STUDENT GRADE ANALYZER

VITYARTHI PROJECT

By Aditya Singh

25BCE11212

PROJECT REPORT – Student Grade Analyzer

1. Cover Page

Project Title: Student Grade Analyzer
Submitted By: Aditya Singh
Roll No: 25BCE11212
Course: Introduction to problem solving

2. Introduction

Evaluating student performance is essential for understanding learning outcomes. Traditionally, teachers analyze scores manually, which is slow, inefficient, and prone to error—especially when dealing with large student batches.

The **Student Grade Analyzer** project offers a simple and efficient Python-based tool that automatically processes student marks, calculates statistical summaries, evaluates performance, and generates a structured report.

The project demonstrates Python programming concepts such as modular coding, file handling, functions, conditional logic, error handling, logging, and JSON report generation.

3. Problem Statement

In many educational settings, instructors receive marks in spreadsheets or handwritten formats and must compute statistics manually. This leads to:

- Inconsistencies
- Calculation mistakes
- Delays in generating performance insights
- Difficulty identifying weak students
- No quick way to understand class distribution

There is a need for a **Python-based automated system** that:

1. Loads marks from CSV or manual input
2. Computes complete statistical measures
3. Evaluates student performance
4. Generates a detailed report
5. Logs actions and errors

The Student Grade Analyzer solves these problems efficiently.

---

4. Functional Requirements

1 – Input Handling Module

- Load marks from a CSV file
- Allow manual input via CLI

- Validate numeric entries

2 – Statistics Calculation Module

- Compute:
    - Mean
    - Median
    - Mode
    - Variance
    - Standard Deviation
    - Highest & Lowest marks

3 – Performance Analysis Module

- Total students
- Pass/fail count
- Pass percentage
- Top 3 performers
- Weakest 3 performers

4 – Report Generation Module

- Save analysis to `report.json`
- Include both stats + performance

5 – Logging Module

- Log warnings, invalid values, and process steps

5. Non-Functional Requirements

1 – Usability

Simple command-line interface with clear options.

2 – Reliability

Handles invalid inputs and missing files gracefully.

3 – Maintainability

Modular structure ensures easy updates and readability.

4 – Performance

Efficient handling of both small and large CSV files.

5 – Accuracy

Uses Python’s reliable `statistics` library for calculations.

6 – Logging

Uses a centralized logging system to maintain logs in `grade_analyzer.log`.

6. System Architecture

The system follows a **modular architecture**, where each task is separated into its own Python file for clarity and maintainability.

```
User
 ↓
main.py
  ├── input_handler.py (CSV/manual input)
  ├── stats_calculator.py (statistics computation)
  ├── performance_analyzer.py (performance evaluation)
  ├── report_generator.py (JSON output)
  └── utils/logger.py (logging)
```

Data Flow:

1. User inputs marks
2. Data is processed and analyzed
3. Final results are saved in JSON
4. Logs are written in log file

---

7. Design Diagrams

A. Use Case Diagram

Actor: User
Use Cases:

- Load marks
- Enter marks manually
- Analyze statistics
- Analyze performance
- Generate report

B. Workflow Diagram

1. Start
2. Choose input method

3. Validate and load marks
4. Calculate statistics
5. Evaluate performance
6. Generate JSON report
7. Display summary
8. End

C. Sequence Diagram

User → Main → InputHandler → StatsCalculator → PerformanceAnalyzer → ReportGenerator → Main → User

D. Class Diagram

Classes:

- `FileLoader` (input_handler)
- `StatsCalculator`
- `PerformanceAnalyzer`
- `ReportGenerator`
- `Logger`

Relationships:
Main uses all modules; ReportGenerator depends on Stats & Performance data.

---

8. Design Decisions & Rationale

1. Modular Architecture

Makes it easier to debug, extend, and test.

2. CSV Input Support

Most teachers use Excel/CSV files; easy integration.

3. Statistics Library

Reliable built-in Python statistics ensures accuracy.

4. JSON Reporting

Lightweight, portable, and easy to view.

5. Logging Implementation

Important for debugging and tracking program flow.

6. CLI Instead of GUI

Keeps the project simple, fast to develop, and easy to test.

---

9. Implementation Details

Technologies

- Python 3
- Standard libraries: `statistics`, `csv`, `json`, `logging`

Code Structure

- `main.py` → orchestrates the app
- `input_handler.py` → handles data input

- `stats_calculator.py` → calculates all statistics
- `performance_analyzer.py` → computes class insights
- `report_generator.py` → writes JSON
- `logger.py` → logging

Input Format

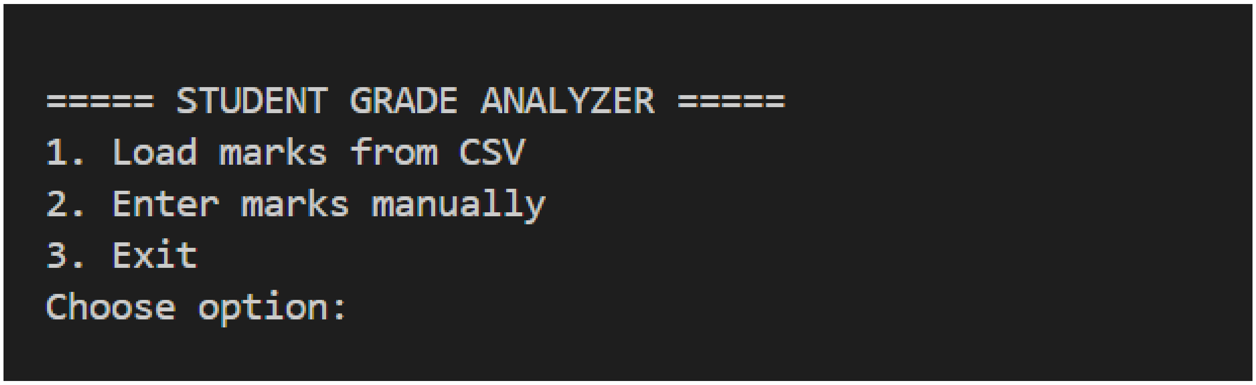
CSV expected format:

Name,Marks
Rahul,78
Sneha,65
Vijay,32

---

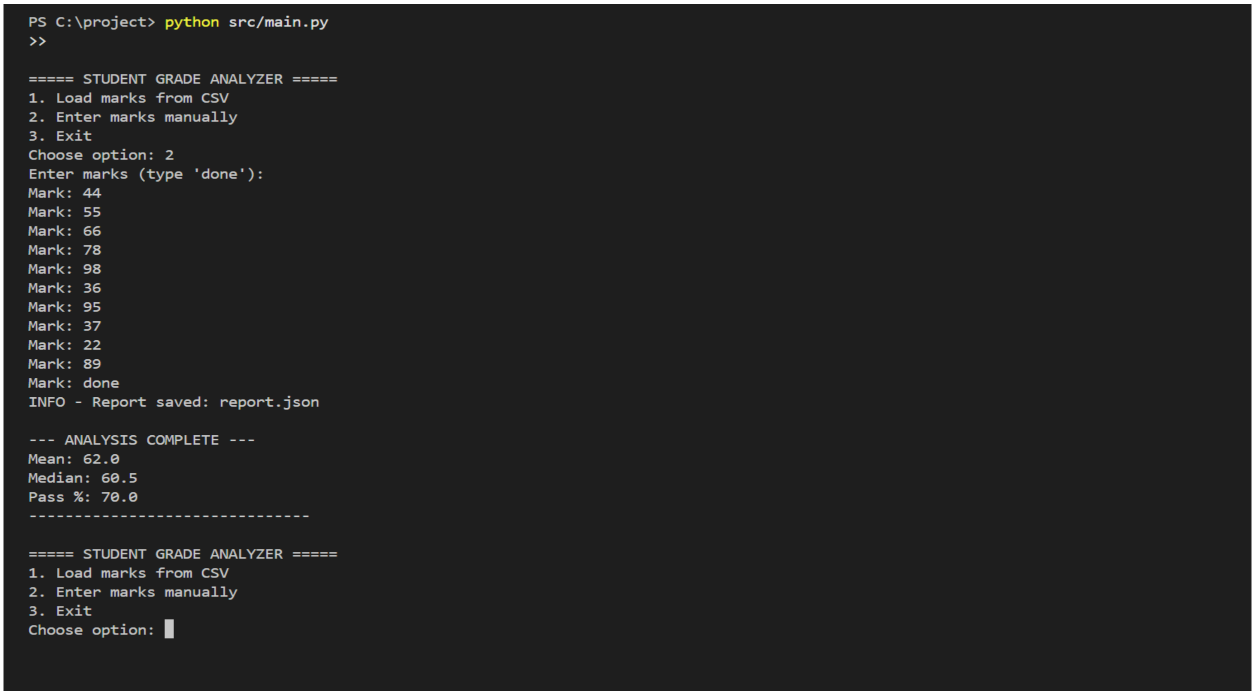
10. Screenshots

1. **Menu**



```
===== STUDENT GRADE ANALYZER =====
1. Load marks from CSV
2. Enter marks manually
3. Exit
Choose option:
```

2. JSON report



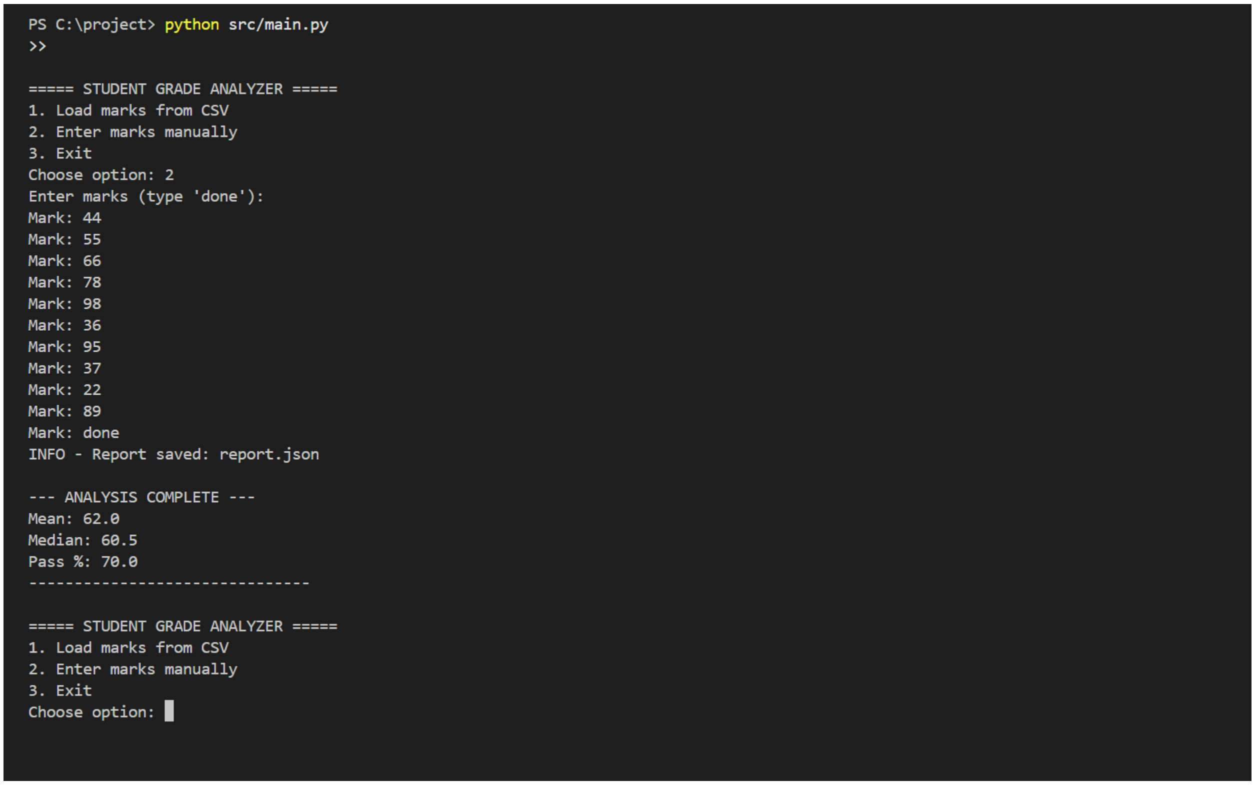
```
PS C:\project> python src/main.py
>>

===== STUDENT GRADE ANALYZER =====
1. Load marks from CSV
2. Enter marks manually
3. Exit
Choose option: 2
Enter marks (type 'done'):
Mark: 44
Mark: 55
Mark: 66
Mark: 78
Mark: 98
Mark: 36
Mark: 95
Mark: 37
Mark: 22
Mark: 89
Mark: done
INFO - Report saved: report.json

--- ANALYSIS COMPLETE ---
Mean: 62.0
Median: 60.5
Pass %: 70.0
------------------------------

===== STUDENT GRADE ANALYZER =====
1. Load marks from CSV
2. Enter marks manually
3. Exit
Choose option: █
```

3. Output



```
PS C:\project> python src/main.py
>>

===== STUDENT GRADE ANALYZER =====
1. Load marks from CSV
2. Enter marks manually
3. Exit
Choose option: 2
Enter marks (type 'done'):
Mark: 44
Mark: 55
Mark: 66
Mark: 78
Mark: 98
Mark: 36
Mark: 95
Mark: 37
Mark: 22
Mark: 89
Mark: done
INFO - Report saved: report.json

--- ANALYSIS COMPLETE ---
Mean: 62.0
Median: 60.5
Pass %: 70.0
------------------------------

===== STUDENT GRADE ANALYZER =====
1. Load marks from CSV
2. Enter marks manually
3. Exit
Choose option: █
```

11. Testing Approach

Manual Tests

- Test with valid CSV
- Test with invalid CSV
- Empty CSV
- Manual entries
- Invalid numbers
- Extreme marks

Validation

Checked accuracy using known datasets.

12. Challenges Faced

- Handling non-numeric CSV values
- Dealing with mode when data is multi-modal
- Ensuring JSON formatting remains clean
- Designing a simple but effective modular structure

13. Learnings & Key Takeaways

- Modular design improves clarity
- File handling and validation are core Python skills
- Logging is essential for debugging
- JSON reports are powerful and easy to parse
- Practical experience with statistics module

---

14. Future Enhancements

- Add a graphical user interface (GUI)
- Create a PDF report instead of JSON
- Add subject-wise comparison
- Add charts (matplotlib)
- Store results in database
- Build REST API

---

15. References

- Python Official Documentation
- Python `statistics` module
- StackOverflow for input validation handling