www.ijarp.com ISSN 2456-9992 Page: 01-08

International Journal Advanced Research Publications

Research Article

Volume: 02
Issue: 04

HARNESSING DATA TO ENHANCE STUDENT PERFORMANCE

Sriram N.*[1], Mrs. P. Shanthi M.C.A.[1], M.Phil., M.Sc., NET[2]

[1]Student, Department of Artificial Intelligence and Machine Learning,
Dr. N.G.P. Arts and Science College (Autonomous), Affiliated to Bharathiar University,
Coimbatore-641 048, Tamil Nadu, India.

[2]Assistant Professor, Department of Artificial Intelligence and Machine Learning,
Dr. N.G.P. Arts and Science College (Autonomous), Coimbatore-641 048, Tamil Nadu, India.

Article Received: 14 March 2026, Article Revised: 03 April 2026, Published on: 23 April 2026

***Corresponding Author: Sriram N.**
Student, Department of Artificial Intelligence and Machine Learning, Dr. N.G.P. Arts and Science College (Autonomous), Affiliated to Bharathiar University, Coimbatore-641 048, Tamil Nadu, India.
DOI: https://doi-doi.org/101555/ijarp.3554

ABSTRACT

Managing and analyzing student academic data has become increasingly important for institutions striving to improve educational outcomes. This paper presents a Student Performance Enhancement System — a full-stack web application that digitizes the collection, processing, storage, and analysis of student academic records. The system integrates React for a dynamic frontend, FastAPI for a high-performance Python backend, and PostgreSQL as a structured relational database. Student information and subject marks are entered through an intuitive interface, and the backend automatically calculates totals and averages while storing records securely. A machine learning component employing Random Forest Regression evaluates and predicts student performance trends with up to 95% accuracy and a low error rate of 5%. The system eliminates manual record-keeping errors, ensures centralized data access, and supports data-driven decision-making for educators. Testing across functional, validation, and database scenarios confirmed robust performance and reliability. The system is well-suited for academic institutions seeking a scalable, secure, and efficient solution for student performance management.

KEYWORDS: Student Performance, Academic Records Management, Machine Learning, Random Forest Regression, FastAPI, React, PostgreSQL, Data Analytics, Educational Technology, Full-Stack Web Application.

1. INTRODUCTION

In the modern educational environment, effective management of student academic records plays a vital role in ensuring transparency, accuracy, and efficiency within institutions. Academic institutions generate a significant volume of data every semester — including student personal information, subject marks, internal assessments, and final examination results. Traditionally, these records have been maintained manually using physical registers or spreadsheet files, presenting several limitations in terms of scalability, data security, and accuracy.

Manual record-keeping systems are time-consuming and prone to human error. Calculating total marks and averages manually increases the risk of arithmetic mistakes, and retrieving specific student records from large batches requires considerable effort. Spreadsheet-based systems offer slight improvement, but still lack centralized access, proper validation mechanisms, and secure authentication controls.

With the rapid advancement of web technologies and database management systems, educational institutions are gradually adopting digital solutions to improve administrative efficiency. A web-based application provides centralized data storage, automated calculations, improved security, and quick retrieval of information.

The proposed system — Harnessing Data to Enhance Student Performance — is developed to overcome the drawbacks of traditional record-keeping. It provides a digital platform where student details and subject marks can be entered, processed, stored, and retrieved efficiently. The system ensures accurate calculation of totals and averages while maintaining data integrity within a structured relational database. By integrating React, FastAPI, and PostgreSQL with a machine learning prediction engine, the system delivers a scalable and maintainable solution for managing academic records effectively.

The rest of this paper is organized as follows: Section 2 reviews the existing system and motivates the proposed approach. Section 3 describes the system architecture and module design. Section 4 presents hardware and software requirements. Section 5 discusses implementation details. Section 6 presents results and comparative analysis. Section 7 concludes with future directions.

2. Literature Review

2.1 Web-Based Academic Record Systems

Earlier academic record management systems relied on standalone desktop applications or paper-based registers. Research by Laudon & Laudon (2020) highlights that centralized

database-driven systems outperform manual methods in terms of retrieval speed, accuracy, and scalability [1]. The shift to web-based architectures has accelerated with the adoption of REST APIs and lightweight frontend frameworks.

2.2 Machine Learning in Educational Data Mining

Educational data mining has grown into a recognized discipline that applies machine learning techniques to academic datasets to predict performance and identify at-risk students. Romero & Ventura (2010) conducted a comprehensive survey demonstrating that decision tree and ensemble methods consistently outperform traditional statistical approaches on student grade prediction tasks [2].

2.3 Random Forest for Performance Prediction

Random Forest is an ensemble learning method that aggregates predictions from multiple decision trees trained on random subsets of features and data. Breiman (2001) introduced the technique and demonstrated its robustness against overfitting [3]. In educational contexts, Kotsiantis et al. (2007) confirmed that Random Forest consistently achieves higher accuracy than support vector machines and naive Bayes classifiers on student performance datasets [4].

2.4 RESTful API Design and Modern Backend Frameworks

FastAPI, introduced by Ramírez (2018), is a high-performance Python framework built on ASGI and Pydantic, enabling automatic OpenAPI documentation and type-safe request validation [5]. Its asynchronous request handling makes it significantly faster than synchronous frameworks such as Flask for I/O-bound workloads, which is critical when the backend must serve concurrent student record queries.

2.5 Limitations of Existing Systems

Most existing student record management tools either rely on static spreadsheets lacking validation, or are heavyweight enterprise resource planning systems that are costly to deploy and maintain. No lightweight open-source system integrates automated mark computation, relational database storage, and machine-learning-based performance prediction in a unified full-stack web application accessible to smaller institutions.

3. System Architecture and Design

The proposed system follows a three-tier client-server architecture: a presentation layer built with React, an application layer built with FastAPI, and a data layer implemented in PostgreSQL. Each tier communicates only with its immediate neighbors, ensuring separation of concerns and making the design easy to maintain and extend.

When a user submits student information or marks through the frontend, an HTTP request is

dispatched to the FastAPI backend. The backend validates the incoming data, performs business logic operations such as calculating totals and averages, persists the record in PostgreSQL, and returns a JSON response to the frontend, which renders the result without a page reload.

3.1 Module 1 — Student Data Input Module

The input module provides forms for entering student identifiers (ID, name, class, roll number) and subject-wise marks. Client-side validation prevents submission of incomplete records. Server-side validation enforced by FastAPI's Pydantic models provides a second layer of integrity checking, rejecting marks that exceed the maximum allowed value before any database write occurs.

3.2 Module 2 — Backend Processing Module

The FastAPI backend exposes a RESTful API with endpoints for creating, reading, updating, and deleting student and marks records. On receipt of a marks submission, the backend calculates the total marks and percentage average, appends these computed fields to the record, and returns the enriched object to the frontend. All endpoints are documented automatically via the built-in Swagger UI.

3.3 Module 3 — Database Module

PostgreSQL stores student records in two normalized tables: a Students table (student_id, name, class, roll_no) and a Marks table (mark_id, student_id [FK], subject, marks_obtained, max_marks). SQLAlchemy serves as the ORM layer, mapping Python class definitions to database schemas and managing connection pooling. Environment variables store database credentials to prevent hard-coding sensitive information.

3.4 Module 4 — Machine Learning Prediction Module

A Random Forest Regression model is trained on historical marks data to predict a student's likely final percentage given their mid-term and internal assessment scores. The model is serialized using joblib and loaded once at application startup. Prediction requests are processed synchronously, with typical inference latency under 10 milliseconds. The trained model achieves 95% accuracy and a 5% error rate on held-out test data.

3.5 Module 5 — Result Display Module

The React frontend renders student records in a tabular layout with sortable columns. A dedicated analytics panel displays the predicted performance score alongside the actual computed average, allowing teachers to quickly identify students whose predicted trajectory diverges significantly from their current standing. Each row includes action buttons for editing and deleting records.

4. System Requirements

4.1 Hardware Requirements

Component	Specification
Processor	Intel i3 minimum; i5 recommended
RAM	4 GB minimum; 8 GB recommended
Storage	1–2 GB for application and dependencies
Network	Required for initial setup and web access

4.2 Software Requirements

Library / Tool	Purpose
React (JavaScript)	Frontend UI framework
FastAPI (Python)	Backend REST API server
PostgreSQL	Relational database
SQLAlchemy	ORM and database abstraction
scikit-learn	Random Forest ML model
Pandas / NumPy	Data preprocessing
Uvicorn	ASGI server for FastAPI
Node.js / npm	Frontend build and dev server

5. System Implementation

The backend is organized as a Python package with one file per module, making it straightforward to replace individual components — for instance, substituting the Random Forest model with a gradient boosting variant — without modifying the rest of the codebase. The FastAPI application exposes RESTful endpoints that accept and return JSON. All database models are loaded once at server startup through SQLAlchemy's declarative base, and connection pooling ensures efficient database access under concurrent load.

5.1 API Endpoint Structure

Method	Endpoint	Description
POST	/students	Create a new student record
GET	/students	Retrieve all student records
POST	/marks	Submit subject marks for a student
GET	/marks/{id}	Retrieve marks for a specific student

POST	/predict	Get performance prediction for a student

6. RESULTS AND DISCUSSION

The system was tested across two representative scenarios: a class of 30 students with five subjects each entered through the web interface, and a batch import of 100 historical student records used to evaluate machine learning prediction accuracy. In both cases, the full pipeline completed without errors and produced consistent outputs across all stages — data entry, validation, storage, computation, and prediction.

6.1 Algorithm Comparison

Algorithm	**Accuracy (%)**	**Processing Time (ms)**	**Error Rate (%)**	**Performance**
Linear Regression	82	120	18	Moderate
Support Vector Machine	88	180	12	Good
Decision Tree	91	95	9	Very Good
Random Forest Regression	95	110	5	Excellent

6.2 Comparative Analysis

Feature	**Existing Systems**	**Proposed System**
Automated Mark Calculation	Manual / Formula-based	Yes (FastAPI backend)
Centralized Data Storage	Local spreadsheets	Yes (PostgreSQL)
Input Validation	None / Limited	Yes (Pydantic models)
Performance Prediction	No	Yes (Random Forest 95%)
Web-Based Access	Limited	Yes (React frontend)
Data Security	Low	Environment-secured credentials
Scalability	Low	High (modular architecture)

6.3 DISCUSSION

The results confirm that combining a modern full-stack web framework with an ensemble machine learning model is not only feasible on standard hardware but practical for everyday academic use. The Random Forest Regression model significantly outperforms all other evaluated algorithms, achieving 95% accuracy and only a 5% error rate — outperforming Decision Tree (91%), SVM (88%), and Linear Regression (82%) — while maintaining a fast

processing time of 110 ms.

The primary bottleneck in the current implementation is the initial database query when retrieving large batch records. Indexing the student_id foreign key in the Marks table and implementing lazy loading on the frontend would substantially reduce retrieval latency for institutions with thousands of student records. Additionally, frontend pagination can be introduced to handle large datasets without impacting page render times.

Data validation testing confirmed that the Pydantic-enforced backend correctly rejects marks exceeding the maximum allowed value, preventing data corruption. All functional test cases for CRUD operations on both the Students and Marks tables passed without error.

7. CONCLUSION

This paper has presented a full-stack Student Performance Enhancement System that integrates React, FastAPI, and PostgreSQL with a Random Forest Regression prediction engine to automate academic record management and deliver data-driven insights into student performance. The system eliminates manual record-keeping errors, ensures centralized and secure data storage, and provides educators with an actionable performance prediction tool that achieves 95% accuracy.

Testing confirmed that the system handles real-world academic data reliably, with robust validation, consistent database operations, and fast API response times. The modular architecture makes it straightforward to upgrade individual components — for instance, replacing the prediction model with a deep learning approach or adding a cloud deployment layer — without disrupting existing functionality.

Future work will focus on three main areas: integrating graphical analytics dashboards so that teachers can visualize class-level performance trends at a glance; adding role-based authentication to differentiate between administrative staff, faculty, and student views; and extending the machine learning module to incorporate LSTM-based temporal models that track individual student performance trajectories across multiple semesters. Integration with institutional learning management systems is also planned to enable seamless data exchange.

ACKNOWLEDGEMENT

The author sincerely thanks Mrs. P. Shanthi M.C.A., M.Phil., M.Sc., NET, Assistant Professor, Department of Artificial Intelligence and Machine Learning, Dr. N.G.P. Arts and Science College, Coimbatore, for her continuous guidance and encouragement throughout this work. Gratitude is also expressed to Dr. S. Saranya, Associate Professor and Head of

Department, and to all faculty members of the department for their support. The author acknowledges the open-source communities behind React, FastAPI, PostgreSQL, and scikit-learn, whose freely available tools made this work possible.

REFERENCES

1. K. C. Laudon and J. P. Laudon, Management Information Systems: Managing the Digital Firm, 16th ed. Pearson, 2020.
2. C. Romero and S. Ventura, "Educational data mining: A review of the state of the art," IEEE Transactions on Systems, Man, and Cybernetics, Part C, vol. 40, no. 6, pp. 601–618, 2010.
3. L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
4. S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, "Machine learning: A review of classification and combining techniques," Artificial Intelligence Review, vol. 26, no. 3, pp. 159–190, 2007.
5. S. Ramírez, "FastAPI: Modern, Fast Web Framework for Building APIs with Python," 2018. [Online]. Available: https://fastapi.tiangolo.com
6. D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Stanford University, 2023.
7. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.
8. T. M. Mitchell, Machine Learning. McGraw-Hill Education, 1997.