I.J. Modern Education and Computer Science, **2026, 1, 1-20**
Published Online on February 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2026.01.01

Prediction of Students' Performance in Introductory Programming in Higher Education

João P. J. Pires
Coimbra Institute of Engineering, Polytechnic University of Coimbra, Portugal
Email: a21280231@isec.pt
ORCID ID: https://orcid.org/0009-0003-0058-1386

Jorge F. R. Bernardino
Coimbra Institute of Engineering, Polytechnic University of Coimbra, Portugal
Email: jorge@isec.pt
ORCID ID: https://orcid.org/0000-0001-9660-2011

Anabela J. Gomes
Coimbra Institute of Engineering, Polytechnic University of Coimbra, Portugal
Email: anabela@isec.pt
ORCID ID: https://orcid.org/0000-0001-8418-8095

Ana Rosa P. Borges*
Coimbra Institute of Engineering, Polytechnic University of Coimbra, Portugal
Email: arborges@isec.pt
ORCID ID: https://orcid.org/0000-0003-3167-8714
*Corresponding Author

Fernanda M. R. Brito R. Correia
Coimbra Institute of Engineering, Polytechnic University of Coimbra, Portugal
Email: fernanda@isec.pt
ORCID ID: https://orcid.org/0000-0002-8330-1903

Received: 25 April, 2025; Revised: 26 June, 2025; Accepted: 21 October, 2025; Published: 08 February, 2026

Abstract: Analyzing student performance in Introductory Programming courses in Higher Education is crucial for early intervention and improved academic outcomes. This study investigates the predictive potential of a Programming Cognitive Test in assessing student aptitude and forecasting success in an Introductory Programming course. Data was collected from 180 students, both freshmen and repeating students, enrolled in a Computer Engineering program. The dataset includes the Programming Cognitive test results, background variables, and final course outcomes. To identify latent patterns within the data, the K-means clustering algorithm was applied, focusing particularly on freshmen students to avoid bias from prior programming exposure. In parallel, six Machine Learning classification models were developed and evaluated to predict students' likelihood of passing the Introductory Programming course: Decision Tree, K-Nearest Neighbor, Naïve Bayes, Random Forest, Support Vector Machine, and Deep Neural Network. Among these, the Deep Neural Network model demonstrated superior performance, achieving the highest values across key metrics—Accuracy, Recall, and F1-score—effectively identifying students at risk of underperformance. These findings underscore the potential of this model in educational settings, where timely and accurate detection of struggling students can enable proactive, targeted interventions. This work contributes to the field by combining cognitive assessment with predictive modelling, offering a novel approach to forecasting programming performance. The models and methods described are adaptable for broader educational applications and may assist educators in refining teaching strategies and improving retention and success rates in programming education.

Index Terms: Introductory Programming, Machine Learning, Prediction, Programming Cognitive Tests

1. Introduction

Learning to program is not easy, and reports of this difficulty are widespread worldwide. Several studies have examined the factors that influence the learning process of students beginning to program, as well as strategies to overcome them [1]. Among the factors found in the literature to explain this problem are the ability to think abstractly [2], previous experience in programming [3,4], the effect of teaching approaches that consider students' motivational strategies, learning preferences and learning styles [5,6], mathematics preparedness [7], or problem-solving skills [8]. Introductory Programming (IP) remains a challenge for students [9-12], with a high failure rate in the first programming subject in Higher Education (HE) [13,14]. Quite a few experiments and studies have also been conducted with the goal of investigating the problem and helping to minimize it [15].

Numerous tools and strategies were developed to contribute to the solution of this problem, like the development of specialized platforms to help students in the process of learning and understanding how to program [16-19], or the use of new technologies, such as robots that were analyzed and tested to see if they help students learn to program [20]. While each has its advantages, none of them has yet solved it completely. However, there is no consensus on the most effective methods, and the role of student aptitude remains largely unexplored. This study aims to fill this gap by exploring the potential of a programming cognitive test as a predictor of student performance in IP.

Programming aptitude is not only a prerequisite but also an essential skill that serves as the foundation for a future engineer's abilities, which lays the groundwork for understanding complex algorithms, problem-solving methodologies, and the fundamentals of software development [21]. The first programming curricular unit is extremely important for aspiring computer engineers, as it serves as the foundation for their entire educational and professional path. Early intervention is crucial for addressing these challenges and improving student learning outcomes.

Predicting the skills required for success in programming is a topic that has been studied for many years and explored by several authors, yet no consensus has emerged. There is also recent research focusing on the development and validation of aptitude tests for programming [22, 23]. Most of these tests require related skills, such as mathematics and logic [24]. The IBM Programmer Aptitude Test is a well-known early test for predicting programming aptitude. This test was widely used in industry to evaluate job applicants and to assess the potential to learn to program [25].

In this study, we collected data to develop a Programming Cognitive Test (PCT) to measure students' aptitude for learning programming and predict students' performance in the IP course at HE. Two Research Questions (RQs) were specified: RQ1 - Does PCT, composed of Verbal, Reasoning, and Aptitude components, can effectively predict student performance in an IP course, using machine learning models? and RQ2 - Which ML models have the best results?

To address these research questions, the PCT was designed using questions in three cognitive categories—Verbal, Reasoning, and Aptitude—alongside additional background information deemed relevant to student performance. Based on this dataset, it was necessary to adopt a systematic approach to ensure that the modeling process was both rigorous and aligned with the study's objectives. A structured approach was used to justify the selection of algorithms for model development. To identify patterns and relationships in the data and predict students' performance in IP courses on HE, K-means clustering and Machine Learning (ML) algorithms were used. This detailed analysis will allow for a deeper appreciation of the results obtained and the conclusions drawn.

This paper presents the machine learning (ML) models used to analyze data and the metrics adopted to evaluate their performance, emphasizing the importance of selecting appropriate algorithms to extract insights and make accurate predictions in programming education. Each chosen algorithm is described in detail, including its operation, development, training, and testing, followed by a comparison of results to identify the most efficient model. The study also discusses the methods used to ensure model robustness and generalizability. Tools such as Jupyter Notebook [26] and the Python programming language, recognized for their effectiveness in data science, were employed. The primary goal is to develop a predictive tool that enables educators to identify students at risk of failure, allowing for timely pedagogical adjustments and potential adaptation to other subjects or courses.

The remainder of this paper is structured as follows. Section 2 summarizes the literature review concerning programming aptitude tests; Section 3 describes the methodology used, including the sample and test description; Section 4 presents the results analysis; Section 5 discusses the results; Section 6 presents some pedagogical recommendations, and finally, Section 7 presents the conclusions and future research.

2. Literature Review

Over the decades, numerous efforts have been made to develop aptitude tests capable of predicting success in programming, particularly in the early stages of a computer science degree program. One of the earliest and most well-known tools was the IBM Programmer Aptitude Test (PAT) [27], which assessed numerical patterns, visual analogies, and arithmetic skills. Despite its popularity in industry during the 1960s, studies revealed only a weak correlation between test scores and actual job performance. In response, Wolfe (1971) [28] designed a new battery of tests focused on the ability to follow complex procedural instructions, aiming to address earlier limitations. Over time, programming aptitude tests have continued to evolve. Parallel to this, general scholastic aptitude tests like the SAT [29] have been used in educational contexts, although their predictive power is limited. Critics argue these tests explain only

a fraction of academic success. Another example is PAAT and I-PAT (International Programming Aptitude Test) [30], which assesses a broader range of cognitive abilities, including logical reasoning, specification interpretation, attention to detail, and symbolic reasoning. In 2006, Dehnadi and Bornat [31] claimed they had found a way to identify students who would fail to learn programming, based on a test administered to 60 students.

Despite this ongoing interest, no single test has achieved widespread consensus or adoption. One reason lies in the complex, multifaceted nature of programming, which involves not only logic and mathematical skills but also abstraction, creativity, and problem-solving under uncertainty. As highlighted by Quille and colleagues [23], even recent attempts, such as the PreSS (Predicting Student Success) test, require careful contextualization and adaptation to diverse educational environments. Similarly, Harris [22] emphasized the difficulty in designing tests that can generalize across populations, given variations in students' backgrounds, learning styles, and prior experiences. The lack of a standardized or widely accepted programming aptitude test underlines the need for continued exploration in this area, incorporating cognitive, motivational, and experiential dimensions. Overall, while numerous programming aptitude tests have been developed over time, no single test has gained universal acceptance, and their effectiveness in reliably predicting programming performance remains debated.

3. Methodology

We collected data from students' responses to the PCT, designed to assess their aptitude for learning introductory programming, as well as their final grades in an IP course in higher education. The data was gathered from first-year Computer Engineering (CE) students enrolled in the 2023/2024 academic year at the Coimbra Institute of Engineering (ISEC), part of the Polytechnic University of Coimbra (IPC), Portugal.

3.1. Sample and Programming Cognitive Test Description

During the first week of classes in the first semester of the 2023/2024 academic year, which began in September 2023, a test was administered to assess students' programming aptitude in the context of the IP course. A total of 217 students answered the test. This course has a total grade of 20 marks, including grades from two tests and a final exam at the end of the semester. The assessment for this course is as follows: 1st test (5 marks), 2nd test (6 marks), and a final exam (9 marks), for a total of 20 marks. For students to take the exam, they need at least 25% in each test and at least 7 practical class attendances in 15 classes. In the exam, they also need at least 25%. If the student obtains a grade of 9.5 or higher after all these requirements, the student has passed. Students who didn't get a final grade because they didn't fulfil the criteria were removed from the sample. Thus, of the 217 participants, only 180 completed all three evaluation methods and received a final grade. The final dataset comprises 180 instances, each corresponding to a student's test answers and course grades.

Our test consisted of three independent parts: Verbal question [32], Reasoning questions [33], and Aptitude questions [34], for testing programming aptitude. Each part assesses, from our perspective, a crucial skill or competency needed for success in programming. Verbal questions typically assess comprehension, language processing, and the ability to interpret written material. Reasoning questions focus on logical analysis, problem-solving, and decision-making. Aptitude questions are often used to measure abstract thinking, spatial reasoning, and quantitative skills. We chose to include these categories in the PCT based on their relevance to programming success, as supported by prior research on cognitive and problem-solving skills essential in this field. From our perspective, each part assesses a core competency required for effective programming. In practice, programmers must be able to read and understand complex documentation, articulate problems clearly, and collaborate with team members. They also interpret problem descriptions, write meaningful code comments, and process language-based error messages. These tasks rely heavily on language comprehension and clarity of thought, making verbal ability a relevant cognitive factor. Although often perceived as secondary to mathematical reasoning, verbal skills are closely tied to abstraction and problem decomposition in programming [2]. Similarly, reasoning ability underpins the capacity to break down problems into smaller components, analyze patterns, and troubleshoot efficiently. Logical reasoning, sequential thinking, and pattern recognition directly support programming constructs such as control flow, conditionals, and algorithmic development. Aptitude, as assessed through numeric or abstract tasks, reflects the learner's ability to grasp new concepts, visualize structures, and apply computational logic. Skills like mental rotation, quantitative reasoning, and abstract symbol manipulation are often associated with algorithm design and data structure comprehension [7, 31] (e.g., Bundy, 1984; Pennington, 1987).

The PCT questions were taken from online tests and are based primarily on those used by IBM in the recruitment process. However, the Verbal question part was considered complex and misaligned with our intentions. Therefore, the researchers involved used triangulation, opting for a single Verbal question, which proved to be effective [35] and has already been used in several studies to assess individual qualities, such as language skills necessary to interpret a programming problem description and logical skills included in problem-solving ability [36]. We acknowledge that the verbal component represented by only one item in our test could limit the reliability of the verbal sub-score; however, the selected item was adapted from a validated question previously used in educational aptitude assessments [35]. Future iterations of the PCT will aim to expand this component and subject it to psychometric analysis to align it with programming performance better.

The test consisted of 21 multiple-choice questions. The questions belong to the three categories mentioned: 1 Verbal question, 10 Reasoning questions, and 10 aptitude questions. The Verbal question presents a logical problem; the Reasoning questions are based on sequences of numbers or letters that must be completed, and the aptitude questions present some mathematical problems. In a simplified way, a score of 1 point was assigned to each question, so the test totaled 21 points. As the cognitive test consisted of three categories of questions, it was decided to create 3 new variables with the final grades for each category: Verbal, Reasoning, and Aptitude. With these variables established, the next step was to explore how students' cognitive performance related to their academic success in the course.

The developed models were based on classification algorithms that aimed to determine whether students would pass the IP course. To do this, the variable with the student's final grade in the course, with values between 0 and 20, has been transformed into one with values 0 or 1, for when students fail (0) and pass (1).

The features used were the answers to the 21 questions, the final grades in each of the three categories (Verbal, Reasoning and Aptitude), the cognitive test grade, the final grade of the course, and the category of the students (FS – Freshmen students or RS – Repeating Students), for a total of 27 features. The dataset built through the cognitive test is described in Table 1.

Table 1. Dataset

Sample size	Number of Features	Freshmen sample size	Repeaters sample size
180	27	148	32

As a result of the initial data pre-processing, the state of the dataset that would be used in the process of training, testing and validating the models has the features, a Boolean freshman (1 if it is a FS and 0 if is a RS), P1-P till P21-P questions (1 if correct and 0 if wrong), Cat_Verbal grade (grade between 0 and 1), Cat_Reasoning grade (grade between 0 and 10), Cat_Aptitude (grade between 0 and 10), P_total grade (grade between 0 and 21) and a Boolean Approved (1 if approved and 0 if not).

3.2. Machine Learning Algorithm for Cluster Analysis

Cluster analysis was used to identify patterns and relationships within data by grouping data in natural ways and revealing structures that may not be immediately apparent in other methods. This can lead to new insights and a better understanding of the data's fundamental structure. The cluster analysis to be carried out will focus solely on the 148 FS, since the main aim of this analyzis is to analyze patterns in the students' data when they have their first contact with IP. Including RS might introduce some bias into the data, as they may already have prior programming experience or at least some previous exposure to it.

The cluster analysis was divided into three stages, each consisting of analyzing clusters for one of the question categories (Verbal, Reasoning and Aptitude), using a hierarchical method, the K-means algorithm, one of the most popular partitioning methods due to its simplicity and efficiency. The elbow rule, attributed to Thorndike [37], was used to define the number of clusters. The average silhouette coefficient [38] was also calculated to assess whether instances were correctly assigned to each cluster.

3.3. Other Machine Learning Algorithms for Prediction

In this study, six widely used supervised classification algorithms were selected to predict student performance in the IP course and were identified in [39] (in alphabetic order): Decision Tree (DT) [40], K-Nearest Neighbor (KNN) [41], Naïve Bayes (NB) [42], Random Forest (RF) [43], Support Vector Machine (SVM) [44] and Deep Neural Network (DNN) [45]. The rationale for this selection was grounded in the suitability of each algorithm for different data distributions, levels of interpretability, and complexity, as well as its ability to model patterns in educational datasets. The research reported in [46] showed that DNN performed well.

DT was included for its high interpretability, making it suitable for educational settings where understanding the decision-making process is as essential as prediction Accuracy. DTs can handle both categorical and numerical data, and their tree-like structure provides transparency in feature importance and rule extraction. RF extends DTs by creating an ensemble of trees, improving generalization through bagging and random feature selection. RF is robust to noise and overfitting and provides measures of feature importance, yielding educational insights into which factors most strongly affect student success. KNN was selected for its simplicity and non-parametric nature, which makes no assumptions about the underlying data distribution. KNN is often effective in low-dimensional, well-separated problems and serves as a strong baseline in classification tasks. Its performance in identifying outlier or at-risk cases (e.g., students likely to fail) is particularly relevant in educational datasets. NB, based on Bayes' theorem with the assumption of feature independence, is computationally efficient and often surprisingly accurate, particularly in text classification and educational contexts. Despite its strong independence assumption, NB serves as a robust baseline, especially in imbalanced datasets, where Precision and Specificity are crucial. SVM was chosen for its robustness in high-dimensional spaces, given the large number of input features derived from cognitive tests. SVM is effective in handling non-linearly separable data through the kernel trick and is known for its strong generalization performance, even in relatively small datasets. DNN was included for its ability to learn complex, non-linear patterns via hierarchical representations. DNNs are especially effective when relationships among features are not explicit or are highly

interdependent, as is likely the case in cognitive performance prediction. Though less interpretable, DNNs offer state-of-the-art performance and are becoming increasingly relevant in educational data mining.

The combination of these six algorithms allowed for a comprehensive comparative analysis of different modeling paradigms—from transparent rule-based models (DT, RF), to statistical and probabilistic classifiers (NB), distance-based methods (KNN), margin-based classifiers (SVM), and deep learning techniques (DNN). This diversity enables a thorough evaluation of model performance and applicability in predicting programming success and identifying students at risk.

This section provides an overview of the workflow used to predict students' success in an IP course on HE. The workflow goes from dividing up the data, selecting the features, oversampling and doing the normalization of the data, to describing the adjustments of the hyperparameters and describing the DNN architecture used, as well as specifying what evaluation metrics were applied to the models.

3.3.1. Data Splitting

To ensure that ML models can generalize effectively to new data, the dataset was split into two subsets: a training set and a test set. Using the "train_test_split" function from the "Scikit-learn" library, 70% of the data was used for training and 30% for testing. The values for this split were chosen based on an initial test that used the following dataset splits: 70/30 and 80/20. Two of the six algorithms were used for the evaluation process (RF and SVM). RF and SVM have contrasting characteristics, so by testing both, it was possible to obtain a representative analysis of the impact of the split on the performances of algorithms with different natures, providing a more informed basis for choosing the final split. The models with the dataset split into 70% training and 30% testing obtained slightly better Accuracy results, as can be seen in Table 2. This dataset split was performed randomly, with the "random_state" parameter set to 42. This ensures that the random splitting process is reproducible: each time the dataset is split with this value, the training and testing sets will be the same. Setting "random_state=42" ensures consistent results across runs and algorithms, enabling a reliable comparison of models applied to the same data split. After the split, the train set had 126 instances, while the test set had 54.

Table 2. Dataset splits performance analysis

Metric	RF 70/30	RF 80/20	SVM 70/30	SVM 80/20
Accuracy	85%	75%	87%	75%

3.3.2. Feature Selection

One stage in model development is feature selection, which involves selecting the features to use for training and testing. Initially, the chosen features were all those obtained from the cognitive test data: student category (FS or RS), each question grade, Verbal category grade, Reasoning category grade, Aptitude category grade, and test grade. However, this first selection totaled 27 features, making it very difficult for the model to learn the data and make accurate predictions. This was demonstrated during RF and SVM testing, where both models classified all instances as likely to be approved (1). This result indicates that the models were unable to learn from the data; instead, they adjusted to the noise and failed to capture significant patterns, leading to overfitting.

To overcome this problem, a decision was made to reduce the number of features. In the feature selection process, two methods were compared: SelectKBest and Categorical Principal Component Analysis (CATPCA). The "SelectKBest" method operates as a univariate filter, ranking features based solely on their individual relationships with the target variable using statistical tests; it does not consider potential feature interactions nor the multivariate structure of the data. [47]. With "SelectKBest" features are selected based on their individual characteristics without considering the ML algorithm used [48]. This was the main reason for choosing this method over others, because the selected features must be the same for every algorithm.

The other method applied focused on using information obtained from the statistical analysis, specifically the Categorical Principal Components Analysis (CATPCA) analysis [49]. CATPCA captures non-linear relationships and interdependencies between features, which are particularly relevant in the context of cognitive and behavioral educational assessments. Another advantage of CATPCA lies in its interpretability: the transformation provides insight into the variance explained by each principal component and facilitates visual inspection of feature relationships, which aligns with the educational focus of this study. Empirical testing confirmed the theoretical advantages.

The features selected by both methods were:

- SelectKBest: P4_P, P6_P, P8_P, P15_P, P18_P, Cat_Verbal, Cat_Resoning, Cat_Aptitude, P_total.
- CATPCA: Freshmen, P9_P, P10_P, P17_P, P18_P, Cat_Verbal, Cat_Resoning, Cat_Aptitude, P_total.

As shown in Table 3, the feature set selected using CATPCA yielded higher Accuracy in both RF and SVM models compared to SelectKBest, with accuracies of 85% and 87% respectively, versus 81% and 85% for SelectKBest. Therefore, CATPCA was chosen for the final feature selection not only because it aligns with the dataset's categorical

nature and reveals deeper structure, but also because it yielded slightly better cross-validation results, confirming its practical value.

Table 3. Feature selection performance comparison: SelectKBest/CATPCA

Metric	RF SelectKBest	SVM SelectKBest	RF CATPCA	SVM CATPCA
Accuracy	81%	85%	85%	87%

Therefore, the final list of features chosen to train and test the models was obtained from the CATPCA analysis and is described in Table 3.

3.3.3. Data Augmentation and Normalization

While analyzing the data, a clear imbalance was detected. Of the 180 instances, more than 80% correspond to students who passed the IP curricular unit, leaving just under 20% who failed the course. The class referring to approved students is the majority class in the dataset, which causes an imbalance in the classes that can lead the model to be biased toward predicting the majority class, potentially resulting in poor Recall and misclassification of the minority class, in this case, the students most in need of early intervention. Therefore, for the model to be able to generalize, it was necessary to balance the training set. To address this issue, the "RandomOverSampler" oversampling technique from the "imbalanced-learn" library was employed [50]. This technique involves generating new instances of the minority class by replicating existing instances within that class. In this case, it used the existing instances in the minority class (failed students). It repeated them until it had the same number of cases as the majority class (approved students). So, after splitting the data, the oversampling technique was applied only to the training set, leaving the test set untouched. While this helped balance class representation during model training and improved Recall for the minority class, it may also increase the risk of overfitting due to duplicate instances. Therefore, cross-validation was essential to assess whether models generalized well despite the synthetic class balance. Additionally, the evaluation metrics selected (Recall, Specificity, and F1-score) were chosen to assess how well each model handled the imbalance. Even with oversampling, the imbalance can still affect model confidence calibration, threshold Sensitivity, and real-world deployment performance, where unseen data may remain skewed. Future work should consider other techniques such as Synthetic Minority Oversampling Technique (SMOTE), cost-sensitive learning, or ensemble methods tuned for imbalanced data, as well as analyzing calibration curves to assess prediction confidence.

The data was also normalized using "StandardScaler" [51] from "Scikit-learn". This technique will transform the data so that its distribution has a mean value of 0 and a standard deviation of 1, ensuring that all features are on the same scale. This normalization is an important step that helps improve the model's performance, especially for algorithms that are sensitive to the scale of the data, such as KNN and SVM.

3.3.4. Hyperparameter Adjustment

In the process of developing and testing the models, an essential stage is parameter fine-tuning. To do so, the "Scikit-learn" tool "GridSearchCV" [52] was used. This method allows hyperparameters to be tuned, enabling the identification of the parameter combination that yields the best model performance according to a specific metric, in this case, Accuracy. This process was repeated for every algorithm applied, except for the DNN.

In Table 4 the hyperparameters that best suit each algorithm to get the best results based on the Accuracy metric are described. This description only shows the selected hyperparameters with non-default values. The missing hyperparameters were used with their default values.

Table 4. Hyperparameters used

Algorithm	Criterion used and description	Default
DT	'log_loss': Defines the metric for evaluating the quality of a split. Penalizes based on the probability associated with each class, encouraging greater purity in the nodes	Metric 'gini', which is an alternative metric, but is less sensitive to probabilities
	Max_features = 'sqrt': Defines the maximum number of features considered in each split, here being the square root of the total number of features, a common practice to avoid overfitting in more complex trees	None (all features are considered)
KNN	N_neighbors = 10: Indicates the number of neighbors considered to determine the class of a new instance. A value of 10 tends to smooth the decision threshold and is more robust to noise	Value 5, where a higher number of neighbors can help to avoid fluctuations caused by atypical values
NB		All hyperparameters have the default values
RF	Max_features = 'log2': Defines the maximum number of features to be considered in each split as the base 2 logarithm of the total number of features. This reduces variance and improves generalization	None (all features are considered)
SVM		All hyperparameters have the default values

3.3.5. DNN architecture

Many tests were conducted across different configurations to define the DNN architecture. In each run, something was changed, whether it was the number of layers, the number of neurons, activation functions or even the addition and removal of special layers such as Batch Normalization and Dropout. In the end, it came down to the following final architecture:

1. Dense input layer with 128 neurons and ReLU activation function.
2. Normalization layer ("BatchNormalization")
3. Dense layer with 64 neurons and ReLU activation function.
4. Dense layer with 8 neurons and ReLU activation function.
5. Dense output layer with 1 neuron, as this is a binary classification problem, with Sigmoid activation function.

3.3.6. Evaluation Metrics

In the performance prediction phase, the metrics used were Accuracy, Precision, Recall, F1-Score and Specificity. Accuracy measures the total percentage of correct predictions. In student performance prediction problems, high accuracy indicates that the model can correctly predict students' status in most cases. Precision focuses on the quality of the positive predictions, measuring the proportion of positive predictions that belong to that class. Precision is crucial when False Positives (FP) are penalized, as in the case of predicting success where, in fact, the student would fail, leading to incorrect expectations. Recall measures the model's ability to identify all instances of the positive class correctly. In a student performance context, good Recall is important to ensure the model does not miss students at risk of failing. The F1-Score combines Precision and Recall into a single metric to assess their balance. It is beneficial in cases of unbalanced classes. Finally, Specificity measures the model's ability to correctly identify the negative class, which is important to ensure it does not misclassify low-performing students as high-performing. This metric was not part of the initial selection, but it was chosen because it enables evaluation of an essential project task (correctly predicting students at risk).

Although Accuracy is the most widely reported and intuitive metric for assessing overall model performance, its limitations in imbalanced classification tasks are well recognized. In this study, the dataset is skewed, with a higher proportion of students passing the course than failing. In such cases, a high Accuracy score may reflect bias toward the majority class, making it an insufficient sole evaluation criterion. To address this, we complemented Accuracy with Precision, Recall, F1-score, and Specificity. Among these, the F1-score is particularly relevant, as it balances Precision and Recall, providing a better representation of the model's performance in the minority class (students at risk). In educational contexts, where identifying underperforming students is critical, the F1-score provides a more informative perspective than Accuracy alone. However, we retained Accuracy as a reference for comparability with existing literature and to provide a general performance overview. Therefore, the reported results should be interpreted in combination rather than relying solely on Accuracy, especially when evaluating models for practical deployment in early intervention systems.

4. Results Analysis

This chapter presents a detailed analysis of the results obtained from applying K-means clustering and the other ML models discussed previously.

4.1. Cluster analysis

The cluster analysis using the K-means algorithm was divided into three stages, each consisting of analyzing clusters for one of the question categories (Verbal, Reasoning and Aptitude).

- **Verbal Category**

A hierarchical method was first applied to perform the cluster analysis, in which the number of clusters was determined using the elbow rule and the coefficients. The expression calculates the number of clusters as the difference between the dataset size (148) and the step (144), where the step is the stage at which the elbow is identified; therefore, 148 minus 144 equals 4 clusters. Next, the K-means algorithm was applied in the cluster analysis. Each instance of the dataset was distributed in a cluster. Table 5 presents the descriptive statistics for the four defined clusters. Here, each cluster's minimum, maximum, mean, and standard deviation are based on the two selected variables: Course grade and Verbal category grade.

The fact that the Verbal category had only one question makes it difficult to extract insights from the cluster's formation. Analyzing, it's possible to see:

Cluster 1 – Students with average course and average Verbal category grades.
Cluster 2 – Students with excellent course grades and average Verbal category grades.
Cluster 3 – Students with good course grades and average Verbal category grades.
Cluster 4 – Students with low course and low Verbal category grades.

The cluster silhouette coefficient was calculated to assess whether instances were correctly assigned to each cluster. The silhouette coefficient measures the similarity of an object to its cluster compared to other clusters. A value close to 1 indicates that the object has a good match with its cluster and a poor match with neighboring clusters. The average silhouette coefficient achieved for the four clusters was 0.510, indicating that, on average, the clusters are relatively well-defined and distinct from one another.

Table 5. Cluster analysis for the Verbal category

Variable	Cluster	Minimum	Maximum	Mean	Std. Deviation
Course Grade	1	9	12	10.85	0.970
	2	17	20	17.65	0.885
	3	13	16	14.37	1.121
	4	5	8	6.94	0.998
Verbal Category Grade	1	0	1	0.51	0.505
	2	0	1	0.52	0.511
	3	0	1	0.5	0.505
	4	0	1	0.31	0.479

- **Reasoning Category**

Based on the elbow rule and using the expression that calculates the number of clusters as the difference between the number of instances and the step, the number of clusters was 5 because 148 minus 143 totals 5. Table 6 presents the descriptive statistics of the five clusters. Here, each cluster's minimum, maximum, average, and standard deviation are based on the two selection variables: Course grade and Reasoning category grade.

Table 6. Cluster analysis for the Reasoning category

Variable	Cluster	Minimum	Maximum	Average	Std. Deviation
Course Grade	1	15	20	16.63	1.384
	2	7	11	9.49	1.292
	3	11	17	14.04	1.786
	4	5	9	6.78	1.394
	5	11	14	12.46	1.047
Reasoning Category Grade	1	5	10	7.5	1.351
	2	5	10	7.06	1.371
	3	1	6	4.04	1.427
	4	2	5	3.44	1.014
	5	6	10	8.18	0.997

Analyzing Table 6 it is possible to gather some insights about the cluster's formation:
Cluster 1 – Students with good course and good Reasoning category grades.
Cluster 2 – Students with low course grades and good Reasoning category grades.
Cluster 3 – Students with good course grades and low Reasoning category grades.
Cluster 4 – Students with low course and low Reasoning category grades.
Cluster 5 – Students with average course grades and good test grades.
The average silhouette coefficient achieved for the five clusters was 0.385, indicating that, on average, the clusters are relatively well-defined and distinct from one another.

- **Aptitude Category**

Based on the elbow rule and using the expression that calculates the number of clusters as the difference between the number of instances and the step, the number of clusters considered was 4 because 148 minus 144 equals 4. Table 7 presents the descriptive statistics of the four clusters. Here, each cluster's minimum, maximum, average, and standard deviation are based on the two selection variables: Course grade and Aptitude category grade.

Table 7. Cluster analysis for the Aptitude category

Variable	Cluster	Minimum	Maximum	Average	Std. Deviation
Course Grade	1	15	20	16.71	1.382
	2	12	18	14.50	1.606
	3	10	14	11.81	1.197
	4	5	11	8.56	1.761
Aptitude Category Grade	1	4	9	6	1.181
	2	1	5	3.31	1.176
	3	4	9	6.08	1.127
	4	1	7	4.18	1.381

Analyzing Table 7 it is possible to gather some insights about the cluster's formation:
Cluster 1 – Students with good course and good Aptitude category grades.
Cluster 2 – Students with good course grades and low Aptitude category grades.

Cluster 3 – Students with average course grades and good Aptitude category grades.

Cluster 4 – Students with low course and low Aptitude category grades.

The average silhouette coefficient across the four clusters was 0.365, indicating that the clusters are relatively well-defined and distinct from one another.

4.2. Other Machine Learning algorithms

When applying ML techniques to the models discussed previously in this study, the aim is to provide a comprehensive assessment of their performance, compare their performance, and discuss the insights derived from these results in the context of the problem under study. After pre-processing, the training set with oversampling had 220 instances (110 for each class), while the test set, in its original state, had 54 instances (47 for approved students and 7 for failed students).

4.2.1. K-fold Cross-Validation

To ensure a robust, unbiased assessment of the models' performance, the K-fold Cross-Validation method was used before the final training and testing stages. This method consists of dividing the data set into k subsets of similar size, called folds. In each iteration, one of these folds is reserved for validation, while the remaining k-1 are used to train the model. This process is repeated k times, ensuring that each instance of the dataset is used for both training and validation, providing a more stable and representative assessment of the model's performance.

This process also allows hyperparameters to be adjusted more precisely, as repeated validation on different subsets of the dataset provides a clearer view of how the model might behave on unseen data. This makes it possible to optimize models before the final evaluation. There is no rule defining the exact value of K. Tests were carried out with K = 5 and K = 10. While the average results across folds were relatively stable for most algorithms, minor variance was observed in individual fold outcomes, particularly for RF, which motivated the final choice of K=5. This selection provided a better trade-off between computational cost and variance stability. It also aligns with standard practice for small-to-moderate datasets like ours, where higher values of K may lead to increased variance due to smaller validation sets per fold. Importantly, cross-validation served as a key step in model tuning and evaluation, helping us mitigate overfitting, especially for models such as DT and RF, which are sensitive to training data noise. For DNN and SVM, cross-validation also provided a more balanced view of model behavior beyond the final test set. In a dataset with high imbalance and limited minority-class examples, cross-validation ensured that every instance was used in both the training and validation phases, thereby enhancing the robustness of model comparisons and providing greater confidence in selecting the best-performing approach. Table 8 shows the results of the cross-validation in all models.

Table 8. K-fold Cross-Validation results, with K=5

Algorithms	Accuracy	Precision	Recall	F1-score
DT	78%	80%	78%	79%
DNN	88%	92%	84%	87%
KNN	84%	81%	84%	80%
NB	81%	81%	81%	81%
RF	81%	78%	81%	78%
SVM	87%	89%	87%	81%

The results obtained by applying various ML algorithms to predict students' performance in an IP course were interpreted and compared with findings reported in the existing literature. The performance metrics used were Accuracy, Precision, Recall, F1-score and Specificity. The results obtained are shown in Table 9.

Table 9. Models' performance

Algorithms	Accuracy	Precision	Recall	F1-score	Specificity
DT	80%	89%	87%	88%	29%
DNN	91%	94%	96%	95%	57%
KNN	69%	97%	66%	78%	86%
NB	69%	94%	68%	79%	71%
RF	85%	90%	94%	92%	29%
SVM	87%	93%	91%	92%	57%

DNN had the best overall performance, with the highest Accuracy (91%), Recall (96%), and F1-score (95%), as well as high Precision (94%) and a significantly high Specificity (57%). This indicates that DNN is particularly effective at predicting students' performance in IP courses. RF also showed good results, with high Accuracy (85%), Precision (90%), Recall (94%) and F1-score (92%). However, its Specificity was low (29%); the same problem occurred with DT, suggesting difficulties in identifying true negatives, indicating that these models fail an important requirement: the ability to identify students who may not have the aptitude for programming. SVM is also a good, stable choice, as it performed well with Accuracy (87%), Precision (93%), Recall (91%), and F1-score (92%), with Specificity equal to that of the DNN (57%). Surprisingly, KNN achieved the highest Specificity (86%), indicating that it is best at identifying students with reduced aptitude for programming and who may have difficulties learning programming

throughout the course. However, the low Accuracy (69%) and Recall (66%) suggest that it may not be the most suitable for general performance balancing. NB also had high Specificity (71%) and good Precision (94%), but KNN had moderate Accuracy (69%) and Recall (68%), resulting in a lower F1-score (79%).

4.2.2. Confusion Matrix

Visualizing the results is an important step in analyzing ML models, as it makes it easier to both interpret and understand the model's performance. Next, the confusion matrices for each model are displayed, providing a clear view of how each model performs in terms of correct and incorrect classifications. A confusion matrix is a performance-evaluation tool for classification models that provides a detailed breakdown of correct and inaccurate predictions. The confusion matrix has four important results: True Positives (TP): positive cases correctly classified as positive; True Negatives (TN): negative cases correctly classified as negative; False Positives (FP): negative cases incorrectly classified as positive; and False Negatives (FN): positive cases incorrectly classified as negative.

Table 10 helps to visually summarize the possible results of a binary classification.

Table 10. Confusion matrix description

Reality / Prediction	Negative (Prediction)	Positive (Prediction)
Negative (Reality)	TN	FP
Positive (Reality)	FN	TP

Fig. 1 shows a dashboard of the confusion matrices generated by all six algorithms: DNN, SVM, RF, DT, NB, and KNN. The confusion matrices are displayed from best to worst performance (most correct predictions). Note that in the following confusion matrices, the class with value "1" corresponds to approved students, and the class with value "0" corresponds to failed students.

In the matrix of the DNN algorithm, it is seen that this model performs well in predicting the majority class (1), with 45 instances correctly classified as positive. However, the model struggles to predict the minority class (0), resulting in only 4 correct predictions and 3 instances incorrectly classified as the positive class.

The SVM confusion matrix is also presented. The SVM model achieves the second-best performance among the six algorithms, striking a good balance between TP and TN predictions, correctly classifying almost all TP instances while failing in only three TN instances.

In the confusion matrix of the RF algorithm, the main problem with this model is expressed. This model is difficult to identify with TN correctly. Of the 7 TN instances, RF correctly classified only 2. However, the RF model correctly predicted one more TP than SVM.

The DT model's confusion matrix reveals the central problem: its difficulty in correctly classifying TN. It is also possible to understand one reason for the DT model's low Specificity values: the small number of TN instances, which has a significant impact even if only a few are misclassified.

In the NB model's confusion matrix, although the TN (5 out of 7) are correctly classified, the main problem is the classification of TP, which explains the model's low Accuracy. The NB model misclassified a large portion of instances in the majority class (approved students).

For KNN, the confusion matrix reveals both its main strengths and weaknesses. The model correctly classified 6 instances of the negative class (class 0) and had only 1 FP, indicating a strong ability to identify the negative class correctly. However, the model underperformed in determining the positive class (class 1), incorrectly classifying 16 positive instances as negative. These results suggest that KNN tends to favor the negative class, resulting in a high rate of FN.

The analysis of the confusion matrix for all six models tested supports the conclusion regarding the five metrics results. DNN and SVM stand out as the most effective, maintaining a good balance between the predictions of the approved student's class and the failed student's class. KNN and NB perform very well at predicting students at risk, but fall short when classifying majority-class instances. DT and RF achieve good results in predicting which students are approved, but they misclassify almost every at-risk student.

A deeper analysis of the confusion matrices reveals key differences in how each model handles FP and FN, both of which have critical implications in the educational context. FP occurs when a student predicted to pass (approved) fails. These cases are particularly concerning because they may not receive the necessary academic support or interventions, potentially increasing the risk of failure. For instance, the RF and DT models had high FP counts, reflected in low Specificity scores (29%), suggesting that these models often overestimate student readiness. FN occurs when a student who was predicted to fail can pass. While less immediately harmful, excessive FNs could result in unwarranted interventions or mislabeling capable students, possibly affecting their confidence or access to challenging material. Models such as KNN and NB exhibited high FN rates, which correlate with their lower Recall and Accuracy. On the other hand, the DNN and SVM models maintained a better balance, showing both low FP and FN counts, and thus offering more reliable predictions across both classes. Notably, DNN correctly identified 45 TP and only misclassified 3 negatives (FP), achieving a high Recall (96%) and moderate Specificity (57%), indicating its relative strength in identifying students at risk while minimizing misclassification. From a pedagogical perspective, minimizing false negatives is especially critical because students at risk of failure should be prioritized for early intervention. Therefore, models with higher Recall, like DNN and SVM, are more appropriate in this domain. At the same time, minimizing FP

is important for avoiding complacency or missed opportunities for support. Thus, striking the right balance (as reflected in the F1-score and Specificity) is essential in selecting models for academic prediction systems.

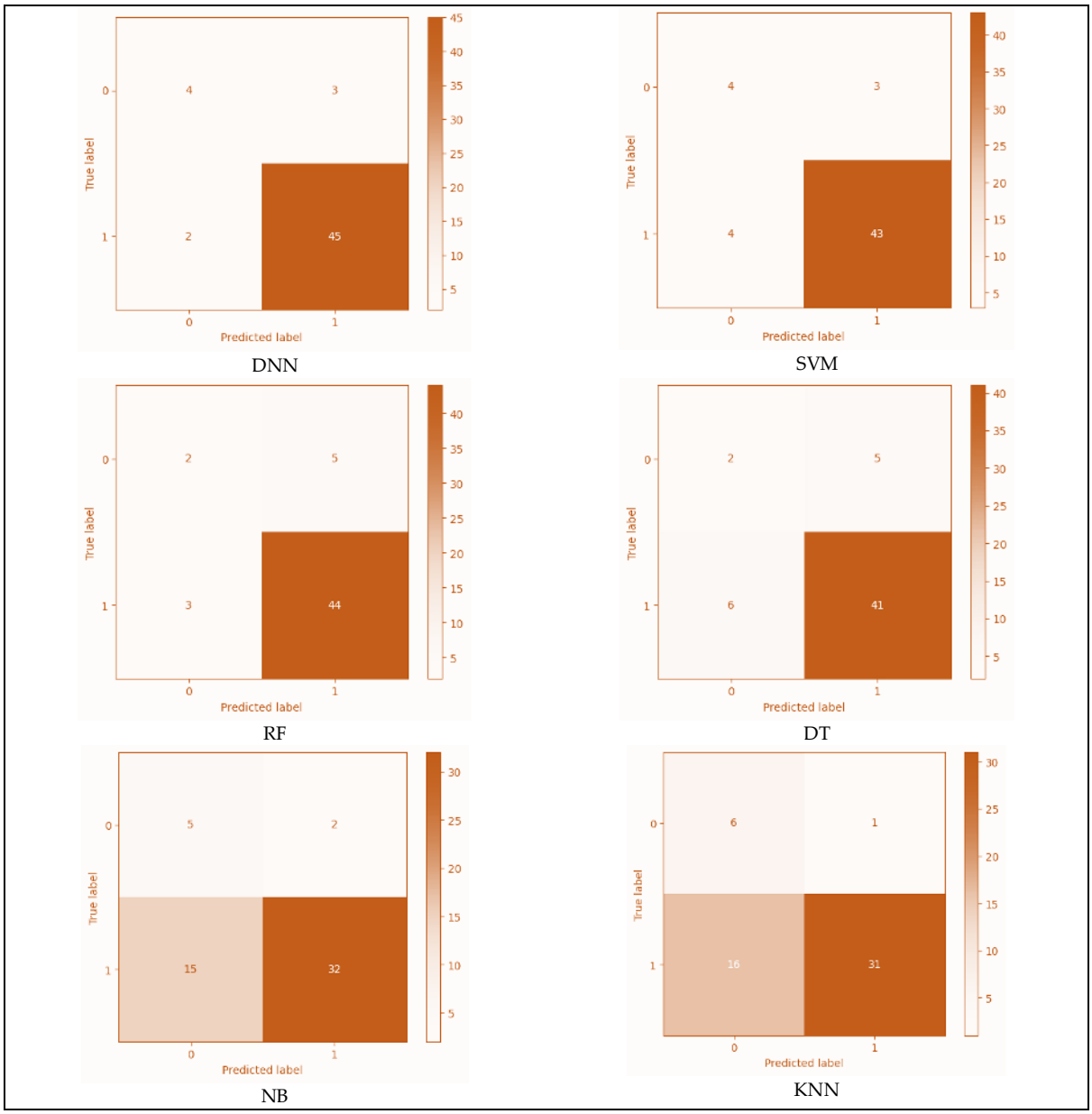

Fig. 1. Confusion matrices dashboard.

4.2.3. Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) curve is presented for each algorithm to understand better the model's ability to distinguish between the two classes. The ROC curve is used to evaluate the performance of binary classification models, with the visualization of the model's ability to discriminate between classes. A model with a curve closer to the top-left corner indicates better performance, as it shows a higher TP rate and a lower FP rate. Fig. 2 shows a dashboard with the ROC curves used to assess the performance of the models, namely KNN, RF, SVM, NB, DNN and DT. The area under the curve (AUC) quantifies the overall performance of the model: the closer it is to 1, the better the performance, indicating a high rate of correct classification. An AUC value of 0.5 or lower suggests that the model is producing random predictions. In the dashboard, the ROC curves are displayed from best to worst performance (highest to lowest AUC).

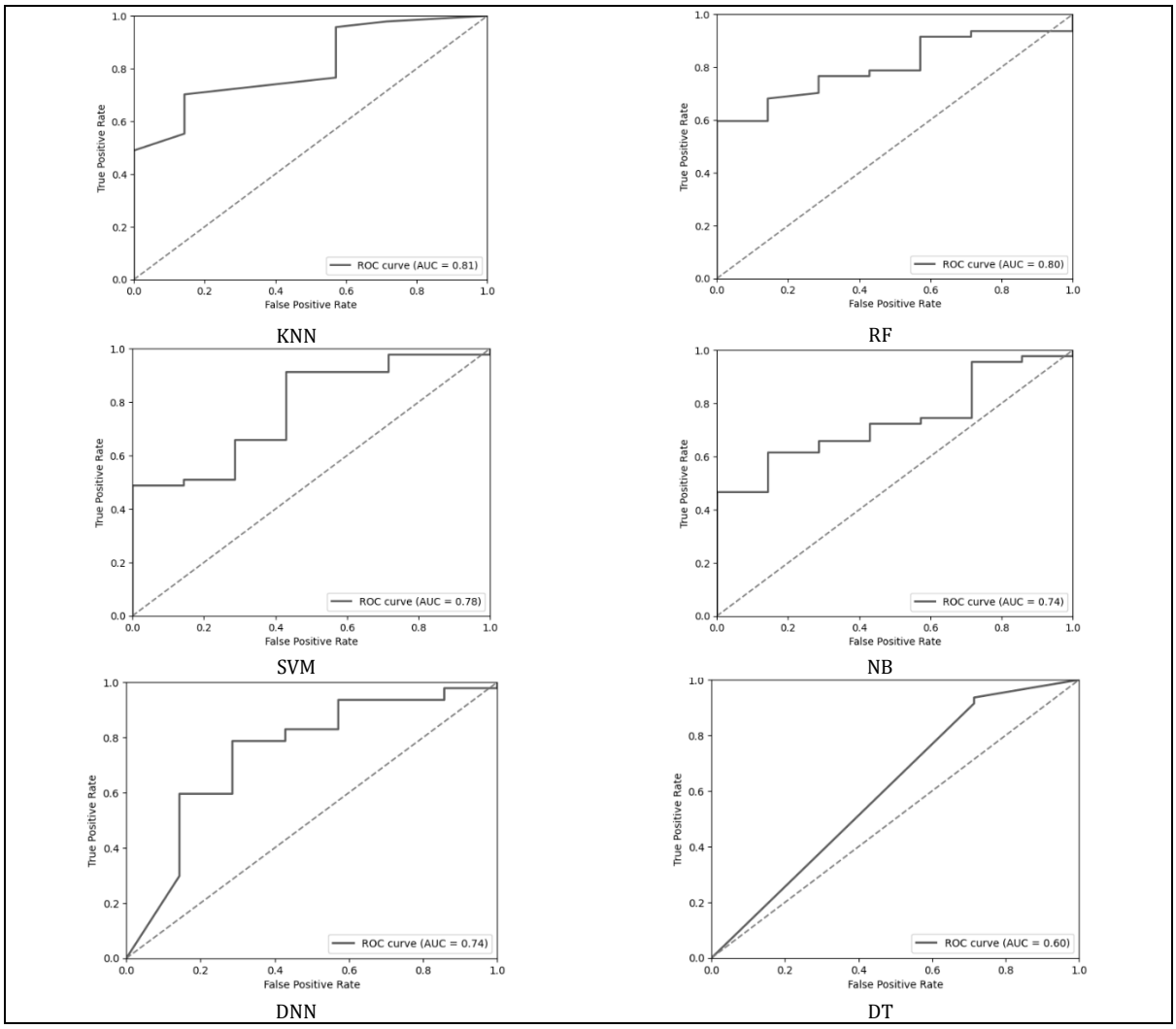

Fig. 2. ROC curves dashboard.

KNN achieved an AUC of 0.81, demonstrating very good performance in distinguishing the classes. This value also indicates that, among the implemented algorithms, KNN is best at distinguishing and predicting between different classes. With an AUC of 0.80, RF demonstrates a strong ability to differentiate between classes. This is an interesting observation since, when analyzing the results, RF was one of the models with the lowest Specificity value, indicating that this model could have a certain difficulty in predicting the students who are unlikely to pass the course. The AUC value obtained may suggest that the sample observed by RF in the test phase does not reflect the model's true predictive capabilities. SVM achieved the third-highest AUC of 0.78, indicating good performance in distinguishing between classes. In the ROC Curve for the NB, the AUC is 0.74, indicating moderate ability to differentiate between classes. The ROC curve of the DNN model shows performance similar to that of the NB model. The AUC value of 0.74, the same as NB, indicates that the DNN model has a moderate ability to discriminate between classes. For the DT model, an AUC of 0.60 was obtained, indicating poor performance in distinguishing between classes. The ROC curve indicates the same performance issue: it is far from the top-left corner, with poor performance values.

Analysis of the ROC curves and AUC values for the six models tested allows a comparative assessment of each approach's performance in predicting student performance. The KNN model performed best with an AUC of 0.81, followed by RF (AUC = 0.80) and SVM (AUC = 0.78). These results suggest that KNN and RF have a relatively strong ability to distinguish between classes, making them suitable for this specific task. This cloud suggests that models based on neighborhoods and multiple decision trees have significant advantages, possibly due to their ability to better deal with the complexities and variations in the dataset. The DNN and NB models achieved AUCs of 0.74, indicating moderate performance, though lower than the best models. Although they still have a good discrimination capacity, their results suggest that they may not be as effective as KNN and RF for this sample. DT had the lowest AUC (0.60), indicating limited classification capacity and possibly a tendency to overfit the training data.

5. Discussion

The project aims to identify students' aptitude for programming. In this context, an important factor is the models' ability to classify negative instances correctly, that is, accurately determine whether a student is at risk of failing, enabling early interventions to improve their chances of success. The results obtained provide insights into relevant information.

5.1. Quality of the Results

The six selected algorithms not only differ in performance but also in their underlying principles and suitability for different aspects of the classification task. Each method brings unique strengths that are beneficial in the context of educational data. For instance, interpretability is crucial when educators seek actionable insights, making models like DT and RF attractive. In contrast, high-dimensional modeling capabilities (as in SVM) and complex pattern learning (as in DNN) are important for accuracy in multifaceted cognitive datasets. These differences explain the performance variations across metrics like Accuracy, Recall, and Specificity observed in Section 4. Algorithms such as DNN, SVM and KNN stand out for different reasons. However, their performance results fall slightly short of the desired values.

DNN has demonstrated exceptional performance in modelling complex, non-linear relationships between input features. This depth is useful in problems where the relationships between features are not linear, as is the case here. Its hierarchical structure allows it to learn increasingly abstract representations of the data, enabling it to capture intricate patterns that other models may miss. This capability is particularly advantageous for predicting student performance, as it often involves multiple interconnected factors. These may be some of the characteristics that have led this model to achieve the most optimistic results. For instance, [46] has shown that DNNs can effectively model the impact of coding exercises, learning styles, and academic performance on future outcomes. DT and RF are powerful tools for capturing non-linear interactions between features. However, their susceptibility to overfitting, especially with limited data, can adversely affect performance, particularly metrics such as Specificity. As the study in [53] highlights, this limitation can lead to models that are overly sensitive to noise in the data, resulting in poor generalization. While KNN achieved the highest Specificity, its performance relies heavily on the appropriate definition of distance metrics in the feature space. In the context of student performance prediction, where features may vary in scale and importance, defining suitable distance metrics can be challenging. As the authors in [54] argue, suboptimal distance metrics can hinder KNN's ability to classify instances accurately. NB assumes feature independence, which can be a restrictive assumption in many real-world scenarios, including student performance prediction. Although it is fast and efficient, this assumption can limit its ability to capture more complex relationships. Finally, the SVM has proved capable of classification. In the case of student performance prediction, where the underlying relationships may be intricate and non-linear, SVM may struggle to capture the full complexity of the data, leading to slightly lower performance than DNN. **S**pecificity is a crucial metric because it reflects the model's ability to identify students who need intervention correctly. After analyzing the results, it is possible to see that apart from KNN and NB, the algorithms do not perform well in the Specificity metric.

A limitation of this study is the relatively small sample size of 180 students, which, while meaningful in an educational setting, can restrict the generalizability of ML models. It is well established that larger datasets typically improve the robustness, stability, and external validity of predictive models. In our case, the sample reflects the entire population of students enrolled in a single course offered at a HE Institution. Despite this limitation, several methodological strategies were applied to enhance model generalizability, including 5-fold cross-validation, feature selection, and oversampling techniques to balance class distributions. These approaches helped mitigate some overfitting risks and ensured that each instance contributed to both the training and validation phases. Moreover, the study's primary goal was not only predictive performance but also the exploratory analysis of cognitive and aptitude indicators for student success, which can yield actionable insights even with smaller samples. That said, caution is warranted in extending the findings to other student populations or institutional contexts. Future research should aim to validate these models on larger, more diverse datasets, potentially involving multi-institutional collaborations or longitudinal data across academic years. Such efforts would increase confidence in model transferability and improve the statistical power of analyses. The reason for this is probably that the dataset is imbalanced. The oversampling technique used doesn't create new instances; it basically just replicates existing ones. The training set ended up with about 220 instances (110 for each class). Still, because the original dataset has only 30 instances in the minority class, the model may not have enough information to learn it. So, due to the large variability that can occur in the data, it can be difficult for the models to learn and generalize. Therefore, when testing the models on the test set, the models tend to be somewhat biased towards the majority class. The class imbalance affected several models. For example, while models like DT and RF achieved high Accuracy and Precision, they struggled with low Specificity, indicating poor performance in the minority class. This highlights the risk of bias introduced by imbalance, even after applying oversampling. The results underscore the need to interpret metrics like Recall and Specificity alongside Accuracy, particularly in educational contexts where identifying at-risk students is a priority.

Apart from the perhaps possible lack of ability to generalize, it can also happen that the minority class is under-represented in the test set, which can affect metrics such as Specificity. Since the minority class is much smaller than

the majority class, even a single incorrect prediction can significantly affect the final metric value, as each instance (TN) carries considerable weight.

The use of 5-fold cross-validation was particularly relevant in reducing potential bias and variance in model evaluation, given the dataset's small size and class imbalance. By ensuring that each data point contributed to both training and validation, the results reported in Section 4 are more likely to reflect real-world performance. The consistency of performance across folds, especially for SVM and DNN, supports their generalizability. Conversely, the slight fold-to-fold variability observed in RF and KNN highlights the importance of cross-validation in identifying models that may be more sensitive to sampling variations. Overall, cross-validation played a central role in validating model robustness, providing a more comprehensive assessment than a single train-test split.

While the evaluation relied on multiple performance metrics (Accuracy, Precision, Recall, F1-score, and Specificity), no statistical significance tests were conducted to formally assess whether the observed differences between models were statistically significant. The observed variations, such as DNN outperforming other models on most metrics, were consistent across both test data and cross-validation, but without significance testing, these results should be interpreted as indicative rather than conclusive evidence of superiority. Incorporating statistical significance tests would strengthen confidence in selecting the best-performing model, and it is an area for future enhancement.

One limitation of this study was the tendency of some tree-based models, particularly DT and RF, to overfit, especially when applied to a relatively small, imbalanced dataset. This was evident in the confusion matrix results, where these models achieved high TP counts but performed poorly in the minority class (failing students), reflected in their low Specificity scores (29%). Such results suggest that the models may have memorized training data patterns without generalizing well to new, unseen data. DT-based algorithms are inherently prone to overfitting, particularly when the number of training samples is limited or when models are not sufficiently regularized. To mitigate this, we applied cross-validation, used feature selection to reduce model complexity, and performed hyperparameter tuning. Despite these efforts, the risk of overfitting remains, and it should be considered when interpreting the predictive results of these models. Nevertheless, DT and RF were included in the study not only for their predictive capacity but also for their interpretability, which is essential in educational contexts. These models provide insights into which variables contribute most to predictions and can help educators understand potential pedagogical patterns in student performance. For example, feature importance scores and decision paths can inform instructional design and early interventions.

In future research, further strategies to address overfitting will be explored, such as ensemble methods with pruning, regularization techniques, and using larger, more diverse datasets. Additionally, leveraging explainable AI (XAI) techniques with more generalizable models (e.g., SVM or DNN) could offer a better balance between predictive robustness and interpretability.

An important consideration in applying ML in educational contexts is model interpretability. Predictive models must not only be accurate but also understandable to educators and decision-makers, who must translate predictions into actionable pedagogical strategies. In this study, interpretability was partially addressed by including inherently interpretable models such as Decision Tree (DT) and Random Forest (RF), which provide feature importance rankings and clear decision structures. Additionally, the CATPCA-based feature selection identified which features explained the most variance in the dataset. Preliminary analyses revealed that variables such as Cat_Reasoning, Cat_Aptitude, and P_total were consistently selected across models and were associated with student success. These results align with the theoretical foundation that cognitive reasoning and problem-solving ability are key predictors of programming performance. However, deeper insights into the contributions of each feature, particularly in complex models such as DNN and SVM, should be further explored in future work.

5.2. Comparison with Literature

This subsection presents a comparison of the results from the prediction models developed with those identified in the literature review conducted in [39]. Each comparison will be based on the results from all evaluation metrics applied. The DNN developed is not included in this section because the single paper that uses DNN only reports RMSE.

5.2.1. Decision Tree

Table 11 presents the results of this study in comparison to the results of the existing literature identified in [39].

Table 11. DT results comparison

Origin	Sample	Accuracy	Precision	Recall	F1-score	Specificity
This study	18	80%	89%	87%	88%	29%
Paper [55]	244	99.18%	100%	98.7%	-	-
Paper [56]	490	85.10%	86.57%	84.89%	85.72%	-

The results of our study show moderate performance compared to those in the literature. Although precision and recall are competitive, especially in comparison with paper [56], the overall Accuracy of our model falls short of expectations. The sample size of 180 records in this study may have affected the model's generalizability, as studies with larger samples, such as [56] with 490 records, have shown slightly superior results, likely due to greater data representativeness. As for Specificity, as is the case with almost all the literature analyzed, it is not very widely used, so it is not possible to compare. The discrepancy with paper [55], which shows exceptionally high results, can probably be

explained by the fact that in that study, the dataset used has information from the entire semester to predict the final grade (with grades from tests and practical assignments), thus having a greater quantity of assessment components already carried out making it easier to predict the final grade.

5.2.2. K-Nearest Neighbors

Table 12 presents the results of this study in comparison to the results of the existing literature identified in [39].

Table 12. KNN results comparison

Origin	Sample	Accuracy	Precision	Recall	F1-score	Specificity
This study	180	69%	97%	66%	78%	86%
Paper [55]	244	98.36%	99.3%	100%	-	-
Paper [56]	490	84.28%	85.82%	83.46%	84.62%	-

The results of our study show mixed performance compared with the literature. While Precision and Specificity are high, overall Accuracy and Recall need significant improvement. The Accuracy and Recall of KNN in our study were significantly lower than those reported in the papers [55,56]. On the other hand, the Precision in our study is comparable to that in one paper [55] and higher than that in another paper [56]. The absence of Specificity values in the studies [55-59] limits direct comparison of this indicator, but the high Specificity of this study model suggests that the model performs well at avoiding FP. Regarding sample size, both studies used larger samples, which may have contributed to their superior performance, as larger samples tend to reduce bias and increase data representativeness.

5.2.3. Naïve Bayes

Table 13 presents the results of this study in comparison to the results of the existing literature identified in [39].

Table 13. NB results comparison

Origin	Sample	Accuracy	Precision	Recall	F1-score	Specificity
This study	180	69%	94%	68%	79%	71%
Paper [57]	300	84.46%	-	-	-	-
Paper [58]	50	84%	-	-	83%	-
Paper [55]	244	98.36%	99.3%	100%	-	-
Paper [59]	86	86%	-	-	-	-
Paper [56]	490	84.28%	85.82%	83.46%	84.62%	-

The results of our study using the Naive Bayes algorithm showed mixed performance compared to the literature. Precision is a strong point, but overall, Accuracy and Recall need improvement. The Accuracy and Recall obtained by our model were the lowest among the other papers. On the other hand, the precision was the second-best, behind the paper [55], which already showed very good results across all algorithms (perhaps even too good). The Specificity in our study indicates that the model is reasonably effective in correctly identifying students who are unlikely to pass. However, this metric was not reported in the papers [55-59]. Compared to other studies, the dataset used to train and test NB in our study was among the smallest, which may, to some extent, explain the poor results, especially in Accuracy and Recall. Studies such as [58, 59] used even smaller samples and achieved better accuracy. However, because of the lack of more diverse metrics, it's not easy to draw clear conclusions about the model's true quality.

5.2.4. Random Forest

Table 14 presents the results of this study in comparison to the results of the existing literature identified in [39].

Table 14. RF results comparison

Origin	Sample	Accuracy	Precision	Recall	F1-score	Specificity
This study	180	85%	90%	94%	92%	29%
Paper [60]	289	60%	-	-	-	77%
Paper [59]	86	90%	-	-	-	-
Paper [56]	490	87.55%	86.06%	88.6%	87.31%	-

The results obtained in our study with the RF algorithm showed good overall performance, with high values for Accuracy, Precision, Recall, and F1-score, the last three of which were slightly better than those reported in the paper [56]. The low Specificity of RF in this study, compared to the reasonably good Specificity in the paper [60], is a significant concern, as it indicates that the model is not efficient at identifying students who are unlikely to pass, resulting in many FP. Even with a smaller dataset, the RF model achieved better results than those reported in other studies. When comparing the results of the paper [56] with this study's model, the sample size of 490 records (almost three times the size of our sample) didn't have much of an impact on the results, since the model in [56] only outperformed our study in Accuracy by approximately 3%.

5.2.5. Support Vector Machine

Table 15 presents the results of this study in comparison to the results of the existing literature identified in [39].

The results of our study using the SVM algorithm showed good overall performance, with high values for Precision, Recall, and F1-score. However, although the Accuracy achieved good values, it could be improved to align with results in the literature, such as those reported in paper [55]. As with RF, the small dataset used to train and test SVM was sufficient to produce results comparable to, and in some cases better than, those of SVM models in similar studies. The good performance of our SVM suggests it could be a helpful tool, but it also points to areas for improvement to achieve even better results.

Table 15. SVM results comparison

Origin	Sample	Accuracy	Precision	Recall	F1-score	Specificity
This study	180	87%	93%	91%	92%	57%
Paper [55]	244	99.18%	100%	98.7%	-	-
Paper [56]	490	88.77%	89.79%	88%	88.88%	-

In [39], it was suggested that greater emphasis should be placed on deep learning techniques, as DNN emerged as the top-performing model. Additionally, SVM was identified as the best algorithm for predicting student performance in IP courses based on a literature review and average Accuracy.

While the literature review highlighted SVM's potential, our findings suggest that DNN offers a more robust and comprehensive approach. Although SVM showed balanced performance and outperformed RF in Specificity, DNN's superior overall performance, particularly in Accuracy, makes it a more promising candidate for future research and practical applications in student performance prediction.

5.3. Pedagogical Implications

The results of this study indicate that the PCT, composed of Verbal, Reasoning, and Aptitude components, can be an effective tool for identifying students who are more or less likely to succeed in an introductory programming course, thereby addressing RQ1. This early identification enables instructors to adapt teaching strategies from the start of the semester, offering differentiated support tailored to each student's cognitive profile.

The findings of this study offer several actionable insights for improving teaching strategies in IP courses. First and foremost, the predictive models developed, particularly the DNN and SVM classifiers, can be deployed as early-warning systems to help identify students at risk of failure based on their performance in the PCT. These predictions, made before or in the early stages of the course, can enable teachers to offer targeted interventions, such as tutoring sessions, mentoring, or scaffolded assignments. Secondly, the analysis of feature contributions and test components (especially Reasoning and Aptitude-based items) suggests that certain cognitive abilities are strongly correlated with programming success. This insight has clear instructional implications: curriculum designers and teachers can incorporate exercises that foster abstract reasoning, pattern recognition, and logical problem-solving early in the course. This can be done through non-coding challenges, such as logic puzzles or flowchart design, which develop programming-relevant thinking without relying on syntax. Third, the models can be used as a diagnostic tool, not only to predict success but also to understand student profiles. By examining common traits among misclassified students (e.g., high aptitude but low performance), instructors may uncover underlying challenges, such as test anxiety, language barriers, or lack of prior exposure to algorithmic thinking. Finally, academic programs may consider integrating the PCT into student onboarding processes, using it to guide course placement, recommend preparatory modules, or adapt pacing strategies for different learner profiles. Over time, aggregating prediction outcomes across cohorts could inform broader curriculum redesign and support data-driven decision-making in computer science education.

Furthermore, the use of machine learning models such as Random Forest and Logistic Regression—which showed the best performance in predicting academic success (RQ2)—demonstrates the potential of integrating analytical tools into pedagogical planning. These models can be used to flag at-risk students, allowing for targeted interventions such as personalized tutoring, additional learning resources, or workload adjustments.

6. Conclusions

The high failure and dropout rates associated with IP courses remain a problem worldwide. Therefore, the project presented in this paper is particularly significant for educational institutions seeking to identify early students at risk of failure in the first programming subject and tailor interventions to enhance student success. This paper aims to predict students' performance in IP courses in HE institutions, using K-means clustering to identify patterns and relationships within the data by grouping it in natural ways, revealing those that may not be immediately apparent, and by developing and comparing ML models. Two RQs were formulated and answered: RQ1 - Does PCT, composed of Verbal, Reasoning, and Aptitude components, can effectively predict student performance in an IP course, using machine learning models? and RQ2 - Which ML models have the best results? Regarding RQ1, the results showed that the Programming Cognitive Test (PCT), composed of Verbal, Reasoning, and Aptitude components, is a valid predictor of students' performance in the IP course. As for RQ2, among the machine learning models tested, Random Forest and

Logistic Regression achieved the best predictive Accuracy. These findings reinforce the value of combining cognitive assessment with advanced analytical techniques to support data-informed pedagogical decisions, promoting early and tailored interventions that meet students' needs."

This study used data from a PCT test designed to assess essential skills, including Reasoning, Aptitude, and Verbal competencies, among students in an IP curricular unit.

This study successfully developed and evaluated six different ML models to predict students' performance in IP courses. To better understand the results, some dashboards with the confusion matrices and ROC curves of the models were developed. Among the models tested, the DNN achieved the highest prediction performance (Accuracy, Recall, and F1-score) and was effective in identifying students at risk of underperformance. These results indicate the potential of DNN models in educational applications, where the accurate identification of students in need of intervention can have a transformative impact on academic success rates.

This work provides a detailed comparison of traditional and deep learning techniques, clarifying which approaches are most effective in educational data contexts. The insights gained can be used to integrate predictive models into educational platforms, enabling institutions to proactively support students who are likely to struggle, potentially increasing pass rates and reducing dropout rates in programming courses. Teachers could also adjust their teaching strategies based on insights derived from these predictions, creating a more responsive and supportive learning environment.

Although this study provides promising results, some considerations must be made. The size of the dataset, although statistically significant, can be improved, and the imbalanced data could have affected the generalization of the model. Although the literature review supported the selection of ML algorithms, it may have excluded some interesting algorithms. Testing new algorithms or advanced neural network architectures could also produce performance improvements. Future studies could investigate the use of Convolutional Neural Networks for tabular data, more sophisticated ensemble techniques, or even other emerging deep learning algorithms. In addition, hyperparameter optimization other than grid search could be tested and potentially improve the performance of existing models.

One limitation of this study is the lack of formal statistical testing when comparing model performance. While differences in metrics were consistent and aligned with known model strengths, significance tests would provide stronger empirical support for the conclusion that certain models, such as DNN or SVM, are superior in this context. Future research should incorporate these tests to confirm performance differences and ensure robust model selection. The dataset used contains data from a cognitive test, as well as academic information such as students' grades and student type (Freshman students - FS or Repeating Students - RS). Future research could extend this study by incorporating additional data and new features, such as past curricular information about the students (grades in other curricular units, course entry grade averages, etc.), the students' motivation, the students' previous programming experience, or learning styles, to improve the reliability of the model. These small pieces of information can provide additional insights and increase the Accuracy of predictive models. Although 180 students offer valuable insights for initial modeling and exploration analysis, larger and more diverse datasets are needed to confirm the findings and support broader generalizations. Future work should include replications across Institutions and explore strategies for collecting larger-scale, longitudinal data to improve generalizability and model robustness.

While this study focused on model performance, future work should emphasize model transparency and feature attribution, particularly for complex classifiers such as DNN and SVM. Tools such as SHapley Additive exPlanations (SHAP) or Local Interpretable Model-agnostic Explanations (LIME) can be employed to understand better how individual features (e.g., reasoning scores or specific test questions) contribute to model decisions. This would support the development of more trustworthy, explainable AI tools in education, enabling instructors to act more confidently on predictions and design targeted interventions grounded in transparent evidence.

In addition to performance metrics, this work offers practical insights for educators, highlighting how predictive modelling can inform instructional strategies, early support systems, and curriculum enhancements tailored to students' cognitive profiles.

References

[1] A. Gomes and A. J. Mendes, "Learning to program-difficulties and solutions," in *International Conference on Engineering Education–ICEE*, 2007, pp. 1–5.

[2] J. Bennedsen and M. E. Caspersen, "Abstraction ability as an indicator of success for learning object-oriented programming?," *ACM SIGCSE Bulletin*, vol. 38, no. 2, 2006, doi: 10.1145/1138403.1138430.

[3] P. Byrne and G. Lyons, "The effect of student attributes on success in programming," in *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, 2001, pp. 49–52.

[4] A. Luxton-Reilly *et al.*, "Introductory programming: a systematic literature review," in *Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education*, 2018, pp. 55–106.

[5] A. Gomes and A. Mendes, "A study on student's characteristics and programming learning," *ED-MEDIA 2008--World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 2008.

[6] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *Proceedings - Frontiers in Education Conference, FIE*, 2014. doi: 10.1109/FIE.2014.7044086.

[7] E. Tomai and C. F. Reilly, "The impact of math preparedness on introductory programming (CS1) success (abstract only)," 2014. doi: 10.1145/2538862.2544292.

[8] A. Lishinski, A. Yadav, R. Enbody, and J. Good, "The influence of problem solving abilities on students' performance on different assessment tasks in CS1," *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pp. 329–334, Feb. 2016, doi: 10.1145/2839509.2844596.

[9] T. Jenkins, "On the difficulty of learning to program," in *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Citeseer, 2002, pp. 53–58.

[10] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, pp. 1–24, 2017.

[11] J. Bennedsen, M. E. Caspersen, and M. Kölling, *Reflections on the teaching of programming: methods and implementations*, vol. 4821. Springer, 2008.

[12] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bulletin*, vol. 37, no. 3, pp. 14–18, Sep. 2005, doi: 10.1145/1151954.1067453.

[13] C. Watson and F. W. B. Li, "Failure rates in introductory programming revisited," in *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, 2014. doi: 10.1145/2591708.2591749.

[14] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming - 12 years later," *ACM Inroads*, vol. 10, no. 2, 2019, doi: 10.1145/3324888.

[15] S. R. Sobral, "Strategies on Teaching Introducing to Programming in Higher Education," in *Advances in Intelligent Systems and Computing*, 2021. doi: 10.1007/978-3-030-72660-7_14.

[16] P. C. Tavares, P. R. Henriques, and E. F. Gomes, "A computer platform to increase motivation in programming students-PEP," in *CSEDU 2017 - Proceedings of the 9th International Conference on Computer Supported Education*, 2017. doi: 10.5220/0006287402840291.

[17] E. Verdú, L. M. Regueras, M. J. Verdú, J. P. Leal, J. P. De Castro, and R. Queirós, "A distributed system for learning programming on-line," *Comput Educ*, vol. 58, no. 1, 2012, doi: 10.1016/j.compedu.2011.08.015.

[18] A. Ferreira, A. Gomes, and A. J. Mendes, "SICAS2: Interactive Tool to Support Programming Learning," in *SIIE 2022 - 24th International Symposium on Computers in Education*, 2022. doi: 10.1109/SIIE56031.2022.9982323.

[19] A. Moreno, N. Myller, and E. Sutinen, "JeCo, a collaborative learning tool for programming," in *Proceedings - 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 2004. doi: 10.1109/VLHCC.2004.33.

[20] M. M. McGill, "Learning to program with personal robots: Influences on student motivation," *ACM Transactions on Computing Education*, vol. 12, no. 1, 2012, doi: 10.1145/2133797.2133801.

[21] R. Scherer, F. Siddiq, and B. S. Viveros, "The cognitive benefits of learning computer programming: A meta-analysis of transfer effects," *J Educ Psychol*, vol. 111, no. 5, 2019, doi: 10.1037/edu0000314.

[22] J. Harris, "Testing programming aptitude in introductory programming courses," *J. Comput. Sci. Coll.*, vol. 30, no. 2, pp. 149–156, Dec. 2014.

[23] K. Quille, S. Nam Liao, E. Costelloe, K. Nolan, A. Mooney, and K. Shah, "PreSS: Predicting Student Success Early in CS1. A Pilot International Replication and Generalization Study," in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, in ITiCSE '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 54–60. doi: 10.1145/3502718.3524755.

[24] J. Ringenberg, M. Lapp, A. Bansal, and P. Shah, "The programming performance prophecies: Predicting student achievement in a first-year introductory programming course," *Computers in Education Journal*, vol. 22, no. 2, 2012, doi: 10.18260/1-2--18930.

[25] L. J. Mazlack, "Identifying potential to acquire programming skill," *Commun ACM*, vol. 23, no. 1, pp. 14–17, Jan. 1980, doi: 10.1145/358808.358811.

[26] "Project Jupyter | Home." Accessed: Jul. 18, 2024. [Online]. Available: https://jupyter.org/

[27] R. E. Mayer, J. L. Dyck, and W. Vilberg, "Learning to program and learning to think: What's the connection?," *Commun ACM*, vol. 29, no. 7, pp. 605–610, Jul. 1986, doi: 10.1145/6138.6142.

[28] J. M. Wolfe, "Wolfe programming aptitude test (school edition)," in *Proceedings of the Ninth Annual SIGCPR Conference*, in SIGCPR '71. New York, NY, USA: Association for Computing Machinery, 1971, pp. 180–185. doi: 10.1145/800159.805105.

[29] D. F. Butcher and W. A. Muth, "Predicting performance in an introductory computer science course," *Commun. ACM*, vol. 28, no. 3, pp. 263–268, Mar. 1985, doi: 10.1145/3166.3167.

[30] B. Winrow, "The Walden programmer analyst aptitude test," *Dr. Dobb's Journal*, Sep. 1999, Accessed: Apr. 16, 2025. [Online]. Available: https://drdobbs.com/the-walden-programmer-analyst-aptitude-t/184411169?queryText=winrow

[31] S. Dehnadi, "Testing Programming Aptitude.," in *18th Workshop of the Psychology of Programming Interest Group*, Brighton, UK, Sep. 2006, pp. 22–37. Accessed: Apr. 16, 2025. [Online]. Available: https://www.ppig.org/files/2006-PPIG-18th-dehnadi.pdf

[32] "IBM Verbal Questions | Verbal Ability Questions For IBM." Accessed: Apr. 01, 2024. [Online]. Available: https://cpt.hitbullseye.com/IBM-Verbal-Questions.php

[33] "IBM Reasoning Questions | ReasoningTest For IBM." Accessed: Apr. 01, 2024. [Online]. Available: https://cpt.hitbullseye.com/IBM-Reasoning-Test.php

[34] "IBM Aptitude Questions | Aptitude Test For IBM." Accessed: Apr. 01, 2024. [Online]. Available: https://cpt.hitbullseye.com/IBM-Aptitude-Questions.php

[35] M. Ragni, I. Kola, and P. N. Johnson-Laird, "The Wason Selection Task: A Meta-Analysis," in *CogSci 2017 - Proceedings of the 39th Annual Meeting of the Cognitive Science Society: Computational Foundations of Cognition*, 2017.

[36] J. S. B. T. Evans, "Deciding before you think: Relevance and reasoning in the selection task," *British Journal of Psychology*, vol. 87, no. 2, 1996, doi: 10.1111/j.2044-8295.1996.tb02587.x.

[37] R. L. Thorndike, "Who Belongs in the Family?," *Psychometrika*, vol. 18, no. 4, pp. 267–276, Dec. 1953, doi: 10.1007/BF02289263.

[38] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J Comput Appl Math*, vol. 20, pp. 53–65, 1987, doi: https://doi.org/10.1016/0377-0427(87)90125-7.

[39] J. P. J. Pires, F. Brito Correia, A. Gomes, A. R. Borges, and J. Bernardino, "Predicting Student Performance in Introductory Programming Courses," *Computers*, vol. 13, no. 9, 2024, doi: 10.3390/computers13090219.

[40] "Decision Tree - GeeksforGeeks." Accessed: Jul. 05, 2024. [Online]. Available: https://www.geeksforgeeks.org/decision-tree/
[41] "K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks." Accessed: Jul. 06, 2024. [Online]. Available: https://www.geeksforgeeks.org/k-nearest-neighbours/
[42] "Naive Bayes Classifiers - GeeksforGeeks." Accessed: Jul. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/naive-bayes-classifiers/
[43] "Random Forest Algorithm in Machine Learning - GeeksforGeeks." Accessed: Jul. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/
[44] "Support Vector Machine (SVM) Algorithm - GeeksforGeeks." Accessed: Jul. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-algorithm/
[45] "What Is Deep Learning? | IBM." Accessed: Jul. 06, 2024. [Online]. Available: https://www.ibm.com/topics/deep-learning
[46] G. Shen, S. Yang, Z. Huang, Y. Yu, and X. Li, "The prediction of programming performance using student profiles," *Educ Inf Technol (Dordr)*, vol. 28, no. 1, 2023, doi: 10.1007/s10639-022-11146-w.
[47] "SelectKBest — scikit-learn 1.5.2 documentation." Accessed: Dec. 06, 2024. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.feature_selection.SelectKBest.html
[48] "Feature Selection in Python with Scikit-Learn - GeeksforGeeks." Accessed: Dec. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/feature-selection-in-python-with-scikit-learn/
[49] "CATPCA - IBM Documentation." Accessed: Apr. 16, 2025. [Online]. Available: https://www.ibm.com/docs/en/spss-statistics/saas?topic=reference-catpca
[50] "RandomOverSampler — Version 0.12.4." Accessed: Dec. 06, 2024. [Online]. Available: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html
[51] "StandardScaler — scikit-learn 1.5.2 documentation." Accessed: Dec. 06, 2024. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.StandardScaler.html
[52] "GridSearchCV — scikit-learn 1.5.2 documentation." Accessed: Dec. 06, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[53] L. Breiman, "Random forests," *Mach Learn*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324/METRICS.
[54] H. A. Abu Alfeilat *et al.*, "Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review," *Big Data*, vol. 7, no. 4, pp. 221–248, Dec. 2019, doi: 10.1089/BIG.2018.0175.
[55] M. M. Jamjoom, E. A. Alabdulkareem, M. Hadjouni, F. K. Karim, and M. A. Qarh, "Early prediction for at-risk students in an introductory programming course based on student self-efficacy," *Informatica (Slovenia)*, vol. 45, no. 6, 2021, doi: 10.31449/INF.V45I6.3528.
[56] Sivasakthi M and Pandiyan M, "Machine Learning Algorithms to Predict Students' Programming Performance: A comparative Study," *Journal of University of Shanghai for Science and Technology*, vol. 24, 2022.
[57] M. Sivasakthi, "Classification and prediction based data mining algorithms to predict students' introductory programming performance," in *Proceedings of the International Conference on Inventive Computing and Informatics, ICICI 2017*, 2018. doi: 10.1109/ICICI.2017.8365371.
[58] I. Khan, A. Al Sadiri, A. R. Ahmad, and N. Jabeur, "Tracking student performance in introductory programming by means of machine learning," in *2019 4th MEC International Conference on Big Data and Smart City, ICBDSC 2019*, 2019. doi: 10.1109/ICBDSC.2019.8645608.
[59] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen, "Exploring machine learning methods to automatically identify students in need of assistance," in *ICER 2015 - Proceedings of the 2015 ACM Conference on International Computing Education Research*, 2015. doi: 10.1145/2787622.2787717.
[60] A. Kumar Veerasamy, D. D'Souza, M. V. Apiola, M. J. Laakso, and T. Salakoski, "Using early assessment performance as early warning signs to identify at-risk students in programming courses," in *Proceedings - Frontiers in Education Conference, FIE*, 2020. doi: 10.1109/FIE44824.2020.9274277.

Authors' Profiles

João P. J. Pires received his M.Sc. in Computer Engineering, specialization in Intelligent Data Analysis in 2025 and his B.Sc. in Informatics Engineering in 2023 from the Coimbra Institute of Engineering, Polytechnic University of Coimbra, Portugal. His research interest includes business intelligence, programming education, and programming teaching and learning.

Jorge F. R. Bernardino, PhD, is a Professor at the Institute of Engineering of the Polytechnic University of Coimbra, Portugal, where he has been teaching since 1994. He received a PhD degree from the University of Coimbra in 2002. In 2014, he was a Visiting Professor at CMU. He was the Director of the Applied Research Institute (i2A), IPC 2019-2021. He has authored more than 200 publications in referred conferences and journals and participated in several projects. His research interests include big data, NoSQL, data warehousing, dependability, the Internet of Things, and software engineering.

Anabela J. Gomes, PhD, is a Professor at the Institute of Engineering of the Polytechnic University of Coimbra, where she has been teaching since 1997. She is a member of the Cognitive and Media Systems group at Center of Informatics and Systems of the University of Coimbra and a member of the Applied Biomechanics Laboratory of the Institute of Engineering. Her research focuses on programming education, learning styles and theories, and e-learning. She has authored over 100 publications in renowned conferences and journals and actively contributes to the scientific community through committee work, session chairing, and peer reviewing.

Ana Rosa P. Borges, PhD, is a Professor at the Institute of Engineering of the Polytechnic University of Coimbra, where she has been teaching since 1989. She received the Computer Engineering degree (1989), the MSc degree in System and Information Technology (1995), and the PhD degree in Electrical Engineering, with a specialization in Informatics (2005), from Coimbra University, Portugal. Her current research areas include multiple objective programming optimization, decision support systems, fuzzy sets, business intelligence, and programming education.

Fernanda M. R. Brito R. Correia, PhD, is a Professor at the Department of Informatics Engineering of the Institute of Engineering of the Polytechnic University of Coimbra, where she has been teaching since 1990. She taught from 1986 to 2000 at the University of Coimbra (UC). She received a PhD degree in Computer Engineering from the University of Aveiro (2019) and an MSc degree in Computer Science from the UC (2000). Her research interests include machine learning, computational biology, complex networks, health informatics, BI&A and higher education. She has authored publications in conferences and journals, and contributes to the scientific community through committee work and peer reviewing, and has participated in international projects.

How to cite this paper: João P. J. Pires, Jorge F. R. Bernardino, Anabela J. Gomes, Ana Rosa P. Borges, Fernanda M. R. Brito R. Correia, "Prediction of Students' Performance in Introductory Programming in Higher Education", International Journal of Modern Education and Computer Science(IJMECS), Vol.18, No.1, pp. 1-20, 2026. DOI:10.5815/ijmecs.2026.01.01