EduGraph

Cristobal, Jhune Anderson
Condesa, Andrie Galicia
Desalit, Hanz Lorenz D.
Olmoguez, Zydric Angelo P.
Lacsamana, Timothy Justin P.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

PROJECT TITLE.. 1
Table of Contents.. 2
Introduction.. 3
The Project..4
Objectives..4
Flowchart of the System..5
Pseudocode..7
Data Dictionary..8
Code.. 10
Results and Discussion..14
Conclusion..14
References..14

Introduction

Students often struggle to track, interpret, and predict their academic performance during a grading period. Traditional grading systems and learning platforms make available limited, delayed, or unclear feedback that prevents a learner from being informed about their current standing or what scores they need for achieving the target final grade.

This means students have to manually calculate their grades using spreadsheets or calculators-a rather time-consuming and error-prone process with limited precision or transparency. In addition, without effective visualization or predictive guidance, learners are unable to effectively set realistic academic goals, manage their study efforts, or realize early signs of academic risk.

These issues point out how there is a lack of real-time, transparent, and predictive grade management that would track current performance and could forecast required performance in coming evaluations.

Delayed Feedback and Limited Awareness
Since students receive updates on their academic standing only after major exams or grading periods, they cannot find out weaknesses in time and improve on those areas.

Lack of Transparency in Grade Computation
Many students are not aware of how each of the coursework components, including quizzes, exams, and projects, contribute to their final grades; this results in confusion and grade misinterpretation.

Manual and error-prone calculations
Without an automated system, students compute their grades manually by calculators or spreadsheets, where mistakes can easily be made and result in incorrect academic expectations.

Absence of Predictive Guidance
Most of the current grade monitoring systems rarely provide predictive insights, such as how much one needs in upcoming assessments to obtain a desired final grade, which leads to poor goal-setting and planning.

Poor Visualization of Academic Progress
In this regard, traditional grading systems show data in the form of static tables or lists; this obscures the possibility of viewing trends, progress over time, or how specific grades may impact overall performance.

Limited Motivation and Academic Self-Regulation
Due to a lack of specific feedback and useful insights, students may get demotivated or be unable to mobilize their learning strategies in ways that would enhance their learning outcomes.

Fragmented Access to Academic Information
Many students' grade data are dispersed among different teachers, platforms, or subjects, thus making it hard to maintain a unified view of academic performance.

EduGraph is proposed as an interactive grade prediction and visualization system to address the issues presented in traditional grading schemes, aimed at increasing students' awareness, motivation, and academic decision-making.

EduGraph is an intelligent academic companion wherein students can input their current grades, respective weights, and target final grade, automatically generating the current weighted performance and how many points they need in future assessments to reach their target. This system will save them from having to compute scores manually and, at the same time, provide real predictive guidance throughout the process.

Along with computation, EduGraph also embeds data visualization tools like line or bar graphs, similar to Desmos, showing current performance versus assessment weights and target outcomes. This will help students better understand grade trends and which assessment weighs more in the standing of the student.

Furthermore, EduGraph empowers learners by showing them exactly how their efforts translate into final grades, encouraging motivation and greater transparency. Through its user-friendly interface and analytical feedback, it supports self-regulated learning, timely decision-making, and early intervention for at-risk students.

Overall, EduGraph transforms the grading experience from a static, delayed system into a dynamic, goal-oriented platform that empowers students to take control of their academic journey.

The Project

Objectives

EduGraph is an intelligent, interactive grade prediction and visualization system that allows students to monitor their academic performance, compute required scores for future assessments, and visualize their progress in real time toward achieving their desired final grade.

1. To design and implement an automated grade computation system
   This program calculates the current weighted grade for a student exactly based on entered scores and corresponding assessment weights, reducing manual calculation errors to a minimum.
2. To develop a predictive algorithm
   which calculates the necessary score on future assessments to obtain a user-specified desired final grade.
3. To provide real-time performance feedback
   It displays the current academic standing and updates it after every input of new assessment results.
4. To integrate data visualization tools
   produce interactive graphs showing current grades, required future performance, and “what-if” projections that will help students understand grade trends.
5. To enhance student awareness and motivation

by providing clear, transparent, and actionable information on their own academic progress, thus supporting goal-setting and self-regulated learning.

6. To ensure the accuracy and usability of the system
via a simple and intuitive interface allowing users to input, compute, and interpret the results in an easy manner.

Flowchart of the System

Below is the EduGraph's flowchart, depicting a logical flow or decision-making process of an academic performance predictor and visualizer. It guides the user from feeding in the data to the analysis of grades, where a user can estimate the scores required to achieve a certain grade at the end. The system will validate, compute, and visualize data for accurate results in a user-friendly format.

EduGraph gets the user input: current grades and their weights, weights of upcoming assessments, and the target grade at the end of the semester. After the inputs are validated, it calculates the total, done, and remaining weights. Then it calculates the current weighted grade and determines whether additional performance is required to achieve the target.

If all weights are valid and remaining assessments exist, it calculates the required upcoming average and checks for the following key conditions:

Whether the target grade is already achieved - required average ≤ 0

Whether the target grade is still achievable ($0 <$ required average ≤ 100)

Whether the target is impossible to reach (required average > 100)

From here, EduGraph produces feedback messages and visual performance summaries as well as interactive What-If Curves to simulate a variety of upcoming scores and see what the effect on the final grade would be. This can be done in a continuous loop until the user stops further exploring projections.

The flowchart thus represents not only the computational logic but also the system's decision structure that ensures user guidance, feedback, and adaptability.

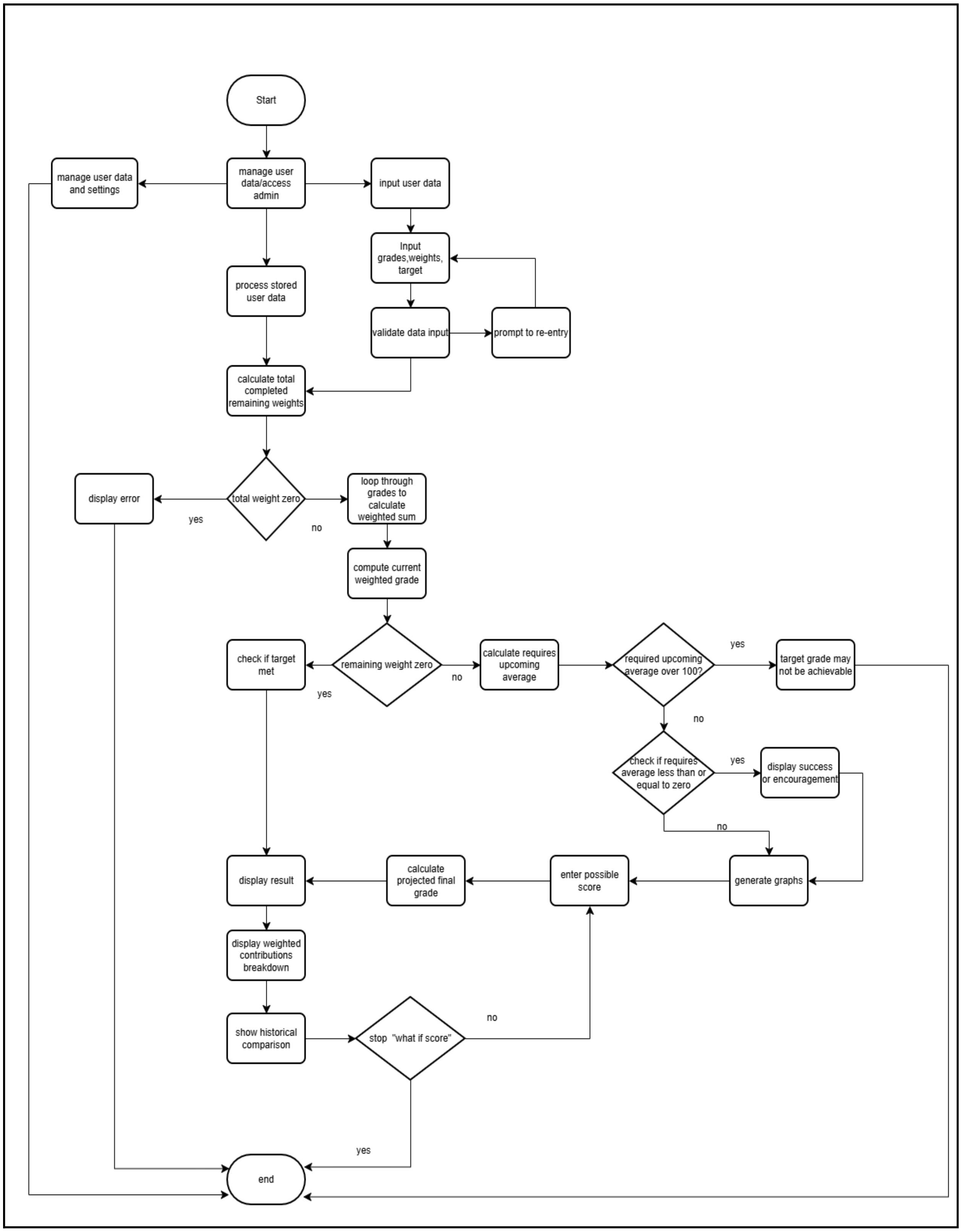
Start
manage user data/access admin
manage user data and settings
input user data
Input grades,weights, target
process stored user data
validate data input
prompt to re-entry
calculate total completed remaining weights
total weight zero
yes
no
display error
loop through grades to calculate weighted sum
compute current weighted grade
remaining weight zero
yes
no
check if target met
calculate requires upcoming average
required upcoming average over 100?
yes
no
target grade may not be achievable
check if requires average less than or equal to zero
yes
no
display success or encouragement
generate graphs
enter possible score
calculate projected final grade
display result
display weighted contributions breakdown
show historical comparison
stop "what if score"
no
yes
end

Figure 1: flowchart of the EduGraph program

Figure 1 illustrates the general logical process of EduGraph, an intelligent grade-monitoring and prediction system that will allow users to manage their profiles, calculate weighted grades, predict required upcoming performance, and visualize results using graphs and “what-if” simulations.

Pseudocode

The pseudocode provides the detailed logical presentation of EduGraph's operations. It highlights the flow of operations involved in computing, analyzing, and visualizing student performance data. This structured algorithm will simulate how a user of EduGraph will input grades, assessment weights, and the target final grade to get the current academic standing and predict the performance required in upcoming assessments.

The pseudocode also incorporates conditional statements and iterative structures for handling multiple situations, such as a required average that exceeds achievable limits or when no remaining assessments are available. Furthermore, the incorporation of a “What-If” loop enables the user to explore different possible scores and visually see through the use of a graph how each affects the final grade. This logical design ensures the system works efficiently, takes accurate input from the users, and provides useful feedback in addition to visual insights that can be used for academic decision-making.

```
START

DISPLAY "Welcome to EduGraph"

INPUT current_grades[]          // e.g., [85, 90, 78]
INPUT current_weights[]         // e.g., [30, 20, 10]
INPUT upcoming_weights[]        // e.g., [40]
INPUT target_final_grade        // e.g., 90

INITIALIZE total_weight = SUM(current_weights) + SUM(upcoming_weights)
INITIALIZE completed_weight = SUM(current_weights)
INITIALIZE remaining_weight = SUM(upcoming_weights)

IF total_weight = 0 THEN
    DISPLAY "Error: Total weight cannot be zero."
    EXIT


INITIALIZE weighted_sum = 0
FOR i FROM 0 TO LENGTH(current_grades) - 1 DO
    weighted_sum += current_grades[i] * current_weights[i] / total_weight
    DISPLAY "After grade ", i+1, ": Partial Weighted Contribution = ", weighted_sum


CALCULATE current_weighted_grade = (weighted_sum * total_weight / completed_weight)
DISPLAY "Current Weighted Grade (so far): ", current_weighted_grade
DISPLAY "Remaining Weight: ", remaining_weight

IF remaining_weight = 0 THEN
    DISPLAY "No remaining assessments."
    IF current_weighted_grade >= target_final_grade THEN
        DISPLAY "Congratulations! You've already reached your target."
    ELSE
        DISPLAY "Final target not met. No remaining weight to improve."
DISPLAY "Generating performance graph..."

// PLOT 1: Current Grades vs Assessment Number
PLOT X = [1, 2, ..., LENGTH(current_grades)]
     Y = current_grades[]
     LABEL "Completed Assessments"

// PLOT 2: Required Grade Point for Remaining Assessment(s)
ADD POINT AT X = LENGTH(current_grades) + 1
     Y = required_upcoming_average
     LABEL "Required Upcoming Grade"

// “What-if” curve
// Show how final grade changes for different upcoming scores
FOR possible_score FROM 0 TO 100 STEP 5 DO
    CALCULATE projected_final = (weighted_sum * 100 / total_weight)
                              + (possible_score * remaining_weight / total_weight)
    STORE (possible_score, projected_final) IN curve_points[]


PLOT curve_points AS LINE
LABEL "Final Grade vs Upcoming Score"

DISPLAY Graph shows your past grades, required score, and final grade curve.

END
```

Figure 2: Pseudo code of <system>

Figure 2 presents EduGraph pseudocode describes the logical flow for an academic grade prediction and visualization system that will help students track their current academic standing and predict the scores required to reach a target final grade.

Data Dictionary

The data dictionary provides a complete reference for all variables used within the EduGraph system. It serves as a guide to understand how data is stored, processed, and used throughout the grade prediction workflow. To specify each data element's name, data type, size or example value, and its purpose, the data dictionary helps ensure uniformity and clarity during system development, testing, and maintenance. It will also help developers and users understand how different components interact in achieving accurate computations of grades and performance visualizations.

The following table outlines the list of variables in EduGraph. It includes arrays for current_grades[], current_weights[], and upcoming_weights[] for storing user input grades and weight values. Other variables, such as weighted_sum, total_weight, and current_weighted_grade, have been used for essential computational functions to derive the learner's current academic standing. Variables required_upcoming_average and projected_final support the predictive function of the system to calculate the scores needed to reach a target final grade and plot the "What-If" scenario curve. Arrays X[], Y[], and curve_points[] provide the required data for plotting graphical visualizations, whereas system_message provides real-time feedback and alerts to guide the user. Together, these data elements allow EduGraph to function as a dynamic grade forecasting and visualization tool.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
current_grades[]	e.g.[85, 90,78]	Array of float	Stores the grades of completed assessments
current_weights[]	E.g. [30, 20, 10]	Array of float	Stores the corresponding weight (in %) for each completed assessment.
upcoming_weights[]	e.g., [40]	Array of Float	Stores the weights of upcoming (future) assessments.
target_final_grade	e.g., 90.0	Float	Represents the desired final grade the student aims to achieve
total_weight	100.0	Float	Sum of all completed and upcoming assessment weights
completed_weight	e.g., 60.0	Float	Sum of all weights from completed assessments
remaining_weight	e.g., 40.0	Float	Sum of all weights from remaining assessments.
weighted_sum	varies	Float	Running total of each grade’s contribution to the final grade
current_weighted_grade	e.g., 86.5	Float	Computed grade based on completed assessments.
required_upcoming_avera	e.g., 92.3	Float	Grade needed in

ge			upcoming assessments to reach the target final grade.
Possible_score	0–100	Float	Test variable used for “what-if” analysis loop.
projected_final	varies	Float	Calculated final grade for each possible future score
curve_points[]	e.g., [(80, 88.5), (90, 91.2)]	Array of Tuple	Stores data points (x = possible score, y = projected final grade) for graph plotting.
x[]	1, 2, 3, ...]	Array of Integer	X-axis values representing assessment numbers.
y[]	85, 90, 78]	Array of Float	Y-axis values representing assessment grades
system_message	varies	String	Temporary text used to display feedback messages (e.g., alerts, errors, congratulations)

Code
<Subsection introduction >

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <string>
using namespace std;

// Cpe11S5
// Cristobal, Jhune Anderson
// Condesa, Andrie Galicia
// Desalit, Hanz Lorenz D.
// Olmoguez, Zydric Angelo
// Lacsamana, Timothy Justin P.

//
```

```
────
// Function: Draw a vertical bar chart with axes and labels
//
──────────────────────────────────────────────────────────────────────
────
void drawBarGraph(const vector<double>& values, const vector<string>& labels, int maxHeight = 10) {
    double maxVal = *max_element(values.begin(), values.end());
    if (maxVal < 1) maxVal = 1; // Avoid division by zero

    vector<int> barHeights;
    for (double val : values) {
        barHeights.push_back(int((val / maxVal) * maxHeight));
    }

    for (int h = maxHeight; h > 0; h--) {
        cout << setw(4) << fixed << setprecision(0) << int(maxVal * h / maxHeight) << " |";
        for (int i = 0; i < values.size(); i++) {
            cout << (barHeights[i] >= h ? " █ " : "   ");
        }
        cout << "\n";
    }

    cout << "    +";
    for (int i = 0; i < values.size(); i++) cout << "---";
    cout << "\n     ";
    for (const auto& label : labels) cout << setw(3) << label;
    cout << "\n";
}

//
──────────────────────────────────────────────────────────────────────
────
// Function: Admin login
//
──────────────────────────────────────────────────────────────────────
────
bool adminLogin() {
    string password;
    cout << "\n--- Admin Login ---\n";
    cout << "Enter admin password: ";
    cin >> password;

    if (password == "admin123") {
        cout << "Access granted. Admin mode ON.\n";
        return true;
    } else {
        cout << "Access denied. Continuing as normal user.\n";
        return false;
    }
}
```

```
//
────────────────────────────────────────────────────────────────────────────
────
// Main Program
//
────────────────────────────────────────────────────────────────────────────
────
int main() {
    cout << "╔══════════════════════════════════════╗\n";
    cout << "║      Welcome to EduGraph      ║\n";
    cout << "╚══════════════════════════════════════╝\n";

    // Admin login prompt
    bool isAdmin = false;
    int adminChoice;
    cout << "Login as admin? (1 = yes, 0 = no): ";
    cin >> adminChoice;
    if (adminChoice == 1) isAdmin = adminLogin();

    // Input completed grades and weights
    int n;
    cout << "\nEnter number of completed grades: ";
    cin >> n;

    vector<double> current_grades(n), current_weights(n);
    cout << "Enter completed grades (0-100): ";
    for (int i = 0; i < n; i++) cin >> current_grades[i];

    cout << "Enter corresponding weights: ";
    for (int i = 0; i < n; i++) cin >> current_weights[i];

    // Admin editing feature
    if (isAdmin) {
        cout << "\n--- Admin Feature: Edit Grades/Weights ---\n";
        int editChoice;

        cout << "Edit grades? (1 = yes, 0 = no): ";
        cin >> editChoice;
        if (editChoice == 1) {
            for (int i = 0; i < n; i++) {
                cout << "Grade #" << (i + 1) << ": " << current_grades[i] << " → ";
                cin >> current_grades[i];
            }
        }

        cout << "Edit weights? (1 = yes, 0 = no): ";
        cin >> editChoice;
        if (editChoice == 1) {
            for (int i = 0; i < n; i++) {
```

```
            cout << "Weight #" << (i + 1) << ": " << current_weights[i] << " → ";
            cin >> current_weights[i];
        }
    }
}

// Input upcoming assessments
int m;
cout << "\nEnter number of upcoming assessments: ";
cin >> m;

vector<double> upcoming_weights(m);
cout << "Enter upcoming weights: ";
for (int i = 0; i < m; i++) cin >> upcoming_weights[i];

double target_final_grade;
cout << "Enter target final grade: ";
cin >> target_final_grade;

// Calculate weights and grades
double completed_weight = 0, remaining_weight = 0;
for (double w : current_weights) completed_weight += w;
for (double w : upcoming_weights) remaining_weight += w;
double total_weight = completed_weight + remaining_weight;

if (total_weight == 0) {
    cout << "Error: Total weight cannot be zero.\n";
    return 1;
}

double weighted_sum = 0;
for (int i = 0; i < n; i++) {
    weighted_sum += current_grades[i] * current_weights[i] / total_weight;
}

double current_weighted_grade = weighted_sum * total_weight / completed_weight;

cout << fixed << setprecision(2);
cout << "\nCurrent Weighted Grade: " << current_weighted_grade << "\n";
cout << "Remaining Weight: " << remaining_weight << "\n";

double required_upcoming_average = 0;
if (remaining_weight > 0) {
    double required_total = target_final_grade * total_weight - completed_weight * current_weighted_grade;
    required_upcoming_average = required_total / remaining_weight;
    cout << "Required Average on Upcoming Assessments: " << required_upcoming_average << "\n";
}

// Display bar graph
vector<double> barValues = current_grades;
```

```
    vector<string> labels;
    for (int i = 0; i < n; i++) labels.push_back(to_string(i + 1));
    if (remaining_weight > 0) {
        barValues.push_back(required_upcoming_average);
        labels.push_back("Req");
    }

    cout << "\nGrade Bar Graph:\n";
    drawBarGraph(barValues, labels, 10);

    return 0;
}
```

Figure 3. <name of figure>

Code discussion

Results and Discussion

EduGraph is a C++ console-based program that will calculate a student's current weighted grades, find out what average they need on future assessments to reach a desired final grade, and graphically display it through the use of a vertical bar graph. It contains an Admin Mode in which, after entering a password, the user can modify already-earned grades and weights to correct calculation discrepancies before performing analysis. Input validation prevents the entry of invalid values, while the program is designed to explicitly show, in visual and numeric format, grade information to the user.

The program can provide weighted contribution and weighted average when a user inputs completed grades and weights, as well as any upcoming assessment weights and a target final grade. For completed grades of 85 with a 30% weight and 90 with a 20% weight, for example, with the remaining assessment weight of 50% and a target final grade of 90, the program will show the completed grades as a bar graph and the required average needed for the upcoming assessments. Summary: Completed weight sum is 50%, remaining weight is 50%, and total weight is 100%; weighted contribution of completed assessments is 46, and current weighted average is 92, thus making the average required for the upcoming assessments 44 to achieve the set target. This would mean that achieving the target should not be a problem, and in this case, visually, it would be easy to see from the bar graph the relationship between the completed performance and the requirement for the rest. Admin mode allows grades or weights to be adjusted before calculating them, which provides for the accuracy of the projections and flexibility in scenario analysis.

Conclusion

EduGraph helps students and instructors understand standings and requirements for future assessments through an effective combination of visual and numerical analysis. The intuitiveness of a bar graph, along with summary and target analyses, enables precise numeric insights. Admin Mode enhances control and accuracy by allowing grade edits before calculations. It is recommended that future versions include “what-if” scenario projections for multiple upcoming scores, CSV export for grade data, or a GUI-based interface for enhanced usability and interactive analysis. These would let EduGraph become a more complete performance monitoring and planning utility.

References

<APA Format Alphabetical Order>

1. Alboaneen, D., Alzahrani, A., & Baghabra, A. (2022). Development of a web-based prediction system for students’ academic performance using machine learning algorithms. Data, 7(2), 21. https://doi.org/10.3390/data7020021

2. Albreiki, B., Zaki, N., & Alashwal, H. (2021). A systematic literature review of student’ performance prediction using machine learning techniques. Education Sciences, 11(9), 552. https://doi.org/10.3390/educsci11090552

3. Chen, S., & Cheng, M. (2023). A machine learning approach to predicting academic performance: The impact of socioeconomic status and educational data. Social Sciences, 12(3), 118. https://doi.org/10.3390/socsci12030118

4. Liu, Y. (2022). Predicting student performance using clickstream data. Education Sciences, 13(1), 17. https://doi.org/10.3390/educsci13010017

5. Namoun, A., & Alshanqiti, A. (2020). Predicting student performance using data mining and learning analytics techniques: A systematic literature review. Applied Sciences, 11(1), 237. https://doi.org/10.3390/app11010237