Assignment – 2

GradeBook Analyzer

Name – Anant Kumar

Roll no. – 2501730189

Course – B-Tech CSE AI/ML

Section - A

Summary

The GradeBook Analyzer is an educational mini-project designed to automate the process of analyzing and reporting student grades. This project addresses a real-world problem faced by academic instructors: the need for efficient, accurate statistical analysis and grade computation without manual calculations. By implementing a command-line interface (CLI) application, students gain practical experience in Python programming fundamentals while solving an authentic educational use case.

Introduction

Project Context

In academic institutions, lecturers regularly encounter the time-consuming task of computing statistics on student marks, assigning letter grades, and generating summary reports. This project transforms that manual process into an automated system that reads student data from multiple sources, performs statistical analysis, assigns grades according to predefined criteria, and generates user-friendly reports.

Learning Objectives

The project is structured to achieve several key learning outcomes:

- **Data Structure Mastery**: Students learn to read and store student marks from manual input or CSV files into appropriate Python data structures (dictionaries, lists, and combinations thereof)
- **Statistical Analysis**: Implementation of core mathematical functions to calculate mean, median, minimum, and maximum values from datasets
- **Control Flow Logic**: Using conditional statements to assign letter grades based on numeric score ranges
- **Functional Programming**: Creating modular, reusable functions that follow the single responsibility principle
- **Data Processing**: Applying list comprehensions for conditional filtering and data manipulation
- **User Interface Design**: Implementing an interactive CLI with menu-driven navigation and formatted output
- **File I/O Operations**: Reading from and writing to CSV files for data persistence and reporting

Project Structure and Components

Task 1: Project Initialization and Setup

The project begins with establishing a proper development foundation. This includes creating an organized project directory structure with a main script file. The initialization phase requires including proper documentation in the script header, which serves as metadata identifying the author, date, and project title. Additionally, a welcome message and usage menu are displayed to guide users through available options.

This foundational setup teaches students about code organization and the importance of clear user communication from the start of any application.

Task 2: Data Input and Import Mechanisms

The system provides two pathways for getting student data into the application. The first mechanism allows manual entry, where users input student names and corresponding marks one by one. This approach is useful for small datasets and teaches input validation principles. The second mechanism reads from CSV (Comma-Separated Values) files, enabling bulk data import for larger classes.

Both input methods store the data in an optimized structure—typically a dictionary where student names serve as keys and marks as values. This data structure choice reflects real-world considerations about lookup efficiency and data organization.

Task 3: Statistical Analysis Functions

Once data is loaded, the system performs statistical analysis through dedicated functions. These functions calculate:

- **Average (Mean)**: The sum of all marks divided by the number of students, providing a measure of central tendency
- **Median**: The middle value in a sorted list of marks, useful for understanding the distribution's center while being resistant to outliers
- **Maximum Score**: The highest mark achieved, indicating peak performance in the class
- **Minimum Score**: The lowest mark achieved, indicating areas needing support

These statistics are computed using either built-in Python functions or custom implementations, helping students understand both practical programming and mathematical concepts.

Task 4: Grade Assignment and Distribution Analysis

Grade assignment implements a tiered classification system that maps numeric scores to letter grades:

- **Grade A**: Scores of 90 or above, representing excellence
- **Grade B**: Scores from 80 to 89, representing above-average performance
- **Grade C**: Scores from 70 to 79, representing average performance
- **Grade D**: Scores from 60 to 69, representing below-average but passing performance
- **Grade F**: Scores below 60, representing failure

After assigning individual grades, the system calculates the grade distribution—counting how many students received each letter grade. This provides a quick overview of class performance and helps identify whether grading distributions are reasonable.

Task 5: Pass/Fail Analysis Using List Comprehensions

This component introduces functional programming concepts through list comprehensions. The system separates students into two categories:

- **Passed Students**: Those with scores of 40 or above
- **Failed Students**: Those with scores below 40

List comprehensions provide a concise, Pythonic way to filter data based on conditions. This component demonstrates how to transform one dataset into subsets based on specific criteria, a fundamental data processing technique.

Task 6: Results Presentation and User Interaction

The final component presents all analysis results in a formatted table format, displaying each student's name, marks, and assigned grade in aligned columns. This requires string formatting techniques to ensure readable output.

Additionally, the system implements a user loop that allows repeated analyses—users can load different datasets or exit the application at will, with a user-friendly menu guiding each interaction.

An optional enhancement allows exporting final results to a CSV file for sharing with stakeholders or archiving.

Key Programming Concepts

Data Structures

The project heavily utilizes dictionaries for mapping student names to their marks and grades. This data structure choice offers several advantages: O(1) lookup time, natural representation of key-value relationships, and straightforward iteration patterns.

Functional Decomposition

Rather than writing monolithic code, the project encourages breaking functionality into discrete functions. Each function has a single, clear purpose—calculating average, assigning grades, filtering students, and so on. This modular approach makes code more testable, reusable, and maintainable.

String Formatting

The project requires generating human-readable output, necessitating techniques like f-strings, escape sequences (newlines, tabs), and field width specifications for aligned table presentation.

File Handling

Both reading from and writing to CSV files introduces students to Python's file I/O capabilities and the CSV module for parsing and generating structured data.

Implementation Approach

Data Flow

The application follows a clear data flow pattern:

1. **Input**: Collect student data either manually or from a file
2. **Processing**: Calculate statistics, assign grades, determine pass/fail status
3. **Analysis**: Generate distribution reports and summaries
4. **Output**: Display results in formatted tables and optionally export to CSV
5. **Iteration**: Allow users to repeat the process or exit

Design Principles

The implementation emphasizes:

- **Modularity**: Each function performs one task well
- **Reusability**: Functions can be called with different data
- **Maintainability**: Clear naming and structure make code easy to understand
- **User-Centricity**: Menu-driven interfaces and formatted output prioritize user experience

Conclusion

The GradeBook Analyzer project provides a comprehensive introduction to practical Python programming while solving a real-world educational problem. Through implementing this system, students develop competency across multiple areas: data structure selection, algorithm implementation, file I/O operations, user interface design, and modular programming practices. The project balances conceptual learning with practical application, preparing students for more complex software development challenges while immediately demonstrating the utility of programming skills in academic and professional contexts.

Code

```
import csv
import statistics


# Task 1: Project Setup & Menu #


def display_menu():
    print(" GradeBook Analyzer ")
    print("1. Enter student data manually")
    print("2. Load student data from CSV file")
    print("3. Exit")


# Task 2: Data Entry or CSV Import #


def get_manual_data():
    marks = {}
    n = int(input("Enter number of students: "))
    for i in range(n):
        name = input(f"Enter name of student {i+1}: ")
        score = float(input(f"Enter marks for {name}: "))
        marks[name] = score
    return marks
```

```
def get_csv_data():
    marks = {}
    filename = input("Enter CSV filename (with .csv): ")
    try:
        with open(filename, newline='') as csvfile:
            reader = csv.reader(csvfile)
            next(reader)  # skip header
            for row in reader:
                if len(row) >= 2:
                    name, score = row[0], float(row[1])
                    marks[name] = score
        print(f"Loaded data from {filename}")
    except FileNotFoundError:
        print("File not found. Try again.")
    return marks

# Task 3: Statistical Functions #

def calculate_average(marks_dict):
    return sum(marks_dict.values()) / len(marks_dict)

def calculate_median(marks_dict):
    return statistics.median(marks_dict.values())

def find_max_score(marks_dict):
    return max(marks_dict.items(), key=lambda x: x[1])

def find_min_score(marks_dict):
    return min(marks_dict.items(), key=lambda x: x[1])
```

```
# Task 4: Grade Assignment #

def assign_grades(marks_dict):
    grades = {}
    for name, score in marks_dict.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(grades_dict):
    distribution = {'A':0, 'B':0, 'C':0, 'D':0, 'F':0}
    for grade in grades_dict.values():
        distribution[grade] += 1
    return distribution

# Task 5: Pass/Fail Filter #

def pass_fail_lists(marks_dict):
    passed = [name for name, score in marks_dict.items() if score >= 40]
    failed = [name for name, score in marks_dict.items() if score < 40]
    return passed, failed
```

```
# Task 6: Results Table & Loop #

def display_results(marks_dict, grades_dict):
    print("\nName\t\tMarks\tGrade")
    for name, score in marks_dict.items():
        print(f"{name:<15}{score:<10}{grades_dict[name]}")

def export_to_csv(marks_dict, grades_dict):
    choice = input("Do you want to export results to CSV? (y/n): ").lower()
    if choice == 'y':
        with open("final_gradebook.csv", "w", newline='') as file:
            writer = csv.writer(file)
            writer.writerow(["Name", "Marks", "Grade"])
            for name, marks in marks_dict.items():
                writer.writerow([name, marks, grades_dict[name]])
        print("Results saved as final_gradebook.csv")

# Main CLI Loop #

def main():
    print("Welcome to the GradeBook Analyzer!")
    while True:
        display_menu()
        choice = input("Enter your choice: ")

        if choice == '1':
            marks = get_manual_data()
        elif choice == '2':
            marks = get_csv_data()
        elif choice == '3':
            print("Exiting GradeBook Analyzer. Goodbye!")
```

```
        break
    else:
        print("Invalid option. Try again.")
        continue

    if marks:
        avg = calculate_average(marks)
        med = calculate_median(marks)
        max_score = find_max_score(marks)
        min_score = find_min_score(marks)

        print("Statistical Summary")
        print(f"Average Marks: {avg:.2f}")
        print(f"Median Marks: {med:.2f}")
        print(f"Highest Score: {max_score[0]} ({max_score[1]})")
        print(f"Lowest Score: {min_score[0]} ({min_score[1]})")

        grades = assign_grades(marks)
        dist = grade_distribution(grades)

        print("Grade Distribution")
        for grade, count in dist.items():
            print(f"{grade}: {count}")

        passed, failed = pass_fail_lists(marks)
        print("Pass/Fail Summary")
        print(f"Passed ({len(passed)}): {', '.join(passed) if passed else 'None'}")
        print(f"Failed ({len(failed)}): {', '.join(failed) if failed else 'None'}")

        display_results(marks, grades)
        export_to_csv(marks, grades)
```

```
        input("Press Enter to continue...")


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