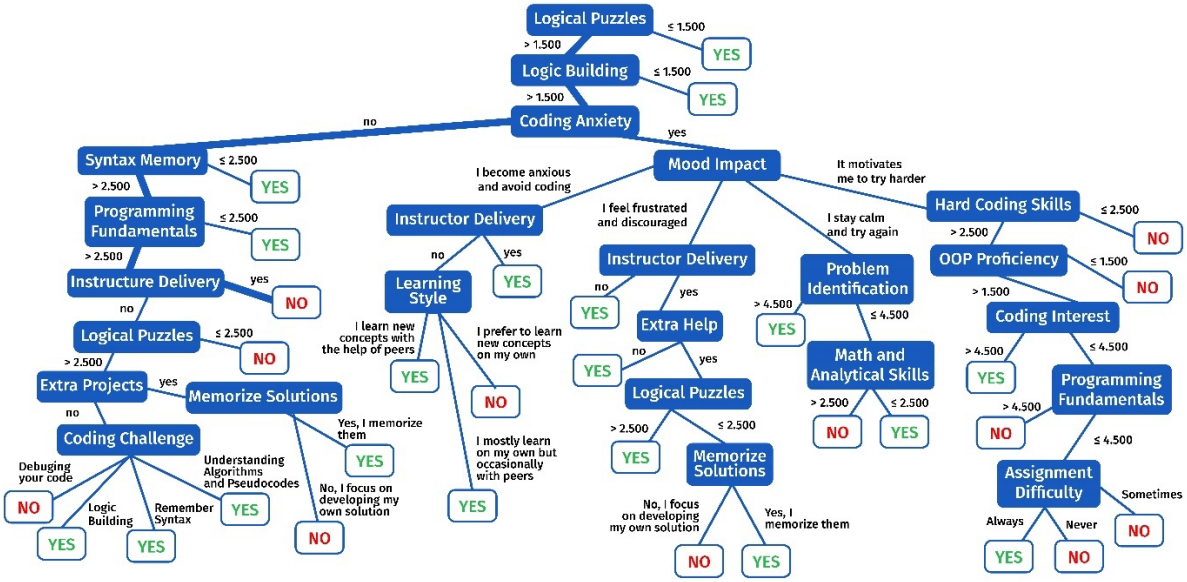


**Figure 4.** Decision tree model generated using the C4.5 algorithm, which illustrates the decision-making process by splitting the dataset into branches based on the most relevant attributes.

Figure 4 showcases the Decision Tree C4.5 modeling results in this analysis reveal various factors influencing programming-related stress, particularly concerning logic, syntax memory, and coding anxiety. The generated decision tree structure maps the relationships between variables with specific rules that provide in-depth insights into the patterns formed. At the first level, the factor "Logical Puzzles," with a threshold of 1.500, serves as the initial determinant. If the score in this aspect exceeds 1.500, programmers have a lower potential for stress, especially when accompanied

by a high score in "Logic Building." Well-developed logical skills lay the foundation for addressing programming challenges. If "Logic Building" exceeds 1.500 and is not accompanied by coding anxiety (Coding Anxiety = No), "Syntax Memory" becomes the next critical factor.

Syntax memory, with a threshold of 2.500, assesses how well a programmer can recall and apply programming syntax rules. When this score is high, combined with a firm grasp of programming fundamentals, the next determining factor is "Instructor Delivery." Suppose the instructor's delivery is perceived as unhelpful. In that case, programmers become more prone to stress unless they participate in additional projects (Extra Projects) and actively engage in logic- and algorithm-related coding challenges. However, if programmers exhibit "Coding Anxiety" with a "Yes" response, the "Mood Impact" factor becomes the next indicator. Mood influences programming stress by avoiding programming, feeling frustrated, staying calm, and trying again to feel motivated to persist. Programmers who avoid programming often experience poor "Instructor Delivery" and rely more on peer-based learning. Conversely, those who remain calm or motivated tend to have higher "Problem Identification" skills, supported by sufficient mathematical and analytical abilities.

Furthermore, basic programming skills (Hard Coding Skills) and object-oriented programming proficiency (OOP Proficiency) become decisive elements for programmers who remain motivated despite stress. If OOP skills are high and coding interest is strong, they are more likely to endure complex programming tasks. Conversely, if interest is low, they are more likely to give up easily. Overall, this analysis indicates that programming stress is influenced not only by cognitive factors such as logic and memory but also by affective factors like Anxiety, mood impact, and support from instructors and the learning environment.

Understanding the relationship between stress and student performance in programming requires a deeper exploration of relevant educational theories, particularly those related to motivation and learning. Motivation is crucial in students' ability to master programming skills, as it influences their engagement, perseverance, and cognitive strategies. In this context, several psychological and educational theories provide a framework for understanding how students respond to stress and build resilience in the programming learning process.

One relevant concept is self-efficacy, which refers to an individual's belief in their ability to succeed in a specific task, as proposed by Schunk [42]. Students with high self-efficacy tend to engage in problem-solving activities, persist in difficulties, and develop effective learning strategies. This belief fosters their confidence when facing programming challenges, reduces anxiety, and promotes a growth mindset. Self-efficacy in programming can be enhanced through various factors. Previous programming experience has been shown to positively impact self-efficacy, which increases as students progress through introductory programming courses [43]. Additionally, programming mental models play a crucial role in shaping students' self-efficacy and academic performance [43]. Students with low self-efficacy can sometimes improve their confidence through guided mentoring [44]. Furthermore, inquiry-based learning (IBL) has been proven to boost students' self-efficacy, particularly those from non-IT backgrounds in interdisciplinary learning environments [45]. IBL helps students see programming as a practical problem-solving tool, increasing their achievement, motivation, and learning satisfaction. Based on the analysis in this study, it was found that self-efficacy, programming experience, and student mindset significantly correlate with programming performance. The Random Forest algorithm revealed that students with high self-efficacy had higher success rates in completing programming tasks. This indicates that self-efficacy influences students' intrinsic motivation, consistent with Schunk's [42] view that strong self-efficacy drives academic achievement.

Meanwhile, the analysis using the C4.5 algorithm identified patterns supporting the importance of feedback and learning experiences in enhancing students' self-efficacy. The resulting decision tree showed that students who received constructive feedback and experienced small, gradual successes demonstrated significant improvements in their programming skills. This finding aligns with the theory proposed by Rhahne and Wijnia [46] regarding the importance of interactions among situational, self, goal, and action factors in building learning motivation. In a broader context, Rhahne and Wijnia [46] introduced an integrative framework of educational motivation theory comprising six key stages: situation, self, goal, action, outcome, and consequence. This framework explains how stress, self-efficacy, and motivation influence students' programming performance. The learning environment, including curriculum, teaching methods, and academic pressures, significantly affects students' learning experiences. When complex tasks and tight schedules become sources of stress, individual factors such as self-efficacy determine how students respond to these challenges.

Furthermore, students' academic goals and intrinsic motivation influence how they face programming challenges. Students with a mastery-oriented mindset tend to perceive difficulties as opportunities to grow rather than threats that increase stress. Their learning strategies, such as independent exploration and problem-solving approaches, contribute to developing critical and algorithmic thinking skills. The results they achieve, whether in grades or project success, reflect the effectiveness of their learning strategies and motivation levels. Success in programming tasks reinforces self-efficacy and resilience to stress, while failure, if not accompanied by constructive feedback, can increase anxiety and lower learning motivation. The findings of this study reinforce previous research emphasizing that self-efficacy, programming experience, and students' mindsets are critical factors in the successful learning of programming.

A study was carried out utilizing the Confusion Matrix to analyze the efficacy of the Random Forest and C4.5 algorithms in classifying the factors influencing students' programming abilities. The results for the Random Forest algorithm's Confusion Matrix are detailed in Table 2, while the outcomes for the C4.5 algorithm can be referenced in Table 3.

**Table 2.** Confusion matrix results for the Random Forest.

| | True Yes | True No |
|---|---|---|
| Pred. Yes | 305 | 162 |
| Pred. No | 155 | 298 |

The confusion matrix results from the Random Forest algorithm demonstrate the model's performance in predicting data through four key components. First, 305 True Positive (***TP***) indicate the number of "Yes" labeled data points correctly predicted. This figure reflects the model's ability to recognize positive data effectively. Next, 298 True Negative (***TN***) represent the model's success in accurately identifying "No" labeled data. On the other hand, the model also produces False Positive (***FP***), which are prediction errors where the model classifies negative data as positive (Type I Error). This number indicates the model's tendency to misclassify harmful data as positive. Additionally, there are 155 False Negative (***FN***) occurring when the model predicts "No" for data that is labeled "Yes" (Type II Error). The ***FN*** count highlights positive data that the model failed to detect.

**Table 3.** Confusion matrix results for the C4.5 algorithm.

| | True Yes | True No |
|---|---|---|
| Pred. Yes | 274 | 108 |
| Pred. No | 186 | 352 |

The confusion matrix results from the C4.5 algorithm illustrate the model's performance in predicting data through four main components. First, there are 274 True Positive (***TP***), indicating the number of "Yes" labeled data points correctly predicted. This figure reflects the model's ability to recognize positive data accurately. Next are 352 True Negative (***TN***), indicating that the model successfully predicted "No" labeled data with a relatively high count. However, the model also produced prediction errors, namely 108 False Positive (***FP***), which occur when the model predicts "Yes" for data that is labeled "No." This ***FP*** count is lower than the results of the Random Forest algorithm, suggesting that C4.5 has a lower tendency to misclassify harmful data as positive. Additionally, there are 186 False Negative (***FN***), representing the number of "Yes" mislabeled data points predicted as "No." This relatively high ***FN***, value indicates that the model still struggles to recognize some positive data accurately.

The accuracy comparison between the Random Forest and C4.5 algorithms was analyzed to evaluate their performance in identifying the factors influencing students' programming skills. The accuracy results of both algorithms are detailed in Table 4.

**Table 4.** Accuracy results of Random Forest and C4.5.

| Evaluation Metrics | Random Forest | C4.5 |
|---|---|---|
| Accuracy | 65.54% | 68.04% |
| Precision | 65.2% | 71.7% |
| Recall | 66.3% | 59.6% |
| F1-Score | 65.7% | 65.0% |

Based on the performance evaluation of the Random Forest and C4.5 algorithms, it can be concluded that the C4.5 algorithm performs slightly better overall. This is evidenced by the accuracy of C4.5, which reaches 68.04%, higher than Random Forest's accuracy of 65.54%. This accuracy indicates that the C4.5 algorithm can classify data correctly in a more significant proportion. In terms of precision, C4.5 also outperforms Random Forest with a precision of 71.7%, compared to 65.2% for Random Forest. This suggests that the "Yes" predictions in the C4.5 algorithm are more reliable, producing fewer false positives. However, when viewed from the recall perspective, Random Forest has the advantage with a recall of 66.3%, compared to 59.6% for C4.5. This indicates that Random Forest is better at identifying positive data overall, although it makes more mistakes when predicting harmful data. Meanwhile, in terms of the F1-Score, both algorithms show nearly balanced performance, with Random Forest achieving an F1-Score of 65.7% and C4.5 scoring 65.0%. The F1-Score

reflects the balance between precision and recall, indicating that both models have relatively comparable capabilities in accurately predicting the data.

Based on the results of the McNemar test conducted to evaluate the significance of the performance difference between the Random Forest and C4.5 algorithms, the test statistic was found to be 1.655 with a p-value of 0.198. Using a significance level of 0.05, these results indicate that the p-value is more significant than 0.05. Therefore, there is insufficient evidence to reject the null hypothesis ($H_0$), which states that no significant difference exists between the performance of the Random Forest and C4.5 algorithms in classifying the dataset used. Thus, the observed accuracy differences between the two algorithms can be considered statistically insignificant, indicating that both algorithms have comparable performance in the context of this study.

## 4. Conclusion

Based on the analysis conducted using the Random Forest and C4.5 algorithms in RapidMiner, it was found that various factors influence students' programming performance, particularly stress levels during coding activities. The study identified key factors such as coding interest, debugging skills, mathematical and analytical skills, instructional delivery, and additional support in learning programming. The results indicate that students with higher self-efficacy, debugging skills, and logical abilities tend to cope better with stress and demonstrate higher programming proficiency. Additionally, perception of coding plays a crucial role in students' problem-solving strategies and overall performance. The study also highlights the importance of effective instructional delivery and additional learning support to reduce students' stress levels and enhance their programming skills.

The performance evaluation of the Random Forest and C4.5 algorithms revealed that C4.5 achieved a higher accuracy of 68.04% compared to 65.54% for Random Forest. C4.5 also exhibited superior precision (71.7%) compared to Random Forest (65.2%), indicating a lower tendency to misclassify harmful data. However, Random Forest outperformed C4.5 in the recall, achieving 66.3% versus 59.6%, signifying its ability to detect positive data more effectively. The F1-Score results were comparable for both models, with Random Forest scoring at 65.7% and C4.5 at 65.0%, demonstrating balanced precision and recall trade-offs. The McNemar test indicated no statistically significant difference between the two algorithms, suggesting that both models have similar classification performance in this context.

These findings emphasize the need for educational strategies integrating stress management and skill development to enhance students' programming capabilities. Future research should explore alternative machine learning models, such as deep learning and ensemble techniques, to improve predictive accuracy and gain deeper insights into student learning behaviors. Additionally, incorporating real-time stress monitoring through physiological and behavioral indicators could provide a more comprehensive understanding of stress dynamics during programming tasks.

Further studies could also investigate the impact of personalized learning approaches, such as adaptive learning systems and AI-based tutors, to mitigate stress and enhance students' engagement in programming education. Expanding the dataset with diverse student populations from various educational backgrounds would improve the generalizability of the findings. By implementing these future research directions, educators and researchers can develop more effective strategies to support students in overcoming programming challenges and fostering a more positive learning experience.

## References

[1] Taslim, Susi Handayani, and Wirdah Choiriyah, "Pengabdian kepada Masyarakat Penerapan Sistem Informasi Inventarisasi Aset pada SMK Migas Inovasi Riau," *J-COSCIS : Journal of Computer Science Community Service*, vol. 4, no. 1, pp. 1–7, Jan. 2024, doi: 10.31849/jcoscis.v4i1.15313.

[2] S. Suryadi and F. A. P. Nasution, "Revolusi Industri, Tren Pekerjaan Masa Depan, dan Posisi Indonesia," *Jurnal Ketenagakerjaan*, vol. 18, no. 2, pp. 124–141, Aug. 2023, doi: 10.47198/jnaker.v18i2.237.

[3] A. Çetinkaya and Ö. K. Baykan, "Prediction of middle school students' programming talent using artificial neural networks," *Engineering Science and Technology, an International Journal*, vol. 23, no. 6, pp. 1301–1307, Dec. 2020, doi: 10.1016/j.jestch.2020.07.005.

[4] L. S. Istiyowati, Z. Syahrial, and S. Muslim, "Programmer's Competencies between Industry and Education," *Universal Journal of Educational Research*, vol. 8, no. 9A, pp. 10–15, Sep. 2020, doi: 10.13189/ujer.2020.082002.

[5] S. GÖKOĞLU, "Algorithm Perception in the Programming Education: A Metaphor Analysis," *Cumhuriyet International Journal of Education*, vol. 6, no. 1, pp. 1–14, Mar. 2017, doi: 10.30703/cije.321430.

[6] K. Afandi, "Identifikasi Proses yang Mempengaruhi Kemampuan Pemrograman Mahasiswa Baru tanpa Pengalaman Pemrograman," *JATISI (Jurnal Teknik Informatika dan Sistem Informasi)*, vol. 8, no. 4, pp. 1785–1795, Dec. 2021, doi: 10.35957/jatisi.v8i4.1120.

[7] F. N. Hasanah and R. S. Untari, "Analisis Kemampuan Mendeteksi Eror Kode Program Mata

Kuliah Pemrograman Berorientasi Objek pada Program Studi Pendidikan Teknologi Informasi Universitas Muhammadiyah Sidoarjo," *Teknologi dan Kejuruan: Jurnal Teknologi, Kejuruan, dan Pengajarannya*, vol. 41, no. 2, pp. 139–146, Sep. 2018, doi: 10.17977/um031v41i22018p139.

[8] S. M. Geronimo, A. A. Hernandez, and M. B. Abisado, "Academic Stress of Students in Higher Education using Machine Learning: A Systematic Literature Review," in *2023 IEEE 13th International Conference on System Engineering and Technology (ICSET)*, IEEE, Oct. 2023, pp. 141–146. doi: 10.1109/ICSET59111.2023.10295141.

[9] M. C. Pascoe, S. E. Hetrick, and A. G. Parker, "The impact of stress on students in secondary school and higher education," *Int J Adolesc Youth*, vol. 25, no. 1, pp. 104–112, Dec. 2020, doi: 10.1080/02673843.2019.1596823.

[10] A. L. A. Ramos and M. A. Ballera, "An Exploration of Electroencephalogram Brain Activity Signals of Computer Science Learners in C++ Programming Using Emotiv EpocX: A Correlation and Regression Analysis," in *2022 6th International Conference on Imaging, Signal Processing and Communications (ICISPC)*, IEEE, Jul. 2022, pp. 94–101. doi: 10.1109/ICISPC57208.2022.00025.

[11] H. Patel and P. A. Koringa, "The impact of students behaviour, their approach, emotions and problem difficulty level on the performance prediction, evaluation and overall learning process during online coding activities," *ArXiv*, vol. abs/2112.14407, 2022, [Online]. Available: https://api.semanticscholar.org/CorpusID:245537525

[12] R. Zatarain Cabada, M. L. Barrón Estrada, J. M. Ríos Félix, and G. Alor Hernández, "A virtual environment for learning computer coding using gamification and emotion recognition," *Interactive Learning Environments*, vol. 28, no. 8, pp. 1048–1063, Nov. 2020, doi: 10.1080/10494820.2018.1558256.

[13] G. Tisza, P. Markopoulos, and T. Bekker, "Learning to code: interplay of attitude, emotions, and fun," *Humanit Soc Sci Commun*, vol. 10, no. 1, p. 748, Oct. 2023, doi: 10.1057/s41599-023-02235-3.

[14] K. Masood and M. A. Alghamdi, "Modeling Mental Stress Using a Deep Learning Framework," *IEEE Access*, vol. 7, pp. 68446–68454, 2019, doi: 10.1109/ACCESS.2019.2917718.

[15] D. M. Devilbiss, R. C. Spencer, and C. W. Berridge, "Stress Degrades Prefrontal Cortex Neuronal Coding of Goal-Directed Behavior," *Cerebral Cortex*, p. bhw140, May 2016, doi: 10.1093/cercor/bhw140.

[16] K. Nolan and S. Bergin, "The role of anxiety when learning to program," in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, New York, NY, USA: ACM, Nov. 2016, pp. 61–70. doi: 10.1145/2999541.2999557.

[17] H. Azemati and S. Pourbagher, "Analysis and Recognition of Factors Affecting Stress in Educational Environments (Based on Shannon Entropy)," *CEPAL Rev*, 2017, [Online]. Available: https://api.semanticscholar.org/CorpusID:149112452

[18] M. Weatherton, J. M. Rosenberg, L. Rocconi, and E. E. Schussler, "Beyond Pre-Post Surveys: Exploring Validity Evidence For The Use Of Experience Sampling Methods To Measure Student Anxiety In Introductory Biology," *bioRxiv*, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:270380615

[19] M. Klemenčič and I. Chirikov, "How Do We Know How Students Experience Higher Education? On the Use of Student Surveys," in *The European Higher Education Area*, Cham: Springer International Publishing, 2015, pp. 361–379. doi: 10.1007/978-3-319-20877-0_24.

[20] W. Ahmed, G. van der Werf, and A. Minnaert, "Emotional Experiences of Students in the Classroom," *Eur Psychol*, vol. 15, no. 2, pp. 142–151, Jan. 2010, doi: 10.1027/1016-9040/a000014.

[21] L. Goncales, K. Farias, B. da Silva, and J. Fessler, "Measuring the Cognitive Load of Software Developers: A Systematic Mapping Study," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, IEEE, May 2019, pp. 42–52. doi: 10.1109/ICPC.2019.00018.

[22] Y. S. Can, N. Chalabianloo, D. Ekiz, and C. Ersoy, "Continuous Stress Detection Using Wearable Sensors in Real Life: Algorithmic Programming Contest Case Study," *Sensors*, vol. 19, no. 8, p. 1849, Apr. 2019, doi: 10.3390/s19081849.

[23] N. Tupikina, G. Dlamini, and G. Succi, "Review: Use of EEG on Measuring Stress Levels When Painting and Programming," in *Proceedings of the 24th International Conference on Enterprise Information Systems*, SCITEPRESS - Science and Technology Publications, 2022, pp. 374–380. doi: 10.5220/0011101200003179.

[24] I. R. Galatzer-Levy, K. V. Ruggles, and Z. Chen, "Data Science in the Research Domain Criteria Era: Relevance of Machine Learning to the Study of Stress Pathology, Recovery, and Resilience," *Chronic Stress*, vol. 2, Jan. 2018, doi: 10.1177/2470547017747553.

[25] S. Goundar, A. Deb, G. Lal, and M. Naseem, "Using online student interactions to predict performance in a first-year computing science course," *Technology, Pedagogy and Education*, vol. 31, no. 4, pp. 451–469, Aug. 2022, doi: 10.1080/1475939X.2021.2021977.

[26] R.-C. Chen, C. Dewi, S.-W. Huang, and R. E. Caraka, "Selecting critical features for data classification based on machine learning methods," *J Big Data*, vol. 7, no. 1, p. 52, Dec. 2020, doi: 10.1186/s40537-020-00327-4.

[27] S. S. Shahapur *et al.*, "Decoding Minds: Estimation of Stress Level in Students using Machine Learning," *Indian J Sci Technol*, vol. 17, no. 19, pp. 2002–2012, May 2024, doi: 10.17485/IJST/v17i19.2951.

[28] , Disha Sharma et al., Disha Sharma et al., "Stress Prediction of Students using Machine Learning," *International Journal of Mechanical and Production Engineering Research and Development*, vol. 10, no. 3, pp. 5609–5620, 2020, doi: 10.24247/ijmperdjun2020534.

[29] H. Hao, T. Chen, J. Lu, J. Liu, and X. Ma, "The Research and Analysis in Decision Tree Algorithm Based on C4.5 Algorithm," in *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, IEEE, Oct. 2018, pp. 1882–1886. doi: 10.1109/IAEAC.2018.8577527.

[30] R. R. Aswathi, K. P. Kumar, and B. Ramakrishnan, "An Extended C4.5 Classification Algorithm using Mathematical Series," *Science & Technology Journal*, vol. 7, no. 2, pp. 54–59, Jul. 2019, doi: 10.22232/stj.2019.07.02.06.

[31] Y. S. Siregar, B. O. Sembiring, H. Hasdiana, A. R. Dewi, and H. Harahap, "Algortihm C4.5 in mapping the admission patterns of new Students in Engingeering Computer," *SinkrOn*, vol. 6, no. 1, pp. 80–90, Oct. 2021, doi: 10.33395/sinkron.v6i1.11154.

[32] S. Oeda and M. Chieda, "Visualization of Programming Skill Structure by Log-Data Analysis

with Decision Tree," *Procedia Comput Sci*, vol. 159, pp. 582–589, 2019, doi: 10.1016/j.procs.2019.09.213.

[33] L. I. P. Aji and A. Sunyoto, "An Implementation of C4.5 Classification Algorithm to Analyze student's Performance," in *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, IEEE, Nov. 2020, pp. 126–130. doi: 10.1109/ICOIACT50329.2020.9332088.

[34] M. S. Ahmed, "Behavioral analysis of Programming students," https://www.kaggle.com/datasets/awansaad6797/behavioral-analysis-of-programming-students.

[35] V. V. Starovoitov and Yu. I. Golub, "Data normalization in machine learning," *Informatics*, vol. 18, no. 3, pp. 83–96, Sep. 2021, doi: 10.37661/1816-0301-2021-18-3-83-96.

[36] D. Elreedy and A. F. Atiya, "A Novel Distribution Analysis for SMOTE Oversampling Method in Handling Class Imbalance," in *International Conference on Conceptual Structures*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:184482708

[37] L. Kovács and H. Ghous, "Efficiency comparison of Python and RapidMiner," *Multidiszciplináris Tudományok*, vol. 10, no. 3, pp. 212–220, 2020, doi: 10.35925/j.multi.2020.3.26.

[38] S. Marzukhi, N. Awang, S. N. Alsagoff, and H. Mohamed, "RapidMiner and Machine Learning Techniques for Classifying Aircraft Data," *J Phys Conf Ser*, vol. 1997, no. 1, p. 012012, Aug. 2021, doi: 10.1088/1742-6596/1997/1/012012.

[39] N. Baharun, N. F. M. Razi, S. Masrom, N. A. M. Yusri, and A. S. A. Rahman, "Auto Modelling for Machine Learning: A Comparison Implementation between RapidMiner and Python," *International Journal of Emerging Technology and Advanced Engineering*, vol. 12, no. 5, pp. 15–27, May 2022, doi: 10.46338/ijetae0522_03.

[40] S. Raschka, "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning," *ArXiv*, vol. abs/1811.12808, 2018, [Online]. Available: https://api.semanticscholar.org/CorpusID:49529756

[41] A. Gholamy, V. Kreinovich, and O. Kosheleva, "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:7467506

[42] D. H. Schunk, "Self-efficacy and academic motivation," *Educ Psychol*, vol. 26, no. 3–4, pp. 207–231, 1991.

[43] V. Ramalingam, D. LaBelle, and S. Wiedenbeck, "Self-efficacy and mental models in learning to program," in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY, USA: ACM, Jun. 2004, pp. 171–175. doi: 10.1145/1007996.1008042.

[44] R. Thinakaran, S. Chuprat, V. Varadarajan, M. Yussubaliyeva, and M. Maqsood, "Self-efficacy as students' motivation factor in learning programming," *Journal of Infrastructure, Policy and Development*, vol. 8, no. 16, p. 9252, Dec. 2024, doi: 10.24294/jipd9252.

[45] C.-Y. Tseng, T.-H. Cheng, and C.-H. Chang, "A Novel Approach to Boosting Programming Self-Efficacy: Issue-Based Teaching for Non-CS Undergraduates in Interdisciplinary Education," *Information*, vol. 15, no. 12, p. 820, Dec. 2024, doi: 10.3390/info15120820.

[46] D. Urhahne and L. Wijnia, "Theories of Motivation in Education: an Integrative Framework," *Educ Psychol Rev*, vol. 35, no. 2, p. 45, Jun. 2023, doi: 10.1007/s10648-023-09767-9.