Smart Score Optimizer

1. Applied Real Time Use Scenario: Smart Score Optimizer

Problem Statement:

In the education sector, raw numerical data (marks) can be difficult to analyse quickly. Students and teachers often struggle to visualize comparative performance across different subjects. Traditional report cards provide numbers but lack visual insights, making it harder to identify weak areas or strengths at a glance.

The objective is to develop a Smart Result Visualizer that helps users:

* Input subject scores interactively.
* Validate data to ensure marks fall within a logical range (0-10).
* Prevent errors caused by invalid non-numeric inputs.
* Visualize performance instantly using a Bar Chart for Immediate analysis.

This project is implemented using Python, incorporating Matplotlib for visualization alongside core concepts like dictionaries, exception handling, and input validation.

2. Concepts Used

* **External Libraries** - Using matplotlib.pyplot to generate professional grade graphical representations (bar charts).
* **Data Structures** – Dictionaries are used to map specific subjects to their respective scores, keeping data organized.
* **Exception Handling** - try-except blocks are used to catch non-integer inputs and logical errors and preventing the program from crashing unexpectedly.
* **Input Validation** - Ensuring scores remain within the valid range (0-10) using if conditions and Value Error.
* **Functions** - Modularizing the plotting logic (plot_subject_scores) for cleaner code.

3. Solution Approach

The Smart Result Optimizer is structured into key functional steps:

* **User Input Module** – Accepts scores for Physics, Chemistry, Maths, and Computer Science.
* **Validation Layer** – Checks if the input is a valid integer and falls between 0 to 10, triggers an error if invalid.
* **Data Aggregation** – Aggregates the valid inputs into a Python Dictionary for easy processing.
* **Visualization Engine** – Feeds the dictionary keys (subjects) and values (Scores) into matplotlib to render a bar chart.
* **Error Management** – Gracefully handles crashes using try-except blocks to ensure a smooth user experience.

4. Python Code Implementation

```
import matplotlib.pyplot as plt


def plot_subject_scores(scores):
    try:
        subjects = list(scores.keys())
        values = list(scores.values())


        # Bar chart
        plt.figure(figsize=(6,4))
        plt.bar(subjects, values, color="lightgreen", edgecolor="black")
        plt.title("Subject Scores (out of 10)")
        plt.xlabel("Subjects")
        plt.ylabel("Scores")
        plt.ylim(0, 10)  # since max is 10
        plt.show()


    except Exception as e:
        print(f"Error while plotting chart: {e}")


try:
    physics = int(input("Enter Physics score (out of 10): "))
    chemistry = int(input("Enter Chemistry score (out of 10): "))
    maths = int(input("Enter Maths score (out of 10): "))
```

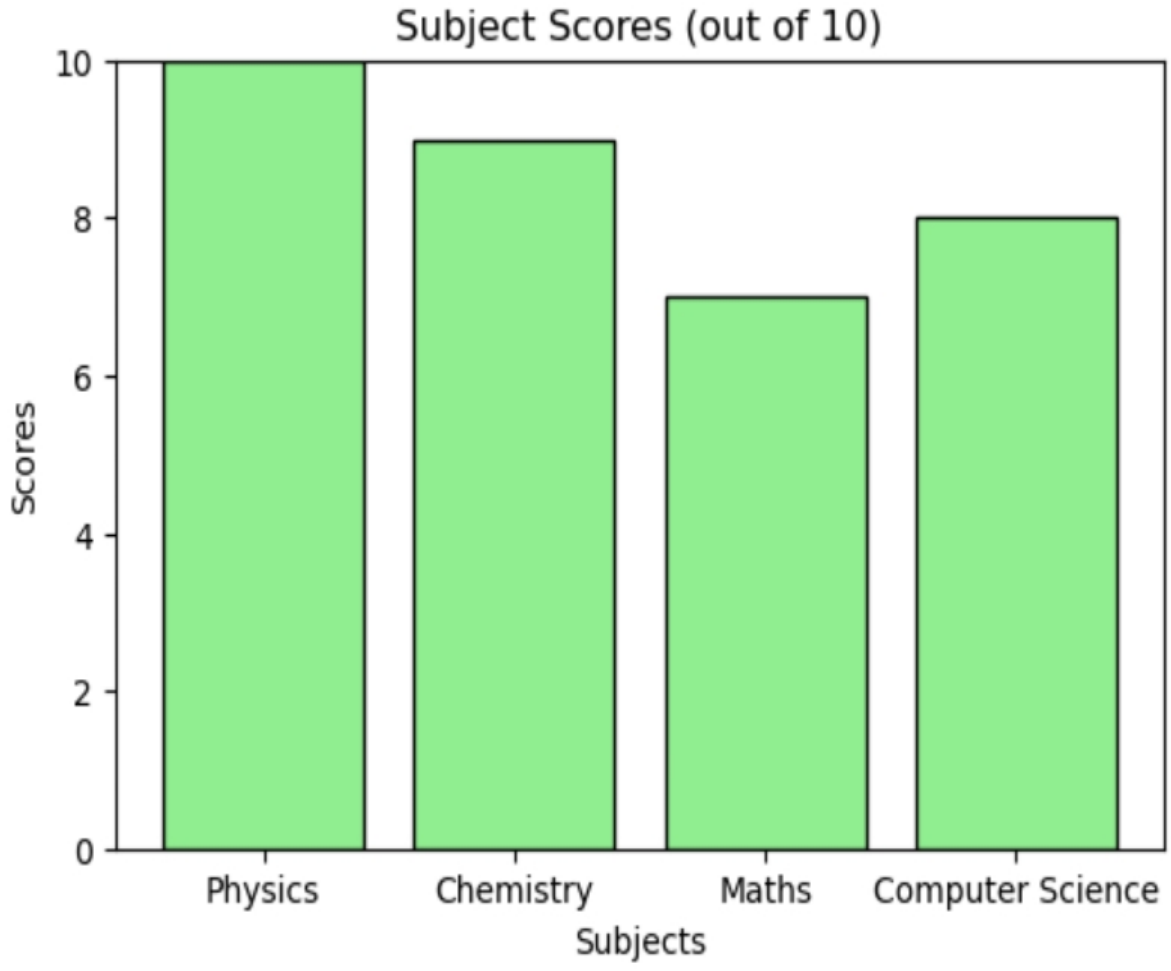
```
    comp_sci = int(input("Enter Computer Science score (out of 10): "))

    # Validate inputs
for val in [physics, chemistry, maths, comp_sci]:
        if val < 0 or val > 10:
            raise ValueError("Scores must be between 0 and 10.")

    scores = {
        "Physics": physics,
        "Chemistry": chemistry,
        "Maths": maths,
        "Computer Science": comp_sci
    }

    print("\n✅ Scores entered successfully!")
    print(scores)

    plot_subject_scores(scores)

except ValueError as ve:
    print(f"Input Error: {ve}")
except Exception as e:
    print(f"Unexpected Error: {e}")
```

code

```
import matplotlib.pyplot as plt

def plot_subject_scores(scores):
    try:
        subjects = list(scores.keys())
        values = list(scores.values())

        # Bar chart
        plt.figure(figsize=(6,4))
        plt.bar(subjects, values, color="lightgreen", edgecolor="black")
        plt.title("Subject Scores (out of 10)")
        plt.xlabel("Subjects")
        plt.ylabel("Scores")
        plt.ylim(0, 10)  # since max is 10
        plt.show()

    except Exception as e:
        print(f"Error while plotting chart: {e}")

try:
    physics = int(input("Enter Physics score (out of 10): "))
    chemistry = int(input("Enter Chemistry score (out of 10): "))
    maths = int(input("Enter Maths score (out of 10): "))
    comp_sci = int(input("Enter Computer Science score (out of 10): "))
    # Validate inputs
    for val in [physics, chemistry, maths, comp_sci]:
        if val < 0 or val > 10:
            raise ValueError("Scores must be between 0 and 10.")

    scores = {
        "Physics": physics,
        "Chemistry": chemistry,
        "Maths": maths,
        "Computer Science": comp_sci
    }

    print("\n✅ Scores entered successfully!")
    print(scores)

    # Call chart function
    plot_subject_scores(scores)

except ValueError as ve:
    print(f"Input Error: {ve}")
except Exception as e:
    print(f"Unexpected Error: {e}")
```

Code Output

```
Enter Physics score (out of 10):  10
Enter Chemistry score (out of 10):  9
Enter Maths score (out of 10):  7
Enter Computer Science score (out of 10):  8

✅ Scores entered successfully!
{'Physics': 10, 'Chemistry': 9, 'Maths': 7, 'Computer Science': 8}
```



5. Code Breakdown & Insights

*** Data Visualization (Matplotlib)**

* The code uses plt.bar() to create a graphical representation of the marks.
* plt.ylim(0, 10) ensures the graph scale is consistent, making it easier to compare bars visually.

*** Robust Input Validation**

* A custom validation loop checks if val < 0 or val > 10.
* If a user enters an invalid number (e.g., 12 or -5), the program raises a ValueError immediately, preventing incorrect graphs.

*** Dictionary Usage**

* scores.keys() and scores.values() are extracted dynamically which allows plotting without hardcoding.

*** Exception Handling**

* The try-except block wraps the entire input process. If a user enters text (e.g., "ten") instead of a number, the program catches the error and prints a friendly message instead of crashing.

6. 📌 Advanced Discussion & Scalability

👨‍💼 **Interviewer**: Currently, this system generates a plot on the screen. How would you modify this to generate reports for 1,000 students automatically?

👨‍🎓 **Student**: Instead of using plt.show() which pauses the code to show a window, I would use plt.savefig(f"{student_id}_report.png"). This allows the script to loop through 1,000 students, generate their graphs in the background, and save them as image files automatically without human intervention.

Thank You

Name - Gaurav Rathore

Roll No -14

Subject - Principles of Programming Language

Course – BSc.Data Science