GradeBook Analyzer CLI - Assignment 2

Course: Programming for Problem Solving Using Python (ETCCPP102)

Program: B.Tech CSE (AI & ML)

Department: SOET

Semester: I

Session: 2025-26

Submitted By: Neha

Roll Number: 2501730466

Submitted To: Ms. Neha Kaushik

University: K.R. Mangalam University

Assignment Description

This project automates the evaluation of student performance using Python.
It allows for both manual entry and CSV import of student marks, computes statistical values such as
mean, median, minimum, and maximum, assigns letter grades, and displays results in a formatted table.
The program can also export results to a CSV file.

Learning Objectives

- Read or import marks.
- Compute mean, median, min, and max.
- Assign grades automatically.
- Filter pass/fail using list comprehension.
- Export results to CSV.
- Practice modular programming.

Python Code

```
"""
GradeBook Analyzer CLI
Assignment: Programming for Problem Solving Using Python (ETCCPP102)
Author: Neha Moyal
Roll No: 2501730466
Date: 2025-11-12
```

```
Instructor: Ms. Neha Kaushik
University: K.R. Mangalam University

This script implements:
- Manual input or CSV import of student marks
- Statistical functions: average, median, max, min
- Grade assignment (A, B, C, D, F)
- Pass/fail lists using list comprehensions
- Formatted results table
- Menu loop for repeated analysis
- Optional CSV export of final grade table
"""

import csv
import sys
from typing import Dict, Tuple, List

def header():
    print("="*60)
    print("GradeBook Analyzer CLI".center(60))
    print("Programming for Problem Solving Using Python - ETCCPP102".center(60))
    print("="*60)

def calculate_average(marks: Dict[str, float]) -> float:
    if not marks:
        return 0.0
    return sum(marks.values()) / len(marks)

def calculate_median(marks: Dict[str, float]) -> float:
    scores = sorted(marks.values())
    n = len(scores)
    if n == 0:
        return 0.0
    mid = n // 2
    if n % 2 == 1:
        return scores[mid]
    else:
        return (scores[mid - 1] + scores[mid]) / 2

def find_max_score(marks: Dict[str, float]) -> float:
    return max(marks.values()) if marks else 0.0

def find_min_score(marks: Dict[str, float]) -> float:
    return min(marks.values()) if marks else 0.0

def assign_grades(marks: Dict[str, float]) -> Dict[str, str]:
    grades = {}
    for name, score in marks.items():
        if score >= 90:
            grade = 'A'
        elif score >= 80:
            grade = 'B'
        elif score >= 70:
            grade = 'C'
        elif score >= 60:
            grade = 'D'
        else:
            grade = 'F'
        grades[name] = grade
    return grades

def grade_distribution_count(grades: Dict[str, str]) -> Dict[str, int]:
    dist = {}
    for g in grades.values():
        dist[g] = dist.get(g, 0) + 1
    return dist

def pass_fail_lists(marks: Dict[str, float]) -> Tuple[List[str], List[str]]:
    passed_students = [name for name, score in marks.items() if score >= 40]
    failed_students = [name for name, score in marks.items() if score < 40]
    return passed_students, failed_students

def display_results(marks: Dict[str, float], grades: Dict[str, str]):
    print("\nFinal Result Table")
    print("-"*40)
    print(f"{'Name':<20}{'Marks':<8}{'Grade':<6}")
    print("-"*40)
    for name, score in marks.items():
        print(f"{name:<20}{score:<8.2f}{grades.get(name,''):<6}")
    print("-"*40)

def read_csv_marks(filename: str) -> Dict[str, float]:
    marks = {}
    try:
        with open(filename, 'r', newline='') as f:
            reader = csv.reader(f)
            header = next(reader, None)  # skip header if exists
            for row in reader:
                if not row:
                    continue
                # Expect: name, score
                name = row[0].strip()
                try:
                    score = float(row[1])
                except (IndexError, ValueError):
                    print(f"Skipping invalid row: {row}")
                    continue
                marks[name] = score
    except FileNotFoundError:
        print(f"File '{filename}' not found.")
    except Exception as e:
        print("Error reading CSV:", e)
    return marks

def export_grades_to_csv(filename: str, marks: Dict[str, float], grades: Dict[str, str]):
    try:
        with open(filename, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['Name', 'Marks', 'Grade'])
            for name, score in marks.items():
                writer.writerow([name, f"{score:.2f}", grades.get(name, '')])
        print(f"Grades exported to '{filename}'")
    except Exception as e:
        print("Failed to export CSV:", e)

def get_manual_marks() -> Dict[str, float]:
    marks = {}
    while True:
        try:
            n = int(input("Enter number of students: ").strip())
            if n <= 0:
                print("Please enter a positive integer.")
                continue
            break
        except ValueError:
            print("Enter a valid integer.")
    for i in range(1, n+1):
        name = input(f"Student {i} Name: ").strip()
        while True:
            try:
                score = float(input(f"Student {i} Marks (0-100): ").strip())
                if score < 0 or score > 100:
                    print("Please enter marks between 0 and 100.")
                    continue
                break
            except ValueError:
                print("Enter a valid numeric mark.")
        marks[name] = score
    return marks

def summary_and_metrics(marks: Dict[str, float], grades: Dict[str, str]):
    avg = calculate_average(marks)
    med = calculate_median(marks)
    max_s = find_max_score(marks)
    min_s = find_min_score(marks)
    passed, failed = pass_fail_lists(marks)
    dist = grade_distribution_count(grades)

    print(f"\nSummary Metrics:")
    print(f"Average: {avg:.2f}")
    print(f"Median: {med:.2f}")
    print(f"Highest Score: {max_s:.2f}")
    print(f"Lowest Score: {min_s:.2f}")
    print(f"Passed: {len(passed)}, Failed: {len(failed)}")
    print("Grade Distribution:", dist)

def main():
    header()
    while True:
        print("\nMenu:")
        print("1. Enter Marks Manually")
        print("2. Load Marks from CSV")
        print("3. Exit")
        choice = input("Enter choice (1/2/3): ").strip()

        if choice == '1':
            marks = get_manual_marks()
        elif choice == '2':
            filename = input("Enter CSV filename (e.g. marks.csv): ").strip()
            marks = read_csv_marks(filename)
            if not marks:
                print("No marks loaded. Returning to menu.")
                continue
        elif choice == '3':
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid option. Try again.")
            continue

        # compute and display
        grades = assign_grades(marks)
        display_results(marks, grades)
        summary_and_metrics(marks, grades)

        # optional export
        while True:
            save_opt = input("\nDo you want to export the result to CSV? (y/n): ").strip().lower()
            if save_opt == 'y':
                out_file = input("Enter output filename (e.g. grade_report.csv): ").strip()
                export_grades_to_csv(out_file, marks, grades)
                break
            elif save_opt == 'n':
                break
            else:
                print("Enter 'y' or 'n'.")

        # allow repeated analysis or exit/return to menu
        while True:
                cont = input("\nDo another analysis? (y to continue / m for menu / e to exit): 
").strip().lower()
            if cont == 'y':
                # continue with manual input or csv again
                break
            elif cont == 'm':
                # go back to main menu
                break
            elif cont == 'e':
                print("Exiting program. Goodbye!")
                sys.exit(0)
            else:
                print("Enter y, m, or e.")
        if cont == 'm':
            continue
        # if cont == 'y', loop continues to ask menu again
```

```
if __name__ == "__main__":
    main()
```

Sample Output

```
Name        Marks   Grade
------------------------
Alice       78.00   C
Bob         92.00   A
Charlie     35.00   F
------------------------
Average: 68.33, Median: 78.00
Highest: 92, Lowest: 35
Passed: 2, Failed: 1
```