1. PROJECT REPORT

Unit-Test Result Analyzer

Submitted by: Hafizur Rahman

Register Number: 25BAI11372

Submitted to :- Dr. Manimaran

2. Introduction

In the modern academic landscape, the timely and accurate analysis of student examination results is critical for educational institutions. Traditional methods of result analysis often involve manual calculations using spreadsheets or physical registers. These methods are labor-intensive, prone to human error, and fail to provide immediate visual insights into class performance.

The Unit-Test Result Analyzer is a Python-based software solution designed to automate this process. By leveraging powerful data science libraries—Pandas for data manipulation and Matplotlib for visualization—this tool transforms raw Excel grade sheets into comprehensive analytical reports. It not only calculates pass percentages and identifies toppers but also visualizes the data, enabling faculty to make data-driven decisions regarding student performance and curriculum adjustments.

3. Problem Statement

Faculty members and academic administrators face several challenges when processing examination results manually:

1. **High Volume of Data:** Processing results for multiple subjects across large batches of students (60+) is time-consuming.
2. **Complex Metrics:** Manually calculating specific metrics like "Pass Percentage excluding Withdrawn students" or identifying "Subject-wise Toppers" requires complex filtering that is difficult to do manually.
3. **Lack of Visualization:** A spreadsheet full of grades does not immediately reveal trends, such as a high failure rate in a specific subject or a skewed grade distribution.
4. **Error Susceptibility:** Manual counts of "Arrears" (Failures) often lead to discrepancies in official records.

The Solution: A Python automation script that reads the standard university result Excel sheet and outputs error-free statistical data and graphs instantly.

4. Functional Requirements

The system is designed to fulfill the following specific functions:

1. **Excel Integration:** The system must accept user input for the specific Excel filename and Sheet name to allow flexibility across different semesters.
2. **Data Cleaning:** It must automatically detect and convert the 'Registration Number' column to an integer format to remove decimal points often added by Excel.
3. **Topper Identification:** The system must implement a priority logic (O > A+ > A > B+ > B) to find the highest grade in a subject and list all students who secured it.

4. **Failure Analysis:** It must categorize non-performing students into distinct groups:
    - **U:** Re-appear (Fail)
    - **UA:** Absent
    - **WH:** Withheld
    - **W:** Withdrawn
5. **Graphical Representation:** It must generate a bar chart for every subject that visualizes the frequency of every grade type.

5. Non-Functional Requirements

- **Performance:** The script must process a dataset of up to 500 students and generate reports in under 10 seconds.
- **Reliability:** The Pass Percentage calculation must be mathematically accurate to two decimal places.
- **Usability:** The application must run in a Command Line Interface (CLI) with clear prompts, requiring no coding knowledge from the end-user.
- **Scalability:** The code should be dynamic enough to handle any number of subject columns without hard-coding column names.

---

6. System Architecture

The project follows a modular architecture centered around the Python Data Science stack.

1. **Data Input Layer:**
    - The user inputs the file path.
    - Pandas library (pd.read_excel) handles the ingestion of .xlsx files.
2. **Processing Layer:**
    - **DataFrame Operations:** The data is stored in a Pandas DataFrame.
    - **Logic Functions:** Custom functions (subjectwise_topper, subjectwise_arrear) iterate through the data to filter, sort, and count values.
3. **Visualization Layer:**

    - Matplotlib (plt) takes the statistical counts and renders them into Bar Charts.
4. **Output Layer:**
    - Textual analysis is printed to the console.
    - Graphs are displayed in pop-up windows.

7. Design Diagrams

7.1 Workflow Diagram

Flow:

Start -> Input File Name -> Load Excel -> Extract Columns -> Loop through Subject Columns -> Calculate Topper & Arrears -> Plot Graph -> End

7.2 Use Case Diagram

- **Primary Actor:** Faculty / Examiner.
- **System Boundary:** Result Analyzer Application.
- **Use Cases:**
    - Load Data Sheet.
    - View Subject Statistics.
    - View Pass Percentage.
    - Visualize Grade Distribution.

8. Design Decisions & Rationale

1. Why Python?
   Python was chosen over Java or C++ because of its superior libraries for data analysis (Pandas). Doing this in C++ would require writing complex file parsers from scratch.
2. Why Matplotlib?
   It is the industry standard for basic plotting in Python. It is lightweight and integrates seamlessly with Pandas.

3. Grade Priority Logic:
   The topper function uses a hardcoded priority list (['O', 'A+', 'A', 'B+', 'B']). This decision was made because grades are categorical, not numerical. We cannot simply use max() on text data. The script iterates through this list; the first grade found in the column is deemed the "highest," and the loop breaks.

9. Implementation Details (Algorithm)

The implementation is divided into modular functions to ensure clean code.

Module 1: File Reading (read_file)

- **Input:** Filename string from user.

Process:
Python
df = pd.read_excel(file_name, sheet_name=sheet_name)
df['RegNumber'] = df['RegNumber'].astype(int)

- **Output:** A clean DataFrame.

Module 2: Topper Analysis (subjectwise_topper)

- **Algorithm:**
  1. Define priority list P = ['O', 'A+', 'A', 'B+', 'B'].
  2. Iterate through P. Check if grade P[i] exists in the subject column.
  3. If yes, set highest_grade = P[i] and break loop.
  4. Filter DataFrame: toppers = df[df[subject] == highest_grade].
  5. Print names and RegNumbers of toppers.

Module 3: Arrear & Pass % (subjectwise_arrear)

- **Algorithm:**
  1. Initialize counters count_arrears = 0, count_withdrawn = 0.
  2. Iterate through rows.
  3. If the grade is 'U' or 'UA', increment count_arrears.
  4. If the grade is 'W' or 'WH', increment count_withdrawn.

5. Calculate Pass %:

   Pass Percentage = (Total Students - Students with U or UA and W or WH Grades) / (Total Students - Students with W or WH Grades) * 100

Module 4: Visualization (graph)

- **Process:**
  1. Define x-axis labels: grades = ['O', 'A+', 'A', 'B+', 'B', 'U', 'UA'].
  2. Count frequency of each grade in the column.
  3. plt.bar(grades, counts, color=['green', 'blue'...]).
  4. Add labels, title, and show plot.

10. Results & Screenshots

(Paste your screenshots here)

- **Figure 1: Input Excel Sheet.** Shows the raw data structure.
- **Figure 2: Console Output.** Displays the list of toppers, specific counts of absentees, and the calculated pass percentage.
- **Figure 3: Bar Charts.** Visual evidence of class performance. For example, a tall bar on 'O' indicates an easy subject or high performance, while a tall bar on 'U' indicates a difficult subject.

11. Testing Approach

The system underwent "Black Box Testing" where various inputs were provided to check outputs without modifying internal code.

Test Case ID	Test Description	Input Data	Expected Output	Status

TC-01	Valid File Input	result.xlsx	Data loaded successfully	**Pass**
TC-02	Missing Topper	Column with no 'O' grades	System finds 'A+' as topper	**Pass**
TC-03	All Absent	Column with all 'UA'	Pass % = 0.00	**Pass**
TC-04	Empty Rows	Excel with blank rows	Script ignores/handles NaN	**Pass**

12. Challenges Faced

1. **Handling NaN (Not a Number):** Empty cells in Excel (students who didn't register) were initially causing the script to crash. This was resolved by using pandas .dropna() or conditional checks like if grade != grade (a Python trick to check for NaN).
2. **Data Types:** The 'Registration Number' often imported as a float Converting this to integer was necessary for clean output.
3. **User Input Errors:** If the user typed the sheet name incorrectly, the program crashed. Error handling (Try-Except blocks) is recommended for future versions.

13. Learnings & Key Takeaways

- **Data Automation:** Learned how Python can save hours of manual administrative work.
- **Library Mastery:** Gained deep practical knowledge of **Pandas** for slicing and dicing data frames.

- **Visual Communication:** Learned that data is useless without visualization; the bar charts provide immediate value to the user that raw text cannot.

14. Future Enhancements

1. **PDF Report Generation:** Currently, the tool prints to the console. A future upgrade will use the FPDF library to save a neatly formatted PDF containing all text and graphs.
2. **GUI (Graphical User Interface):** Developing a front-end using **Tkinter** so users can click buttons instead of typing commands.
3. **CGPA Calculation:** Extending the logic to calculate the Cumulative Grade Point Average for every student based on credit points.

15. References

1. **Pandas Documentation:** *pandas.pydata.org* - For DataFrame manipulation techniques.
2. **Matplotlib Gallery:** *matplotlib.org/stable/gallery* - For bar chart customization.
3. **Python Official Docs:** *docs.python.org* - For file I/O and basic syntax.