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 Number:25BCE11212
Course: introduction to problem solving

2. Introduction

Analyzing student marks is something that teachers regularly do, but it often involves going through spreadsheets manually or calculating statistics by hand. This can be slow and prone to mistakes.
To solve this, I built a simple Python-based tool that can automatically read marks, calculate useful statistics, evaluate performance, and generate a structured report.

This project shows how Python can be used for real-world tasks like data handling, modular programming, statistics, logging, and file processing.

3. Problem Statement

Manual mark analysis becomes difficult when:

- There are many students
- The teacher needs quick insights
- Calculations like mean, median, or pass percentage must be repeated

- Errors in data entry or calculations happen

There is a need for a **simple, automated, and reliable system** that:

1. Accepts marks from CSV or manual input
2. Calculates statistics accurately
3. Analyzes pass/fail performance
4. Highlights top and weak performers
5. Generates a clean summary report

The *Student Grade Analyzer* solves all the above problems using Python.

---

4. Functional Requirements

1 – Input Handling

- Load marks from a CSV file
- Accept user-entered marks via terminal
- Validate numeric inputs

2 – Statistics Calculation

- Compute:
    - Mean
    - Median
    - Mode
    - Variance

    - Standard deviation
    - Highest & lowest marks

3 – Performance Analysis

- Total students
- Passed and failed count
- Pass percentage
- Top 3 scorers
- Weakest 3 scorers

4 – Report Generator

- Save results in `report.json`
- Include both stats and performance

5 – Logging

- Save errors, warnings, and activity in `grade_analyzer.log`

---

5. Non-Functional Requirements

1 – Usability

The interface is simple and only needs basic terminal input.

2 – Reliability

The program handles invalid CSV rows and incorrect inputs properly.

3 – Maintainability

The codebase is modular, so each function is in its own file.

4 – Performance

The program can quickly process both small and large sets of marks.

5 – Accuracy

Python's `statistics` module ensures correct calculations.

6 – Logging

All events are logged to help track issues.

---

6. System Architecture

The system uses a **modular design** where each major task is handled by a separate Python file.

```
main.py  →  controls the program flow
│
├── input_handler.py        → handles input (CSV/manual)
├── stats_calculator.py     → computes statistics
├── performance_analyzer.py  → evaluates pass/fail & ranking
├── report_generator.py     → creates JSON report
└── utils/logger.py         → logging system
```

The architecture ensures clarity, easy debugging, and simple extensions in the future.

---

7. Design Diagrams

A. Use Case Diagram

Actor: User
Use Cases:

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

B. Workflow Diagram

1. Start
2. User selects input method
3. Load/enter marks
4. Validate data
5. Calculate statistics
6. Analyze performance
7. Generate report
8. End

C. Sequence Diagram

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

D. Class Diagram

Classes:

- InputHandler
- StatsCalculator
- PerformanceAnalyzer
- ReportGenerator
- Logger

---

8. Design Decisions & Rationale

- **Modular approach:** Makes the project clean and easy to maintain.
- **CSV input support:** Most teachers store marks in Excel, so CSV support is practical.
- **JSON output:** Lightweight, readable, and good for data exchange.
- **Logging:** Helpful for debugging and tracking invalid entries.
- **Statistics module:** Ensures accuracy and reduces manual coding.
- **CLI interface:** Keeps the project simple and avoids GUI complexity.

---

9. Implementation Details

Technologies

- Python 3
- Modules: `csv`, `statistics`, `json`, `logging`

Code Files

- `main.py` handles program flow
- `input_handler.py` manages data input
- `stats_calculator.py` performs all statistical calculations
- `performance_analyzer.py` evaluates overall performance
- `report_generator.py` creates JSON files
- `logger.py` tracks logs

Input Requirements

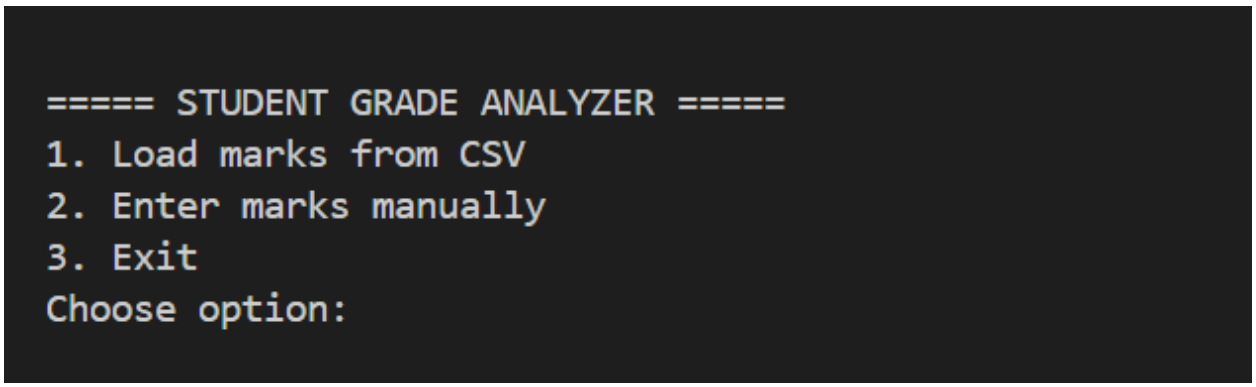
CSV Format:

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

---

10. Screenshots

1. Program start menu



2. Program output showing statistics

```
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. Generated `report.json`

report.json > ...


```json
{
  "statistics": {
    "mean": 62.0,
    "median": 60.5,
    "mode": 44.0,
    "variance": 740.0,
    "std_dev": 27.202941017470888,
    "highest": 98.0,
    "lowest": 22.0
  },
  "performance": {
    "total_students": 10,
    "passed": 7,
    "failed": 3,
    "pass_percentage": 70.0,
    "top_performers": [
      98.0,
      95.0,
      89.0
    ],
    "weak_students": [
      22.0,
      36.0,
      37.0
    ]
  }
}
```

4. `grade_analyzer.log` contents

---

11. Testing Approach

Manual Tests

- Loaded CSV with valid marks
- Tried CSV with invalid rows
- Entered marks manually
- Tested empty input
- Tested with large number of marks

Expected Results

- Program should calculate correct statistics
- Errors should be logged properly
- Invalid inputs should be handled safely

---

12. Challenges Faced

- Handling mode when the dataset has no unique mode
- Managing invalid rows in CSV files
- Keeping the project modular but simple
- Making sure the output JSON is readable

---

13. Learnings

- Using Python’s built-in statistics methods
- Working with CSV files
- Importance of cleaning and validating data
- Designing modular programs
- Using logging for debugging
- Generating machine-readable reports like JSON

---

14. Future Enhancements

- Add graphs using matplotlib
- Support multiple subjects
- Export report as PDF
- Add a Tkinter or web-based UI
- Store student data in a database

---

15. References

- Python Documentation
- Python statistics module
- CSV handling tutorials
- Logging module official docs
