International Journal of Data Informatics and Intelligent Computing (IJDIIC)
Vol. 3, No. 4, December 2024, pp. 40~61
DOI: 10.59461/ijdiic.v3i4.145

Investigating Aptitude in Learning Programming Language Using Machine Learning and Natural Language Processing

Muhammad Faisal Iqbal[1], Adeel Zafar[1], Umer Khalil[2], Afia Ishaq[1]
[1]Department of Data Science & Cyber Security, Riphah International University, Islamabad, Pakistan
[2]ITC Faculty of Geo-Information Science & Earth Observation, University of Twente, Enschede, The Netherlands

Article Info

Article history:

Received October 07, 2024
Revised November 29, 2024
Accepted December 09, 2024

Keywords:

Education of Computing
Programming Language
Aptitude Predictors
Machine Learning
Natural Language Processing

ABSTRACT

This study investigates the relationship between prerequisite courses and skill acquisition in programming education. It proposes a case study examining cognitive, natural language, and mathematical aptitude indicators as predictors of programming performance. Analyzing data from 1238 undergraduate students at Riphah International University, the research employs Machine Learning models to predict outcomes, achieving high R2 scores and low Mean Squared Error rates. A zero-shot text classification model identifies required aptitude skills: 62% cognitive, 24% natural language, and 14% mathematical. These skills are mapped to predicted programming course scores, offering a new approach to understanding programming language aptitude. The study aims to bridge the gap between prerequisite courses and subsequent skill development, contributing valuable insights to computing education curriculum design.

Corresponding Author: Muhammad Faisal Iqbal (e-mail: muhammadfaisal.softech@gmail.com)

1. INTRODUCTION

In today's rapidly evolving technological landscape, computing education continues to grapple with persistent challenges in teaching programming language effectively. Despite significant advances in research and curriculum development [1], students often struggle to acquire the intended programming skills. While prerequisite courses aim to prepare students for advanced programming studies, a significant gap often remains between the intended and actual skill acquisition [2], [3], [4].

Recent studies have explored various facets of programming language learning, including the impact of AI tools on novice learners [5], student perspectives on AI assistants [6], shared neural resources in programming [7], and the effect of professional development on coding education [8]. Additionally, researchers have investigated instructional modalities, conceptual transfer between languages, and correlations with Computational Thinking skills [9].

Despite these advancements, the fundamental question persists: What aptitudes are essential for successfully learning programming languages? Traditional approaches, such as the IBM Programmer Aptitude Test and standardized examinations like the SAT, have shown limited efficacy in predicting programming success [10]. Moreover, while programming logic is rooted in mathematical skills, the role of linguistic abilities in mastering programming syntax remains underexplored.

This study aims to address this gap by developing a comprehensive methodology to assess students' programming performance and identify the specific aptitudes influencing their learning outcomes. We hypothesize that a combination of cognitive, linguistic (Natural Language), and mathematical skills significantly impacts a student's ability to learn programming languages effectively.

To test this hypothesis, we will employ a mixed-methods approach, utilizing Machine Learning (ML) and Natural Language Processing (NLP) techniques to analyze student performance data and aptitude indicators. Our research question is: What combination of aptitudes best predicts success in learning programming languages?

The significance of this study lies in its potential to inform more effective curriculum design and student support strategies in computer science education. By identifying key aptitudes, educators can develop targeted interventions to enhance students' programming skills, potentially improving retention rates and overall success in computing programs.

2. LITERATURE REVIEW

In the paper "Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming," a controlled experiment with 69 novice learners was conducted to investigate the impact of AI code generators, specifically OpenAI Codex, on introductory programming [5]. The findings revealed that learners with Codex access demonstrated significantly improved code-authoring performance, with increased completion rates and higher scores. While learners with access to Codex showed better performance in post-tests, the difference was not statistically significant. The study also highlighted the influence of prior programming competency, with learners having higher pre-test scores benefiting more from AI code generators, such as Codex, during their training. Additionally, qualitative feedback suggested that Codex usage reduced stress and enhanced learners' enthusiasm for programming, although concerns about over-reliance were raised.

The study by [6] investigates students' perspectives on using ChatGPT in programming education. The study involved 41 undergraduate students over eight weeks, where students used ChatGPT for weekly programming assignments. The findings indicate advantages such as quick and correct answers, improved thinking skills, and debugging facilitation. However, limitations include potential laziness, incomplete or incorrect answers, and concerns about professional impact. The research suggests integrating generative AI tools into programming courses with caution, considering both advantages and limitations.

In [7] demonstrated the shared neural resources between computer code comprehension and formal logical inference, shedding light on the cognitive processes underlying programming language learning. However, a potential limitation lies in the study's focus on expert programmers, possibly neglecting the nuances of aptitude development in novice learners. Further research may be needed to explore how these neural patterns evolve with varying levels of programming experience, contributing valuable insights to investigations into the aptitude required for learning programming languages.

In study [8] assessing the impact of continuous professional development on elementary teachers' self-efficacy in teaching coding and computational thinking. The research emphasizes the importance of hands-on methods and student success in enhancing teachers' confidence. The findings highlight the effectiveness of year-long professional development in improving teacher self-efficacy, with a focus on various coding concepts. The study provides insights into the professional growth of elementary teachers in coding and computational thinking.

The literature reveals a noteworthy association between programming experience and enhanced neural efficiency in figural reasoning tasks, as evidenced by the study conducted by [9]. This aligns with the broader understanding of programming skills influencing cognitive processes and figural reasoning abilities. However, a potential gap in the existing research is the lack of direct assessment of cognitive aptitude, such as critical thinking, problem-solving, and logical reasoning, which are crucial components in the context of learning programming languages. Addressing this gap is essential for a comprehensive understanding of the cognitive aptitude required for effective programming language acquisition.

The researcher [11] investigated predictors of success in an introductory programming course, revealing first-semester GPA and language admission test scores as significant factors. The study recognized the complexity of predicting success, considering varied factors across engineering specializations. However, the study did not directly address the nuanced aptitude for learning programming languages, leaving a gap that does not align with research on programming language aptitude using machine learning and natural language processing.

A study [12] delves into the impact of technical reading training and spatial skills training on novice programming ability, revealing that a CS-focused technical reading intervention yields larger programming gains than a standardized spatial intervention. The research underscores the distinctiveness of cognitive skills, emphasizing the relevance of technical reading and spatial abilities to programming success. However, a notable limitation is the potential lack of generalizability due to a small, homogeneous sample with high spatial abilities. The study conducted during the COVID-19 pandemic introduces confounding factors, urging caution when applying findings to diverse learner populations or alternative contexts.

The study about neuroimaging [13] delves into the neural underpinnings of reading, visualization, and coding in novice programmers, using fNIRS to identify distinct brain activation patterns in occipital, parietal, and frontal cortices. While shedding light on the cognitive processes involved in coding, the study faces limitations, including a restricted participant pool from a single university, potentially limiting generalizability. The reliance on fNIRS introduces the risk of false negatives, and the study's construct validity is challenged by the multifaceted nature of spatial abilities. Acknowledging these limitations, the research underscores the need for further exploration into alternative experimental paradigms and a broader focus on specific programming activities to strengthen the study's findings.

To address a gap in the literature on instructional modalities in programming education by investigating the effectiveness of audio, text, or combined explanations for code comprehension. Rooted in educational psychology, the research focuses on the modality effect and its implications for instructional design, contributing to ongoing discussions on improving programming education methodologies [14]. However, limitations include a narrow focus on the modality effect, potential oversight of other influencing factors, limited discussion on sample characteristics, and questions about generalizability. The replication nature of the study may also restrict its applicability to diverse educational settings, suggesting a need for future research with more comprehensive variables and larger, more diverse samples.

The study [15] explored conceptual transfer in the context of transitioning from procedural Python to object-oriented Java. The study underscores the significance of semantic transfer based on syntax similarities and introduces a validated Model of Programming Language Transfer (MPLT). Emphasizing the need for deliberate semantic transfer interventions, the findings contribute a validated model and pedagogical guidelines for educators. However, limitations include potential generalizability issues due to the specific focus on this programming language transition, sample size constraints with relatively novice programmers, and the reliance on mixed methods, which may introduce biases. The study's duration and scope may also limit insights into long-term conceptual transfer, and external factors such as individual learning styles could impact results.

To examine undergraduate students' performance on iterative and recursive programming tasks, highlighting significant differences in success rates between these problem framings. Notably, iterative versions excel in non-branching numeric computation, while recursive versions outperform in array classification. The research underscores the impact of contextual factors like experience, gender, ethnicity, and spatial ability on programming performance, providing nuanced insights into language learning [16]. However, limitations include a potential lack of generalizability, reliance on self-reported data posing reliability concerns, and specificity to certain languages and curricula, limiting broader applicability. Despite these constraints, the study yields valuable insights into student performance and error patterns in iterative and recursive programming scenarios.

The study [17] investigated the correlation between Computational Thinking (CT) skills and proficiency in ChatGPT-based software development. The study found that CT skills were predictive of the ability to develop software using ChatGPT. However, the research has limitations, such as a narrow focus on CT skills, reliance on a single task for assessment, and potential age-related and generalizability concerns. Nevertheless, the study offers valuable insights into the evolving competencies of programmers using advanced tools like ChatGPT, laying the groundwork for understanding the skills required in this context.

The researcher in [18] utilized machine learning algorithms to predict academic performance in middle- and high school students, revealing the significant influence of factors like physical activity, stress, family size, parent's marital status, food intake, prior academic performance, and obesity on academic outcomes. Despite its valuable insights, the study's limitations include a sample population confined to a metropolitan city, potentially restricting generalizability to rural settings. The reliance on self-reported questionnaires for stress assessment introduces subjectivity, prompting the need for more objective stress measurement techniques. Acknowledging low variability in body weight raises concerns about generalizability, emphasizing the importance of diverse participant pools in future research for a comprehensive understanding of academic performance predictors.

The dissertation [19] comprehensively investigated the understanding of computer programs from computational and cognitive standpoints. The study evaluates code models' comprehension, explores behavioural responses to code, and employs neuroimaging to study the neural bases of code comprehension. While contributing significantly to the understanding of cognitive processes in code comprehension, the research suggests future directions, including applying findings to computer science education and exploring separate architectures for distinct reasoning tasks. However, the study has limitations, such as a narrow focus on code comprehension, potential ecological validity concerns in behavioural metrics, and insufficient exploration of cognitive differences across programming languages. The dissertation also highlights the need to address challenges and biases in training large-scale language models and calls for explicit methodologies for applying neuroimaging results to computer science education.

Study [20] investigates the prediction of Python programming and debugging skills through a combination of intermittent knowledge assessments, individual psychometrics, and resting-state EEG measures. The research highlights the significance of declarative knowledge assessments, particularly post-module quizzes, in predicting multiple-choice test accuracy, programming accuracy, and debugging accuracy. Despite the valuable insights into the interplay between cognitive characteristics and declarative knowledge, the study has limitations. Its narrow focus on Python programming may restrict generalizability, and its reliance on intermittent knowledge assessments might overlook crucial aspects of programming skills. The use of online platforms introduces variability, and the sample's lack of prior programming experience raises questions about the study's broader applicability. Further, the validity of resting-state EEG measures for predicting real-world programming expertise requires additional validation. Lastly, the study's reliance on self-paced online learning may limit its generalizability to more structured educational settings.

In [21], click or tap here to enter text employed functional magnetic resonance imaging (fMRI) to investigate neural representations of computer programs, revealing insights into how specific brain regions encode static and dynamic code properties. The study's mapping of brain representations to machine learning models enhances our understanding of the intricate connection between the human brain and code representations, showcasing the potential application of neuroscience in deciphering programming cognition. However, limitations include a focus on a dataset with simple Python programs, potentially restricting generalizability, an emphasis on comprehension tasks, and the need for further exploration of brain-code mappings across diverse programming languages. Sample size concerns and the evolving nature of neuroimaging technology also prompt considerations for refining future research.

In [22] underscored the importance of Computational Thinking (CT) education in fostering essential skills such as problem-solving and logical reasoning. They discuss the current emphasis on CT in education, citing initiatives like "CT for All" and various standards. The authors present four studies highlighting the effectiveness of metaphors in teaching CT concepts, particularly in programming education. Identified challenges encompass the need for clear CT competencies, efficient use of metaphors, exploration of pedagogical strategies and technologies, teacher professional development, and the assessment of CT competencies. However, limitations include a primary focus on primary education, potential oversights in discussing challenges, and a perceived emphasis on the need for further research without comprehensive solutions.

The study by [23] introduces the Programming-oriented Computational Thinking Skills (P-CTS) scale, assessing conceptual knowledge, algorithmic thinking, and evaluation in the context of programming education. The findings underscore the importance of programming experience, emphasizing the critical threshold of over one year for skill enhancement. Notably, the research highlights gender differences in evaluation skills, contributing valuable insights to inclusive programming education. However, limitations include a focus on university students, raising questions about generalizability to other educational levels, and the need for further exploration regarding the scale's applicability in interdisciplinary contexts or STEM education. Additionally, the study suggests avenues for future research to delve into specific factors influencing gender differences in evaluation skills, enhancing the scale's robustness and applicability.

The study by [24] delves into the cognitive benefits of learning to code, asserting that coding skills share thinking processes with mathematical Modelling and creative problem-solving. The research presents empirical evidence supporting the potential transfer effects of coding skills to other cognitive domains, underscoring the necessity for explicit training to facilitate such effects. Despite the promising evidence, the study acknowledges limitations, including the absence of a definitive causal relationship and potential confounding factors. The meta-analytical approach's limitations are recognized, emphasizing the need for more precise assessment tools and further research to unravel the intricate mechanisms and conditions influencing the transferability of coding skills.

Researchers [25] investigated the neurocognitive processes in code comprehension, using electrophysiological measures to reveal N400 effects for semantic congruencies and P600 effects for syntactic anomalies in Python code. The study suggests that expert programmers prioritize structural aspects over semantics, emphasizing the incremental nature of code comprehension. However, limitations include the cross-sectional design hindering causal inferences, a focus on Python programmers limiting generalizability, and the exclusive emphasis on certain code violations overlooking broader aspects of comprehension. Addressing these limitations in future research is crucial for a more comprehensive understanding of the cognitive processes involved in programming language learning.

Study [1] tackled challenges in teaching introductory programming through the HTP programming application, designed for early problem identification and intervention. Utilizing action research, the study focuses on students at the Polytechnic of Guarda, yielding positive outcomes in monitoring and a predictive neural network model for early failure detection. However, limitations include context-specific effectiveness, potential bias in student engagement, and uncertainties introduced by external factors like the COVID-19 pandemic, emphasizing the need for cautious interpretation and further validation.

In this era, computer programming plays a significant role in solving real-life problems in every domain. The field usually resembles the taxonomy of science, technology, engineering, and mathematics (STEM) areas. Programming languages are also like natural languages. Empirically, minimal research has been evaluated on the cognitive aspect of programming to understand the human mind and has the power to change educational practices [26]. Thus, it shows that programming is specifically like natural language. However, as their proposed framework explains, computer programming and natural language are based on a set of building blocks, such that words and phrases are specifically composed in natural language. Secondly, computer languages are based on variables. Moreover, it explains the criteria for these building blocks to combine and create new meanings from them.

A recent study by [27] investigated the relationship between a second natural language (L2) and a programming language. However, their purpose was to explore the 1st year students' (College of Engineering and Applied Sciences at Cincinnati University) concepts of computing with a 1st year engineering curriculum. Moreover, students' course performance is evaluated based on previous and ongoing experiences with SLA and accounted for through their previous knowledge of programming languages. They set the criteria for a second natural language that at least one foreign language course has to be taken or studied (e.g., Spanish, German, French, and Italian). Secondly, for the computing language, a single course must be studied previously at the university level or in high school. They said in their study that they used non-parametric approaches to discover the significant differences between those who experienced a second natural language and those who did not. Thus, their statistical findings explain that there is no inconclusive relationship between them, but further study will be more appropriate.

Additionally, a study was conducted to measure the programming skills of high school (K-12) students [28]. This research was carried out on a group of students at a large university in the Pacific Northwest of the United States. The work aims to measure the effects of previous programming experience in introductory programming courses. Secondly, they evaluated the previous programming knowledge with the introductory programming course midterm and final term assessments, as well as the survey and aptitude test. Thus, their findings explain the mismatch or differences between the results of surveys and aptitude tests. The results of the survey reveal that previous knowledge, particularly that obtained in high school, has only a minor impact on midterm performance, while the aptitude performance of the students has a crucial impact on both the midterm and final term in the introductory programming course.

A particular study was also conducted on problem-solving as a predictor of programming performance [29]. However, the purpose of the research specified the correlation of problem-solving ability with their academic performance in 1st year programming courses. Moreover, their limitation explains that they used five specific variables in the study, such that a student's achievements in a programming course are set as a dependent variable. As for independent variables, they defined four aptitude tests composed of predictors: logical reasoning, non-verbal reasoning, numerical reasoning, and verbal reasoning. Notably, each student's participating group consists of 379. Nevertheless, their finding indicates a correlation between students' logical reasoning, numerical reasoning, verbal logic, and performance in computer programming modules. Secondly, their study mentions no correlation between students' non-verbal reasoning and performance in computer programming modules.

Countless instructors and educators claim that innumerable students are facing difficulty in learning 1st-year University or institution computing science courses. Thus, various initiatives are assessed to support student's academic success. University instructors provide several forms of academic support with the programme, such as learning strategies that should be discussed with students. The forum is named "Academic Enhancement Program" (AEP), which was invented and is active by the School of Computing Science and the Learning Commons at Simon Fraser University. This particular forum provides learning strategies for 1styear CS University courses from late 2006. In parallel to these learning strategies, the authors in [30] came up with another strategy to improve or enhance the learning experience of students that focused on peer instruction and active learning with audience response systems and was named "i-clickers". In their study, they defined three variables such as predictors (MID, WIC, and FIN), which were used in ordinary multiple regression analysis and will be analyzed for the potential of these activities over course success. The limitation of the study is based on the introductory course CS, which is offered in the 2013 Fall Semester and is composed of 363 students. Their findings indicate that the weighted i-clicker is not a very suitable predictor for the final exam score.

Nonetheless, a recent study by [31] linked natural language aptitude to atomic differences in learning programming languages. In their study, they believe that natural language itself is a strong predictor for learning and coding programming languages, which signals that learning a new programming language may be parallel in terms of learning a new natural language. However, they described various variations of the findings in their study, such that behavioural and neural (resting-state EGG) are the indicators of measuring language aptitude, whereas numeracy and fluid cognitive measures, i.e., fluid reasoning, working memory, and inhibitory control, were defined as the predictors. On the other hand, researchers also suggested that programming languages can be predicted with the predictor's mathematical aptitude [32]. Chomsky's theory, based on formal language, recently defined accurate sentence building in natural languages as an origin tool in mathematics theory and programming languages [33].

The possible limitation of the present approaches, as determined by the literature review, is that previously, no single educational case study existed that investigated the aptitude predictors collectively, such as cognitive, natural language, and mathematics. This is where the novelty of the present work comes in trying to fill this gap by undertaking a comprehensive study that touches on all aspects of aptitude for learning programming languages. Firstly, a large undergraduate academic record and course objectives dataset related to Pakistan's universities are employed in this study. Secondly, the study evaluates the developed dataset using several ML algorithms and uses an NLP state-of-the-art pre-trained text-based algorithm. Thirdly, it evaluates all the models based on error metrics. Lastly, it compares all the employed models to select the most effective and best-performing model.

3. METHODOLOGY

The complexity of understanding programming language aptitude necessitates a rigorous and systematic research approach.

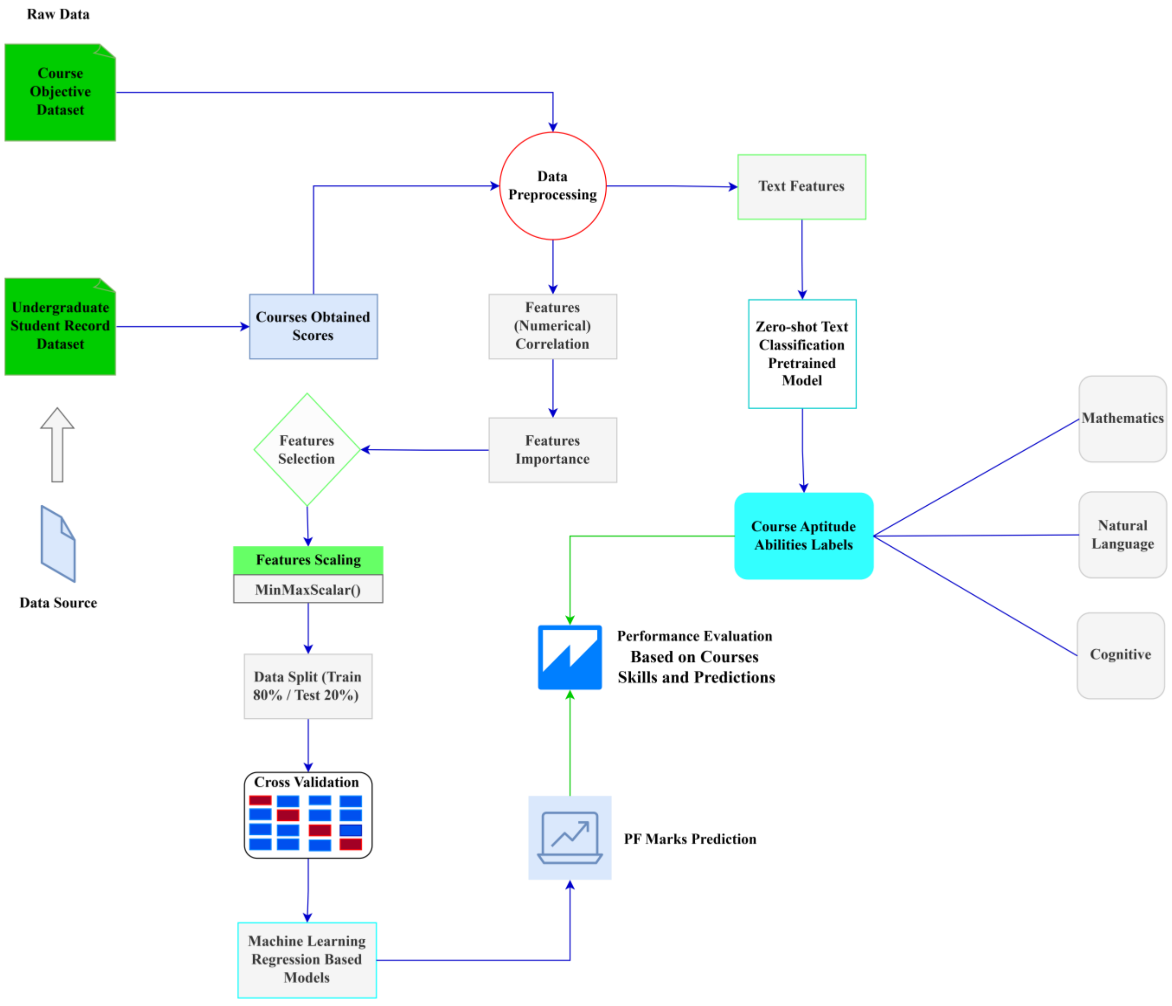
Raw Data
Course Objective Dataset
Data Preprocessing
Text Features
Undergraduate Student Record Dataset
Courses Obtained Scores
Features (Numerical) Correlation
Zero-shot Text Classification Pretrained Model
Features Selection
Features Importance
Mathematics
Course Aptitude Abilities Labels
Natural Language
Data Source
Features Scaling
MinMaxScalar()
Performance Evaluation Based on Courses Skills and Predictions
Cognitive
Data Split (Train 80% / Test 20%)
Cross Validation
PF Marks Prediction
Machine Learning Regression Based Models

Figure 1. Flow chart of the implemented methodology for this study

Table 1. Data Sources and Characteristics

Index	Data Sources	Key Characteristics
1	Student’s Academic Scores	Secondary School Certificate (SSC) scores Higher Secondary Certificate (HSSC) scores First and second-semester course scores, with emphasis on programming fundamental core courses
2	Course Outlines	Analyzed to determine the specific types of skills required for each course Skills categorized into cognitive, natural language, and mathematics Mapped to the programming fundamental scores to estimate individual student skills

This single case study was designed to investigate the aptitude skills necessary for learning a programming language at Riphah International University Islamabad I-14 Campus, Islamabad. By employing a carefully structured research design, our study aimed to capture the multifaceted nature of programming language learning through two primary data sources, as detailed in Table 1.

The methodological architecture, visually represented in Figure 1, provides a holistic framework for our investigative process. This approach allows for a nuanced examination of the various skills and factors that potentially contribute to successful programming language acquisition.

3.1. Dataset Gathering

Understanding the right data is crucial to any research. This section provides a comprehensive overview of the datasets utilized in our research, detailing the sources, composition, and characteristics of the undergraduate student and course objective datasets. By providing a description of data collection and preprocessing, we establish the foundation for our investigation into programming aptitude skills. The dataset for this case study was collected from the Department of Computer Science and Software Engineering. Two distinct datasets were compiled, as detailed in Table 2.

Table 2. Overview of Gathered Datasets

Datasets	Sample Counts
Undergraduate student dataset	1238
Course’s objective dataset	15

3.1.1. Undergraduate Student’s Dataset

The undergraduate student dataset forms the core of this study’s analysis, capturing academic records from students enrolled in two disciplines: Bachelor of Science in Software Engineering (BSSE) and Bachelor of Science in Computer Sciences (BSCS). It includes Secondary School Certificate (SSC) and Higher Secondary School Certificate (HSSC) marks, academic records from the fall 2019 and spring 2020 semesters, and programming fundamental marks for all students.

Notable characteristics of the dataset are that some records may be incomplete due to semester-specific course requirements or student withdrawals. Certain SSC and HSSC records were missing from the original data sheets. The dataset was consolidated by merging records based on student roll numbers. Due to missing variables and elective course choices, the dataset is imbalanced. The dataset comprises 1238 student records with 19 attributes, including:

- Identification: Roll Number
- Prior Education: SSC Marks, HSSC Marks
- Course Marks: 16 different courses, including Programming Fundamentals (dependent variable)

All attributes except Programming Fundamentals are independent variables.

3.1.2. Courses Objective Dataset

The course objective dataset plays a pivotal role in this study by linking specific course goals to programming fundamental performance. This mapping helps identify the key aptitude skills necessary for programming success. Key features include:

- Source, i.e., course outlines from the Computer Science and Software Engineering Department
- Attributes, i.e., Courses (15 instances) and Objectives (course-specific criteria and aims)
- Nature, i.e., Text-based dataset reflecting natural language

Courses included:

1. Introduction to Information and Communication
2. English Composition and Comprehension
3. Applied Physics
4. Discrete Structures
5. Data Structures and Algorithms
6. 3D Modelling
7. Communication and Presentations Skills
8. Linear Algebra
9. Pre-Calculus-1
10. Pre-Calculus-2
11. Probability and Statistics

12. Life and Living
13. Calculus and Analytical Geometry
14. Digital and Logic Design
15. Programming Fundamentals

Each course has tailored objectives, which may overlap with other institutions but are specific to this university's curriculum.

3.2. Aptitude Abilities and Mapping

Expanding on the dataset descriptions, this section outlines our systematic approach to identifying and mapping the aptitude abilities that are essential for programming language learning. We detail our research focus, text classification model, and the innovative process of correlating course objectives with student performance in programming fundamentals.

3.2.1. Research Focus

This study is centred on identifying the key aptitudes and abilities that influence programming language learning, focusing on three key areas, i.e., cognitive, natural language, and mathematical skills.

We employ these categories to estimate the probability of each aptitude's influence on programming language learning.

3.2.2. Text Classification Model

To effectively analyze course objectives and map them to the identified aptitude categories, we employ a pre-trained zero-shot text classification model with the following key characteristics:

- Training Dataset, i.e., Multi-Genre Natural Language Inference (Multi-NLI), comprising 433,000 sentence instances
- Input, i.e., Course objective corpora (textual data)
- Output, i.e., probability estimates for each aptitude category (cognitive, natural language, mathematics)

The model classifies text sequences using relevant labels, accommodating various domains such as education, politics, astronomy, etc.

3.2.3. Aptitude Mapping

The estimated aptitude abilities derived from course objectives are mapped to students' performance in programming fundamentals. This mapping process involves predicting programming fundamental scores using Machine Learning (ML) models and correlating these predictions with the estimated aptitude abilities from the text classification model.

3.2.4. Feature Selection

For ML modelling, we selected a subset of features from the original dataset. The total number of features used was 7, while the selection criteria were features with minimal missing data, and the main purpose was to enhance model accuracy and reliability.

The subsequent sections will detail the specific attributes used and the ML models employed for predicting programming fundamental scores.

3.3. Implementation

3.3.1. Dataset Overview

The implementation phase utilized a student dataset comprising 18 features, including identification, prior education marks, and various course marks. This dataset focused on students who completed the programming fundamentals course alongside elective and core subjects. Due to variations in course timing, some missing values were present in the dataset. However, comprehensive academic records (SSC and HSSC) were available for all students, minimizing the impact of missing data.

3.3.2. Imputation Technique

To address missing course mark values, we employed the K-Nearest Neighbor (KNN) imputation technique. This method, while computationally intensive for large datasets, proved effective and highly accurate for our small to medium-sized datasets. The KNN imputer computed distances between new points and training points, selected the k closest data points based on distance, and applied appropriate distance metrics depending on the data type. Figure 2 illustrates the workflow for filling in missing values in a student's record.

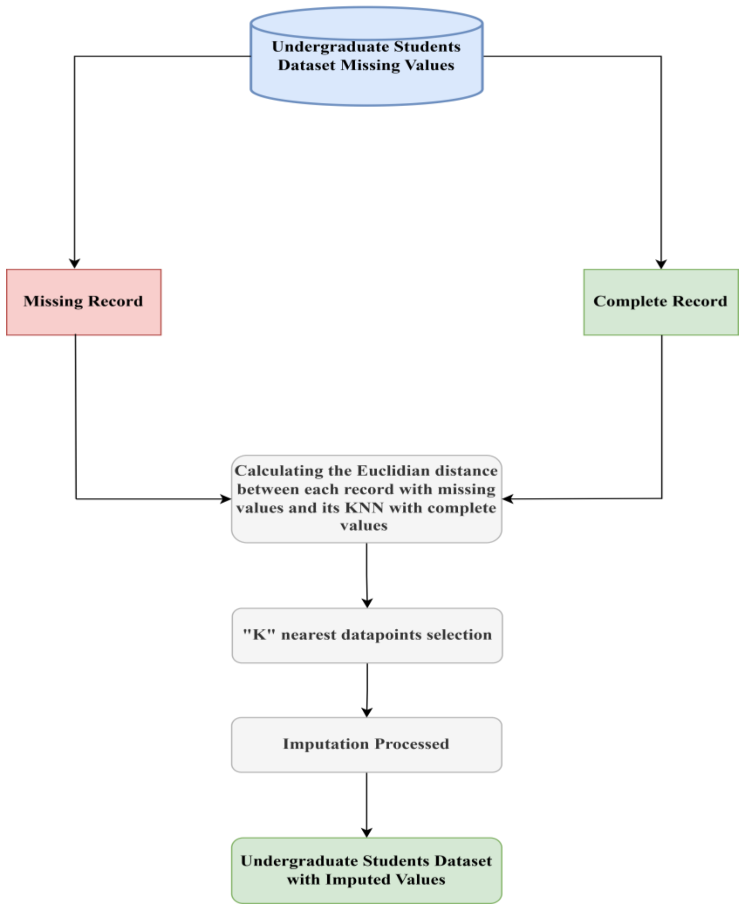
Undergraduate Students Dataset Missing Values
Missing Record
Complete Record
Calculating the Euclidian distance between each record with missing values and its KNN with complete values
"K" nearest datapoints selection
Imputation Processed
Undergraduate Students Dataset with Imputed Values

Figure 2. KNN imputation workflow for student's regression-based dataset

3.3.3. Data Preprocessing

Data preprocessing was crucial to address inconsistencies in grading scales between SSC/HSSC marks (0-1050, 0-1100 scale) and other variables (0-100 scale). We normalized all features using min-max scaling to a range of 0 to 1, ensuring data generalization and improving machine learning model interpretation. This transformation allowed for more accurate analysis and comparison across different academic measures.

3.3.4. Correlation Analysis

To assess the relationships within the dataset, we conducted a correlation analysis between dependent and independent variables. Our findings revealed that all attributes showed strong self-correlation, as expected. The dependent variable "Programming Fundamentals" positively correlated with most independent variables, except for "SSC/HSSC Marks," which showed a different relationship pattern. This analysis provided valuable insights into the factors potentially influencing success in programming fundamentals. The results are illustrated in Figure 3.

3.3.5. Implementation Strategy

This implementation strategy, combining imputation techniques, feature scaling, and correlation analysis, ensured a robust examination of the factors influencing success in programming fundamentals. By accounting for data complexities and variations in student academic histories, we established a solid foundation for further analysis and model development in our study of aptitude skills necessary for learning programming languages.

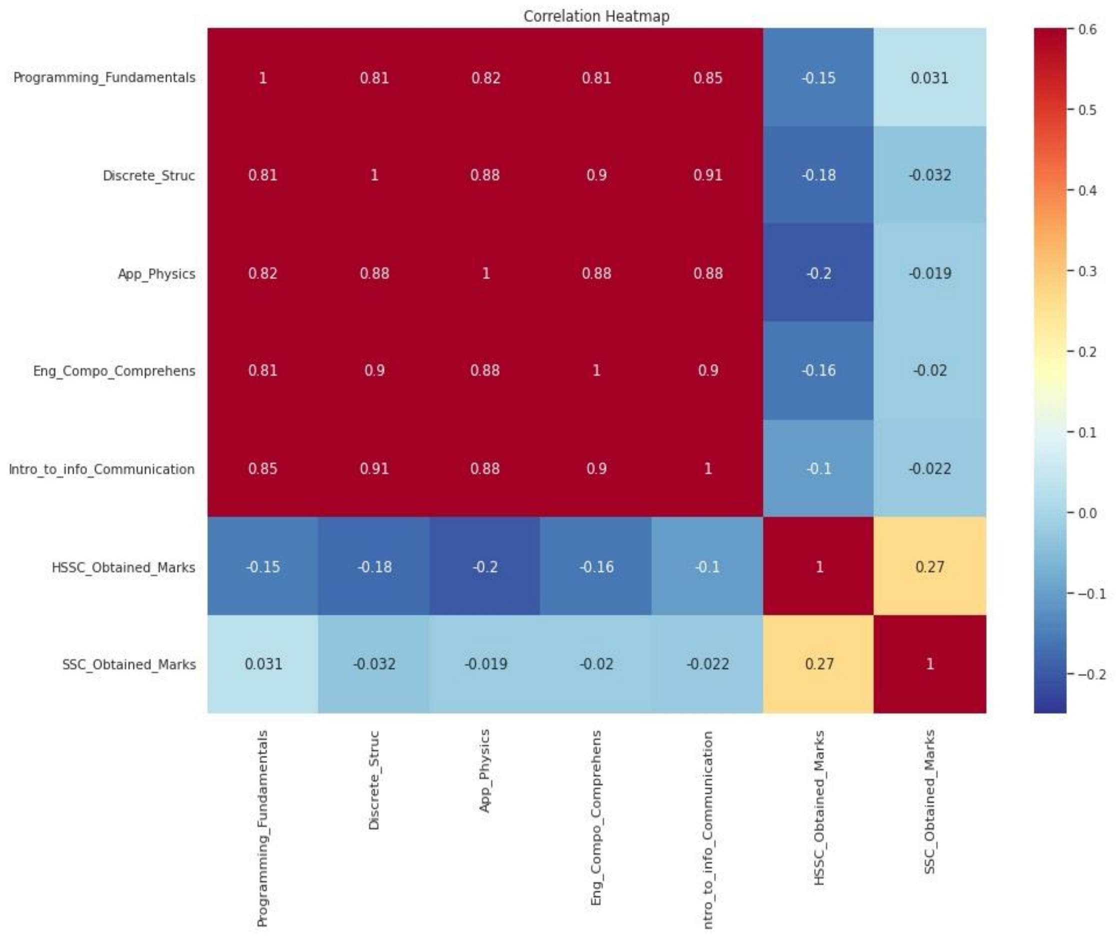

Figure 3. Programming Fundamental correlations along with independent variables

3.4. Modeling Using Machine Learning (ML) and Natural Language Processing (NLP)

In this section, we outline a comprehensive framework for predicting programming fundamental marks through machine learning (ML) and natural language processing (NLP) techniques. This approach leverages state-of-the-art regression models and zero-shot text classification methods to map aptitude skills and predict student performance.

We divided the data into training (80%) and testing (20%) sets using scikit-learn's train_test_split function, with random_state=45 for consistency. Four state-of-the-art ML models were employed to predict programming fundamentals marks.

3.4.1. KNeighbors Regression (KNR)

The KNeighbors Regression (KNR) is a supervised machine learning method used for predicting numerical values by evaluating the similarity between data points [34]. Distance metrics such as Euclidean, Manhattan, and Minkowski are used for continuous variables. The Euclidean distance formula is mentioned in Equation 1.

$$\sqrt{\sum_{i=1}^{n}(yi - xi)^2} \qquad (1)$$

Selecting an appropriate K value is crucial to balance between overfitting (small K) and underfitting (large K). While rare exceptions exist, higher K values are generally recommended to avoid overfitting.

3.4.2. Random Forest Regression (RFR)

Random Forest Regression (RFR) is a robust ensemble learning technique that leverages multiple decision trees to enhance prediction accuracy. By combining bootstrapping, bagging, and aggregation, RFR mitigates overfitting and improves generalization performance [35]. The model's performance is determined by the aggregated results of these trees.

The RFR model is constructed by drawing an input variable X from the training set, creating a specified number (N) of regression trees, after (N) variables in the RFR model and trees such as $\{\{T(i)\}\}_i^N$ are increased, averaging predictions from all decision trees. The predictor of regression after n variables and m trees is mentioned in Equation 2.

$$f_{rf}^N(i) = \frac{1}{N}\sum_{N=1}^{N} T(i) \tag{2}$$

RFR promotes tree diversity through bagging, which involves randomly sampling training data with replacement. When constructing trees, RFR randomly selects a subset of features and chooses the best feature and split point from this subset. This approach reduces the correlation between trees and lowers generalization error [36].

RFR trees grow without pruning, making them relatively lightweight. Out-of-bag (OOB) subsets, formed by data points not included in the training, can be used to estimate individual tree performance [37]. As the number of trees increases, the generalization error decreases, indicating reduced overfitting risk.

3.4.3. Gradient Boosting Regressor (GBR)

Gradient Boosting Regression (GBR) is an ensemble technique designed to sequentially improve prediction accuracy by adding models iteratively. Each new model addresses the errors of the previous ensemble, making GBR effective for reducing prediction errors [38]. GBR generalizes by optimizing any differentiable loss function using gradient descent [39]. GBR comprises three components: a loss function to be minimized, a weak learner for predictions, and an additive model combining weak learners to reduce the loss function [40]. The key innovation is constructing new weak learners strongly correlated with the negative gradient of the loss function, computed with respect to the entire ensemble. This approach allows for arbitrary loss functions, resulting in sequential error fitting for typical squared-error loss.

3.4.4. Light GBM Regressor (LGBMR)

Light GBM Regressor (LGBMR) is a cutting-edge machine learning method known for its efficiency in both data regression and categorization tasks. It employs innovative techniques like Exclusive Feature Bundling (EFB) and Gradient-based One-Side Sampling (GOSS) to optimize performance and reduce processing times [41]. LGBMR outperforms traditional methods in data scanning, sampling, clustering, and classification, making it superior in terms of memory usage, processing time, and computational efficiency. Its advantages include faster training, optimal memory utilization, satisfactory accuracy, parallelism, and large-scale data processing capabilities.

3.4.5. Aptitude Mapping Using NLP

Aptitude Mapping Using NLP leverages zero-shot learning techniques to classify unseen aptitude labels within course objectives. By exploiting semantic relationships in a high-dimensional vector space, this method enables accurate classification without requiring extensive labelled data [42].

For aptitude mapping, a pre-trained NLP model for zero-shot text classification is deployed on a text-based course objective dataset. The process involves installing necessary libraries and loading the zero-shot classification pipeline, providing the model with text sequences (course objectives) and premise candidate labels (predictors: cognitive, natural language, and mathematics). The model classifies courses based on their potential abilities or the predictor with the greatest influence.

This approach estimates values for each listed course and its objectives, effectively mapping aptitudes without requiring extensive labelled training data. Zero-shot learning models require the presence of a labelled training set of seen and unseen classes, with both classes related in the semantic space where seen class knowledge can be transferred to unseen classes [42].

4. RESULTS AND DISCUSSION

This section presents the results of our investigation into aptitude skills necessary for learning programming languages. Using undergraduate student data from Riphah International University's Islamabad campus, our analysis identifies cognitive abilities as the primary aptitude influencing programming performance with a potential range of 62%. Additionally, 24% of natural language skills and 14% of mathematics skills are necessary. As a result, these findings are significant, and in an educational setting, it is crucial to focus on students' cognitive abilities, as these may have a bearing on their future learning. Furthermore, natural language and mathematics skills are also important in the context of learning programming languages.

The single educational case study was designed to delve deeper into the aptitude skills required for learning programming languages. However, to accomplish this task, we employed novel approaches to identify the specific aptitude skills. Our methodology encompasses ML and NLP models that are used to address a specific problem. The results demonstrate that we can predict students' scores in the fundamental programming course based on their previous courses and academic scores. Second, course objective predictors represent the required skills, namely cognitive, natural language, and mathematics. These predictors should be applied to the predicted scores of PF to determine the skill set a student possesses in order to pursue their journey in the programming language domain. Additionally, previous academic courses also define the abilities attained by students, making it easier to identify their specific ability traits. Therefore, when designing curricula, instructors need to focus on students' abilities based on their previous courses. This will make it easier for students to tackle their further learning aspects. Moreover, by employing this methodology, instructors can also predict the learning of programming language courses, leading to more effective learning paradigms. Figure 4 illustrates the essential aptitude skills required for learning programming languages.

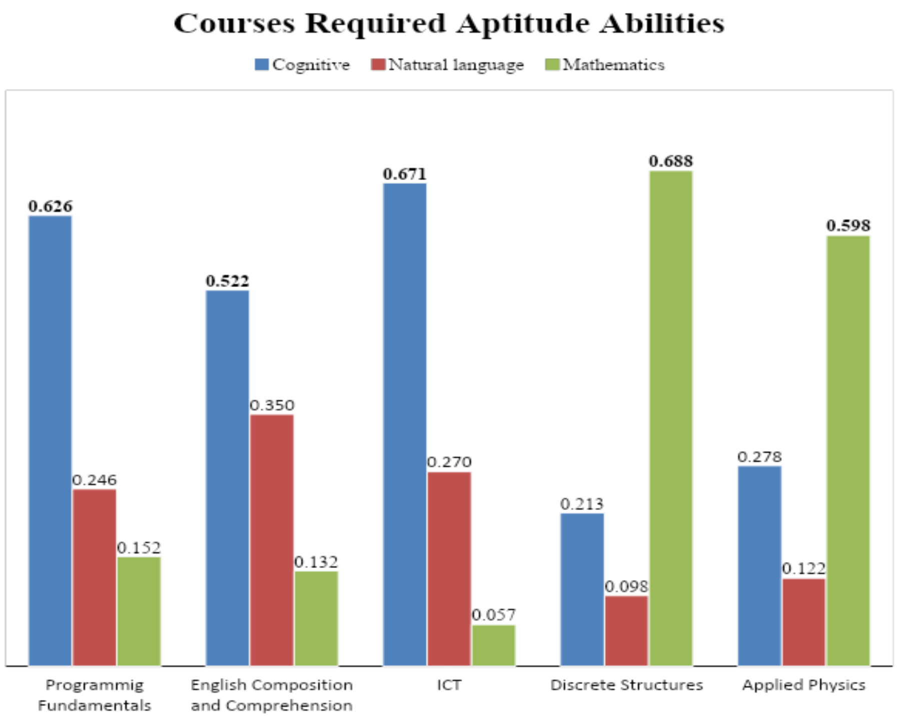

Figure 4. Programming language learning aptitude abilities.

4.1. Data Preparation

4.1.1. Assessment of Missing Data

Several courses contain varying levels of missing data, as outlined in Table 3. These missing values are either due to administrative errors or course withdrawals by students.

Table 3. Missing Values Count Per Variable

Index	Courses	Missing Values
1	Roll Number	0
2	SSC Obtained Marks	1120
3	HSSC Obtained Marks	283
4	Introduction to Information and Communication Marks	623
5	English Composition and Comprehension Marks	624
6	Applied Physics Marks	634
7	Discrete Structures Marks	865
8	Data Structures and Algorithms Marks	1236
9	3D Modeling Marks	1236
10	Communication and Presentations Skills Marks	1237

11	Linear Algebra Marks	1236
12	Pre-Calculus – I Marks	1187
13	Pre-Calculus – II Marks	1237
14	Probability and Statistics Marks	1236
15	Life and Living Marks	1237
16	Calculus and Analytical Geometry Marks	1022
17	Digital and Logic Design Marks	1236
18	Programming Fundamentals Marks	621

Nine features exhibit the most significant amount of missing data, with some courses showing up to 95% missing entries, as highlighted in Table 4. These courses are considered for removal due to the infeasibility of imputation.

Table 4. Features with Maximum Missing Data

Index	**Features**	**Maximum Present Values**
1	Data Structures and Algorithms Marks	1236
2	3D Modeling Marks	1236
3	Communication and Presentations Skills Marks	1237
4	Linear Algebra Marks	1236
5	Pre-Calculus – I Marks	1187
6	Pre-Calculus – II Marks	1237
7	Probability and Statistics Marks	1236
8	Life and Living Marks	1237
9	Digital and Logic Design Marks	1236

4.1.2. Dropping Features

Handling missing data effectively is essential for improving model performance and ensuring data integrity. In cases where features contain an excessive amount of missing values, such as those listed in Table 3, imputation may not be feasible. Features with over 95% missing data are considered unreliable and are thus excluded from the dataset.

Python's Pandas library offers tools to efficiently drop these features from the dataset. Consequently, features such as mentioned in Table 4, have been removed from the analysis. These actions streamline the dataset by retaining only the most relevant and complete records for subsequent analysis.

4.1.3. Identification of Outliers in Data

Outliers are data points that significantly deviate from the general trend of a dataset, often referred to as anomalies, deviants, or discordant observations [43]. In the student academic records dataset, outliers are observed in the SSC Obtained Marks and HSSC Obtained Marks features. Other features, such as Introduction to Information and Communication, English Composition and Comprehension, Applied Physics, Discrete Structures, and Programming Fundamentals, were not found to contain outliers based on descriptive statistical analysis.

4.1.4. Outliers in SSC Obtained Marks

This feature displayed a negatively skewed distribution on the right tail and minimal positively skewed values on the left tail. While SSC marks should range from 0 to 1,050 or 1,100, the observed distribution ranged from 0 to 1,200 (Figure 5). Addressing these outliers is crucial for achieving a symmetrical distribution, which is essential for the accurate calculation of quartiles, mean, median, and mode.

4.1.5. Outliers in HSSC Obtained Marks

The HSSC obtained marks data exhibited outliers (Figure 6), with positive skewness on the left tail and negative skewness on the right. The observed data ranged from below 0 to 5,000, with an expected upper bound of 1,150. These outliers distort the frequency distribution and impact the interpretability of the mean, median, and mode.

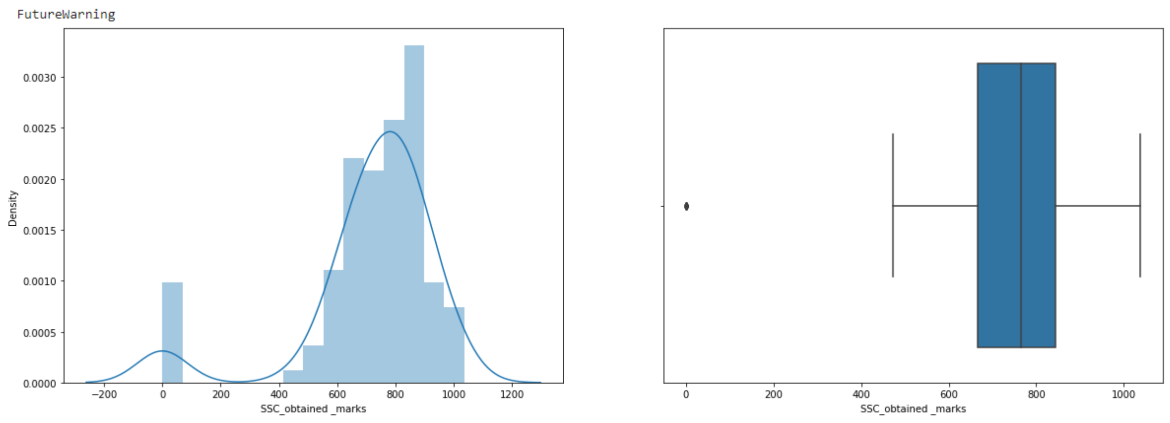

Figure 5. Attribute SSC Obtained Marks Outliers

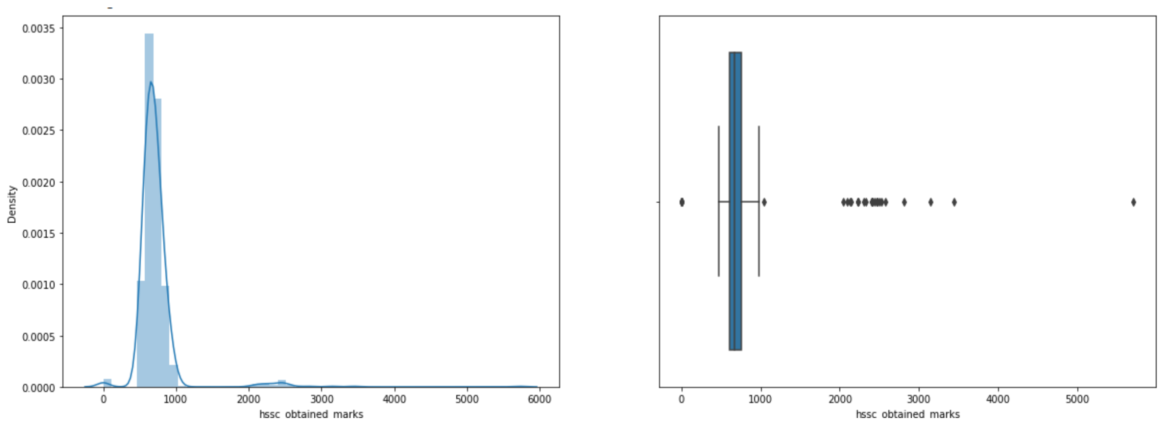

Figure 6. Attribute HSSC obtained marks outliers

4.1.6. Handling Outliers in SSC Obtained Marks

To address the outliers in the "SSC Obtained Marks" feature, we employed the following methodology. Utilized the quantile() function from the Python Pandas library to estimate value quantiles along the index axis. Determined the distribution of variable values, including lower and upper bounds. Initialized variables for the lower bound (LB = 0.1) and upper bound (UB = 0.95) effectively trimmed the data to the 10th-95th percentile range, as shown in Table 5.

Table 5. SSC Obtained Marks Data Distribution Range

Bounds	Range	SSC Obtained Marks Min/Max Data Range
Lower Bound	0.1	562
Upper Bound	0.95	968

Established criteria for "SSC Obtained Marks" feature values based on the determined ranges. Filtered true values and removed false values from the feature index. The resulting symmetric data distribution is illustrated in Figure 7, with the box plot quartile values (Minimum: 570, Q1: 685, Q2 (Median): 773, Q3: 841, Maximum: 967).

This approach effectively addressed the outliers, resulting in a more evenly distributed dataset suitable for further analysis.

4.1.7. Handling outliers in feature HSSC obtained marks

The HSSC Obtained Marks feature contains outliers that result in a skewed distribution, with significant differences between the mean, median, and mode. To mitigate this issue, the quantile() function from Python's Pandas library was used to identify and manage the outliers. This method estimates quantiles to define the boundaries for acceptable data ranges.

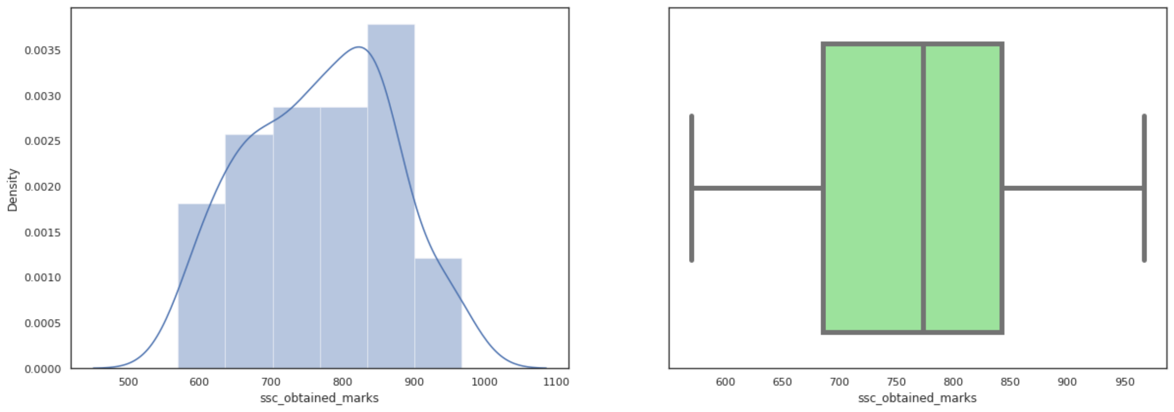

Figure 7. SSC obtained marks data distribution and box plot

Through quantitative analysis, the lower bound (LB) was set at the 10th percentile (0.1) and the upper bound (UB) at the 95th percentile (0.95). This approach removes extreme data points, retaining values within the 10-95% range of the distribution, as shown in Table 6.

Table 6. HSSC Obtained Marks Data Distribution Range

Bounds	**Range**	**SSC Obtained Marks Min/Max Data Range**
Lower Bound	0.1	565.0
Upper Bound	0.95	909.3

After filtering based on these bounds, outliers were removed, and the data exhibited a more balanced distribution. This is reflected in the box plot (Figure 8), with quartile values as follows: minimum = 566, Q1 = 628, Q2 = 685, Q3 = 755 and maximum = 909. The removal of outliers ensures a more accurate representation of the data.

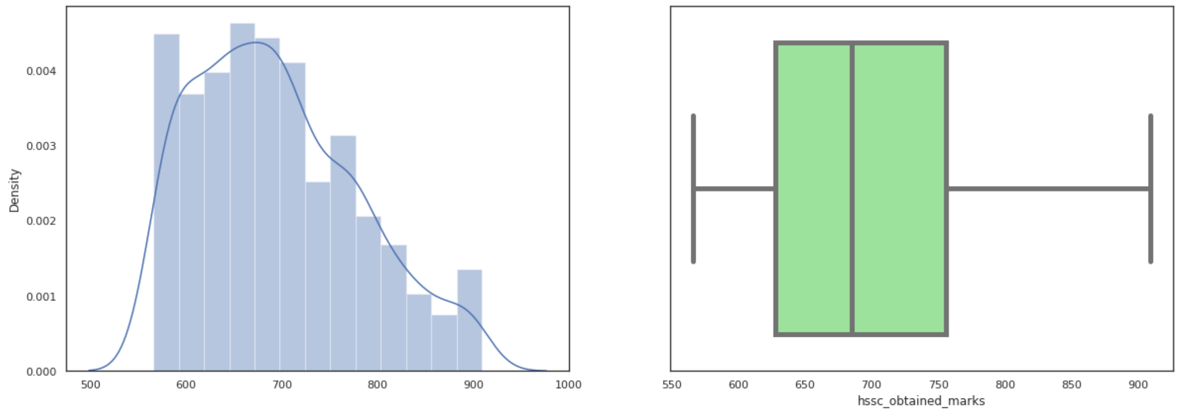

Figure 8. HSSC Marks data distribution and boxplot

4.2. Machine Learning Models for Student Performance Prediction

4.2.1. Train / Test Split Data Preparation

For all models, the dataset was split into 80% training and 20% testing sets, with a random state of 45 to ensure reproducibility.

4.2.2. KNR Model

The KNR model was optimized by iteratively testing K values up to 20. The optimal K value of 4 was determined based on the minimum mean squared error (MSE) of 0.0152. The final model achieved:

- Training accuracy (R2 score): 98%
- Testing accuracy (R2 score): 97%

The model's performance is visualized in Figure 9, showing actual vs. predicted programming fundamentals marks.

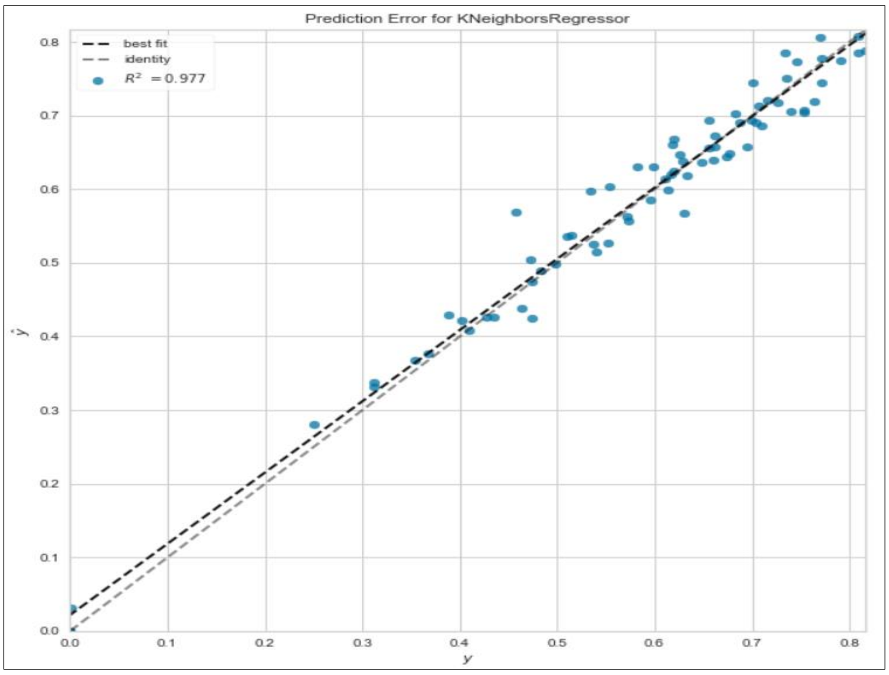

Figure 9. Regression line plot of actual and predicted programming fundamental scores for KNR

4.2.3. RFR model

The model achieved an R² score of 80% on the training set and 79% on the test set, with a mean squared error (MSE) of 0.00128%. The five regression slope coefficient values and interpretation are listed in Table 7.

Table 7. RFR Slope Coefficients& Influence

Independent Variable	Coefficient	Influence
ICT	0.294	Strongest predictor of programming performance
English Composition and Comprehension	0.239	Second strongest predictor
Applied Physics	0.231	Third strongest predictor
Discrete Structures	0.177	Fourth strongest predictor
SSC obtained marks	0.037	Weak positive influence
HSSC obtained marks	0.022	Weakest positive influence

All predictors showed positive influences on programming marks, with course-specific marks having a stronger impact than general academic performance (SSC and HSSC marks).

Furthermore, the actual y values (actual programming marks) and predicted (Ý) values (predicted programming marks) are plotted along with both regression lines, as shown in Figure 10.

4.2.4. GBR model

The GBR model achieved an R² score of 86% on the training set and 84% on the test set, with an MSE of 0.00740%. The five regression slope coefficient values and interpretation are listed in Table 8.

Similar to the RFR model, all predictors showed positive influences, with course-specific marks having a stronger impact than general academic performance.

4.2.5. LGBMR model

The model was initially trained with default parameters and then refined with specific parameter values to address overfitting. Cross-validation was implemented using RepeatedKFold with ten splits and three repeats.

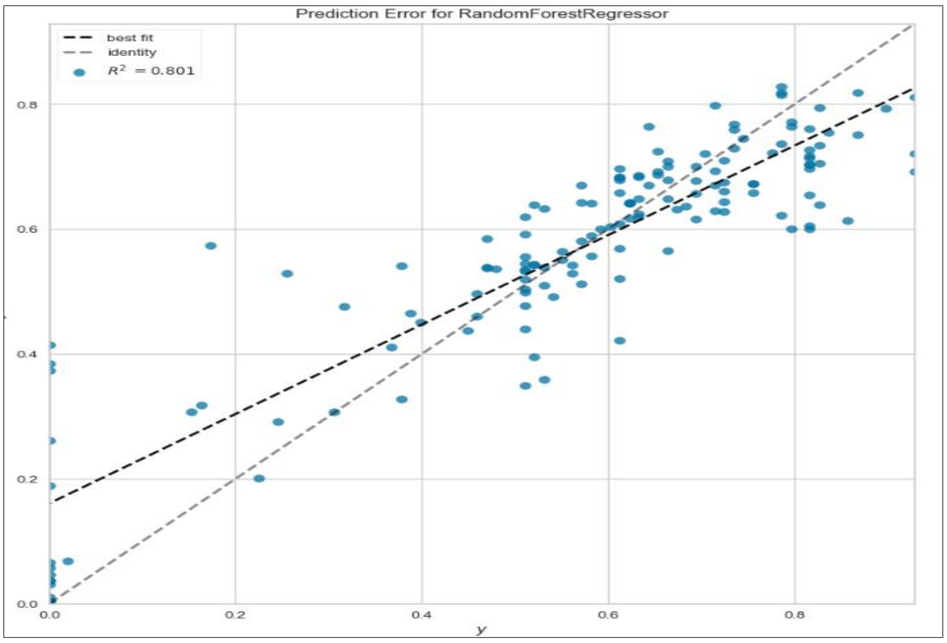

Figure 10. Regression line plot of actual and predicted programming fundamental scores for RFR

Table 8. GBR Slope Coefficients& Influence

Independent Variable	Coefficient	Influence
English Composition and Comprehension	0.382	Strongest predictor
ICT	0.247	Second strongest predictor
Applied Physics	0.221	Third strongest predictor
Discrete Structures	0.069	Fourth strongest predictor
SSC obtained marks	0.056	Weak positive influence
HSSC obtained marks	0.024	Weakest positive influence

Table 9. LGBMR Slope Coefficients& Influence

Independent Variable	Coefficient	Influence
SSC obtained marks	189	Strongest predictor
HSSC obtained marks	120	Second strongest predictor
ICT	100	Third strongest predictor
Applied Physics	92	Tied for fourth strongest predictor
Discrete Structures	92	Tied for fourth strongest predictor
English Composition and Comprehension	67	Weakest predictor

The LGBMR model achieved an R² score of 88% on the training set and 85% on the test set, with an MSE of 0.0067. The specific coefficient values and interpretation are presented in Table 9. The LGBMR model suggests that general academic performance (SSC and HSSC marks) has a stronger influence on programming performance than course-specific marks. Furthermore, the R2 prediction is visualized using a regression line that represents both the actual (Y) and predicted (Ý) programming fundamentals marks, as shown in Figure 11.

4.3. Comparison of ML Models with Evaluation Metrics

The performance of the four regression models (LGBMR, KNR, GBR, and RFR) was evaluated using the R² score and Mean Squared Error (MSE). Table 10 summarizes these metrics for each model. The K-Nearest Neighbors Regressor (KNR) demonstrated the highest predictive power with an R² train score of 98% and a test score of 97%. However, the Random Forest Regressor (RFR) exhibited the lowest Mean Squared Error (0.001), indicating the highest precision in its predictions.

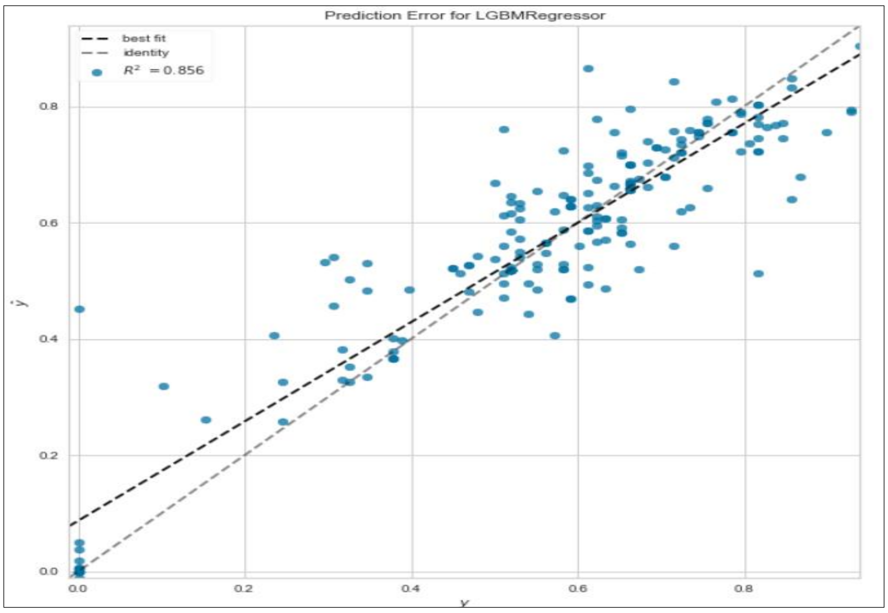

Figure 11. Light GBM Regressor, regression line plot of actual and predicted programming fundamental scores

Table 10. Models' evaluation with regression metrics

ML Models	R^2 Train Score	R^2 Test Score	Mean Squared Error
LGBMR	88%	85%	0.007
KNR	98%	97%	0.008
GBR	86%	84%	0.007
RFR	80%	79%	0.001

4.4. Evaluating the Effectiveness of Course Predictors

The potential of various courses to predict different skill sets was analyzed. Table 11 presents the estimated predictive power of each course for cognitive, natural language, and mathematics skills.

Table 11. Courses potential based on predictors' estimated results

Courses	Objectives	Cognitive	Natural Language	Mathematics
Applied Physics	Goals students should be able to understand the...	0.278	0.122	0.598
English Composition and Comprehension	To enable students to identify the main topic ...	0.517	0.350	0.132
Introduction to Communication and Presentation Skills	To develop among students the skills necessary...	0.671	0.270	0.057
Programming Fundamental	Goal students should be able to implement in the...	0.618	0.228	0.152
Discrete Structures	To develop an understanding of logic sets and fun...	0.213	0.098	0.688

Noted findings:

Introduction to Communication and Presentation Skills, Programming Fundamentals, and English Composition and Comprehension show strong potential for predicting cognitive skills.

English Composition and Comprehension demonstrates the highest potential for predicting natural language skills.

Applied Physics and Discrete Structures exhibit the strongest potential for predicting mathematics skills.

5. DISCUSSION

5.1. Key Findings

The K-Nearest Neighbors Regressor (KNR) demonstrated the best overall performance with R^2 scores of 98% (train) and 97% (test). However, the Random Forest Regressor (RFR) showed the lowest Mean Squared Error (0.001), indicating high precision.

Our analysis revealed that cognitive skills were the most crucial for learning programming, accounting for 62% of the required aptitude. Natural language and mathematics skills contributed 24% and 14%, respectively. This finding challenges the assertion by [31] that mathematics is not an essential predictor of programming proficiency.

The course predictors for the courses Introduction to Communication and Presentation Skills, Programming Fundamentals, and English Composition and Comprehension showed strong potential for predicting cognitive skills. Applied Physics and Discrete Structures demonstrated high potential for predicting mathematics skills.

5.2. Implications

Our findings suggest that a balanced approach incorporating cognitive, linguistic, and mathematical skills development is crucial for effective programming education. This challenges traditional curriculum designs that may overemphasize mathematical skills at the expense of cognitive and linguistic development.

By identifying key aptitudes, educators can develop targeted interventions to enhance students' programming skills. This could potentially improve retention rates and overall success in computing programs. The predictive model developed in this study could inform more effective admissions criteria for computer science programs, helping to identify students with the highest potential for success in programming courses.

Regarding the course-specific focus on predicting performance in Programming Fundamentals, our methodology demonstrated broader applicability across various courses in the computer science curriculum. Our findings revealed that different courses served as effective predictors for specific aptitudes: Introduction to Communication and Presentation Skills, Programming Fundamentals, and English Composition and Comprehension showed strong potential for predicting cognitive skills, English Composition and Comprehension also shows higher potential for predicting natural language skills, while Applied Physics and Discrete Structures demonstrated high potential for predicting mathematics skills. The successful implementation of our Machine Learning and Natural Language Processing approach suggests that this framework could be effectively adapted to predict student performance in other courses.

6. CONCLUSION

This study aimed to identify the essential aptitudes for successfully learning programming languages, addressing a fundamental question in computing education. Our findings reveal that cognitive skills play the most crucial role (62%), followed by natural language (24%) and mathematics skills (14%), challenging previous assertions about the relative importance of these aptitudes. The K-Nearest Neighbors Regressor demonstrated the best overall performance in predicting student success, while specific courses showed strong potential for predicting various aptitudes. These insights have significant implications for curriculum design, student support strategies, and admissions criteria in computer science education, suggesting a need for a balanced approach that incorporates cognitive, linguistic, and mathematical skills development. While limited by its focus on a single institution and specific courses, this study provides a foundation for future research, including cross-institutional and longitudinal studies. By employing a novel approach combining Machine Learning and Natural Language Processing techniques, we have developed a methodology that could be implemented in other institutions to predict student performance and inform curriculum design.

DATA AVAILABILITY STATEMENT

As the study uses the academic records from Riphah International University, it is not possible to make the data available due to the privacy policy of the university.

ACKNOWLEDGEMENTS

As this research was conducted as part of the MS Thesis at Riphah International University so, the authors would like to thank the Department of Computer Science and Information Technology, the Faculty, and the Head of the Department, who permitted the use of the university's dataset for this research work. Also, the authors would like to extend their gratitude to the programme coordinator, who allotted and helped gather and give sequence/detail about the student's record.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest to this work.

REFERENCES

[1] J. Figueiredo and F. Garcia-Penalvo, "Teaching and Learning Tools for Introductory Programming in University Courses," *SIIE 2021 - 2021 Int. Symp. Comput. Educ.*, no. September, 2021, doi: 10.1109/SIIE53363.2021.9583623.

[2] L. T. Yong, C. Y. Qi, C. S. Yee, A. Johnson, and N. K. Hoong, "Designing and Developing a PDA Food Ordering System Using Interaction Design Approach: A Case Study," in *2009 International Conference on Computer Technology and Development*, 2009, pp. 68–71. doi: 10.1109/ICCTD.2009.18.

[3] I. Milne and G. Rowe, "Difficulties in learning and teaching programming - Views of students and tutors," *Educ. Inf. Technol.*, vol. 7, no. 1, pp. 55–66, 2002, doi: 10.1023/A:1015362608943.

[4] M. N. Ismail, N. A. Ngah, and I. N. Umar, "Instructional strategy in the teaching of computer programming: A need assessment analyses," *Turkish Online J. Educ. Technol.*, vol. 9, no. 2, pp. 125–131, 2010.

[5] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, *Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming*, vol. 1, no. 1. Association for Computing Machinery, 2023. doi: 10.1145/3544548.3580919.

[6] R. Yilmaz and F. G. Karaoglan Yilmaz, "Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning," *Comput. Hum. Behav. Artif. Humans*, vol. 1, no. 2, p. 100005, 2023, doi: 10.1016/j.chbah.2023.100005.

[7] Y. F. Liu, J. Kim, C. Wilson, and M. Bedny, "Computer code comprehension shares neural resources with formal logical inference in the fronto-parietal network," *Elife*, vol. 9, pp. 1–22, 2020, doi: 10.7554/eLife.59340.

[8] P. J. Rich, S. L. Mason, and J. O'Leary, "Measuring the effect of continuous professional development on elementary teachers' self-efficacy to teach coding and computational thinking," *Comput. Educ.*, vol. 168, no. March, 2021, doi: 10.1016/j.compedu.2021.104196.

[9] B. Helmlinger, M. Sommer, M. Feldhammer-Kahr, G. Wood, M. E. Arendasy, and S. E. Kober, "Programming experience associated with neural efficiency during figural reasoning," *Sci. Rep.*, vol. 10, no. 1, pp. 1–14, 2020, doi: 10.1038/s41598-020-70360-z.

[10] R. Asif, A. Merceron, S. A. Ali, and N. G. Haider, "Analyzing undergraduate students' performance using educational data mining," *Comput. Educ.*, vol. 113, pp. 177–194, 2017, doi: 10.1016/j.compedu.2017.05.007.

[11] J. Köhler, L. Hidalgo, and J. L. Jara, "Predicting Students' Outcome in an Introductory Programming Course: Leveraging the Student Background," *Appl. Sci.*, vol. 13, no. 21, 2023, doi: 10.3390/app132111994.

[12] M. Endres, M. Fansher, P. Shah, and W. Weimer, "To read or to rotate? comparing the effects of technical reading training and spatial skills training on novice programming ability," *ESEC/FSE 2021 - Proc. 29th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 754–766, 2021, doi: 10.1145/3468264.3468583.

[13] M. Endres, Z. Karas, X. Hu, I. Kovelman, and W. Weimer, "Relating reading, visualization, and coding for new programmers: A neuroimaging study," *Proc. - Int. Conf. Softw. Eng.*, pp. 600–612, 2021, doi: 10.1109/ICSE43902.2021.00062.

[14] A. Zavgorodniaia, A. Hellas, O. Seppälä, and J. Sorva, "Should Explanations of Program Code Use Audio, Text, or Both? A Replication Study," *ACM Int. Conf. Proceeding Ser.*, vol. 2020, pp. 1–10, 2020, doi: 10.1145/3428029.3428050.

[15] Y. Kao, B. Matlen, and D. Weintrop, "From One Language to the Next: Applications of Analogical Transfer for Programming Education," *ACM Trans. Comput. Educ.*, vol. 22, no. 4, 2022, doi: 10.1145/3487051.

[16] M. Endres, W. Weimer, and A. Kamil, "An Analysis of Iterative and Recursive Problem Performance," *SIGCSE 2021 - Proc. 52nd ACM Tech. Symp. Comput. Sci. Educ.*, pp. 321–327, 2021, doi: 10.1145/3408877.3432391.

[17] J. Jeuring, R. Groot, and H. Keuning, "What Skills Do You Need When Developing Software Using ChatGPT? (Discussion Paper)," *ACM Int. Conf. Proceeding Ser.*, pp. 1–11, 2023, doi: 10.1145/3631802.3631807.

[18] S. Rajendran, S. Chamundeswari, and A. A. Sinha, "Predicting the academic performance of middle- and high-school students using machine learning algorithms," *Soc. Sci. Humanit. Open*, vol. 6, no. 1, p. 100357, 2022, doi: 10.1016/j.ssaho.2022.100357.

[19] S. Srikant, C. Science, T. Supervisor, L. Kolodziejski, and C. Science, "Understanding Computer Programs : Computational and Cognitive Perspectives by," no. 2011, 2023.

[20] C. H. Kuo, M. Mottarella, T. Haile, and C. S. Prat, "Predicting Programming Success: How Intermittent Knowledge Assessments, Individual Psychometrics, and Resting-State EEG Predict Python Programming and Debugging Skills," *2022 30th Int. Conf. Software, Telecommun. Comput. Networks, SoftCOM 2022*, 2022, doi: 10.23919/SoftCOM55329.2022.9911411.

[21] E. H. Brain and O. F. C. Programsthe, "Representations of Computer Programs in the Human Brain," pp. 1–30, 2022.

[22] C. Angeli and M. Giannakos, "Computational thinking education: Issues and challenges," *Comput. Human Behav.*, vol. 105, p. 106185, Apr. 2020, doi: 10.1016/J.CHB.2019.106185.

[23] S. Kılıç, S. Gökoğlu, and M. Öztürk, "A Valid and Reliable Scale for Developing Programming-Oriented Computational Thinking," *J. Educ. Comput. Res.*, vol. 59, no. 2, pp. 257–286, 2021, doi: 10.1177/0735633120964402.

[24] R. Scherer, F. Siddiq, and B. Sánchez-Scherer, "Some Evidence on the Cognitive Benefits of Learning to Code," *Front. Psychol.*, vol. 12, no. September, pp. 1–5, 2021, doi: 10.3389/fpsyg.2021.559424.

[25] C. H. Kuo and C. S. Prat, "Computer programmers show distinct, expertise-dependent brain responses to

violations in form and meaning when reading code," *Sci. Rep.*, vol. 14, no. 1, 2024, doi: 10.1038/s41598-024-56090-6.

[26] E. Fedorenko, A. Ivanova, R. Dhamala, and M. U. Bers, "The Language of Programming: A Cognitive Perspective," *Trends Cogn. Sci.*, vol. 23, no. 7, pp. 525–528, 2019, doi: 10.1016/j.tics.2019.04.010.

[27] J. Agarwal, G. W. Bucks, K. A. Ossman, T. J. Murphy, and C. E. Sunny, "Learning a Second Language and Learning a Programming Language: An Exploration," *ASEE Annu. Conf. Expo. Conf. Proc.*, 2021, doi: 10.18260/1-2--37423.

[28] D. H. Smith, Q. Hao, F. Jagodzinski, Y. Liu, and V. Gupta, "Quantifying the Effects of Prior Knowledge in Entry-Level Programming Courses," in *CompEd 2019 - Proceedings of the ACM Conference on Global Computing Education*, 2019. doi: 10.1145/3300115.3309503.

[29] G. Barlow-Jones and D. van der Westhuizen, "Problem solving as a predictor of programming performance," in *Communications in Computer and Information Science*, 2017. doi: 10.1007/978-3-319-69670-6_14.

[30] D. Cukierman, "Predicting success in university first year computing science courses: The role of student participation in reflective learning activities and in I-clicker activities," in *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 2015, pp. 248–253. doi: 10.1145/2729094.2742623.

[31] C. S. Prat, T. M. Madhyastha, M. J. Mottarella, and C. H. Kuo, "Relating Natural Language Aptitude to Individual Differences in Learning Programming Languages," *Sci. Rep.*, vol. 10, no. 1, pp. 1–10, 2020, doi: 10.1038/s41598-020-60661-8.

[32] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," *Int. J. Comput. Inf. Sci.*, vol. 8, no. 3, pp. 219–238, 1979, doi: 10.1007/BF00977789.

[33] V. J. Shute, "Who is Likely to Acquire Programming Skills?," *J. Educ. Comput. Res.*, vol. 7, no. 1, pp. 1–24, 1991, doi: 10.2190/vqjd-t1yd-5wvb-rypj.

[34] Y. Ao, H. Li, L. Zhu, S. Ali, and Z. Yang, "The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling," 2019. doi: 10.1016/j.petrol.2018.11.067.

[35] V. Rodriguez-Galiano, M. Sanchez-Castillo, M. Chica-Olmo, and M. Chica-Rivas, "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines," *Ore Geol. Rev.*, vol. 71, pp. 804–818, Dec. 2015, doi: 10.1016/J.OREGEOREV.2015.01.001.

[36] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324/METRICS.

[37] J. Peters *et al.*, "Random forests as a tool for ecohydrological distribution modelling," *Ecol. Modell.*, vol. 207, no. 2–4, pp. 304–318, 2007, doi: 10.1016/j.ecolmodel.2007.05.011.

[38] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4, no. 4. Springer, 2006.

[39] A. Keprate and R. M. C. Ratnayake, "Using gradient boosting regressor to predict stress intensity factor of a crack propagating in small bore piping," *IEEE Int. Conf. Ind. Eng. Eng. Manag.*, vol. 2017-Decem, no. December, pp. 1331–1336, 2017, doi: 10.1109/IEEM.2017.8290109.

[40] J. Brownlee, "A gentle introduction to the gradient boosting algorithm for machine learning," *Mach. Learn. Mastery*, vol. 21, 2016.

[41] N. S. Zheng, X. W. Jiang, Y. Ao, and X. Zhao, "Prediction of tariff package model using ROF-LGB algorithm," *ACM Int. Conf. Proceeding Ser.*, pp. 54–58, 2019, doi: 10.1145/3352411.3352421.

[42] L. Zhang, T. Xiang, and S. Gong, "Learning a deep embedding model for zero-shot learning," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 3010–3019, 2017, doi: 10.1109/CVPR.2017.321.

[43] C. C. Aggarwal, "An Introduction to Outlier Analysis," *Outlier Anal.*, pp. 1–34, 2017, doi: 10.1007/978-3-319-47578-3_1.

BIOGRAPHIES OF AUTHORS

Muhammad Faisal Iqbal is a data science professional. I hold a Bachelor's degree in Computer Science from Sarhad University of Sciences & Information Technology and a Master's degree in Data Science from Riphah International University, both in Islamabad, Pakistan. My professional experience spans various domains, including Oracle Financial Applications, Database Development, Data Science, Machine Learning, Deep Learning, Natural Language Processing, and Multimodals. I have also provided training on software utilities and version releases to both private and government organizations. My research interests align with the cutting-edge fields of Data Science, Machine Learning, Deep Learning, Natural Language Processing, Multimodal, Generative AI, and Large Language Models. He can be contacted at email: muhammadfaisal.softech@gmail.com

Adeel Zafar is currently working as an associate professor / Head of Department at Riphah Institute of System Engineering, Riphah International University Islamabad, Pakistan. He can be contacted at email: adeel.zafar@riphah.edu.pk.

Umer Khalil is an enthusiastic and adaptive professional with a passion for enhancing his skills and contributing to innovative projects. With a strong foundation in remote sensing, geoinformatics (GIS), and civil engineering, Umer specializes in urban systems, geospatial mapping, and environmental sustainability. Umer is currently working as a GIS engineer in a tech company. His expertise spans a range of interdisciplinary fields, including urban and regional planning, smart city development, geospatial data analysis, machine learning applications, hazard and risk assessment, and addressing wicked problems in environmental and urban contexts. He can be contacted at email: umerkhalil745@gmail.com

Afia Ishaq is currently working as a Lecturer at Riphah Institute of System Engineering, Riphah International University Islamabad, Pakistan. She can be contacted at email: afiaishaq21@gmail.com