Assignment 1: HealthStat – A Patient Data Analysis & Visualization API

Problem Statement:

You are hired as a backend data analyst at a health tech startup. Your task is to build a small **FastAPI-based web service** that provides **insights** from a simulated dataset of patients. You will use **object-oriented design** to handle data processing and **FastAPI** to serve this data as endpoints.

Objectives:

- Practice **object-oriented design** for data handling.
- Apply **data analysis** using **Pandas** and **NumPy**.
- Create visualizations using **Matplotlib/Seaborn**.
- Expose insights as **REST API endpoints** using **FastAPI**.

Tasks & Instructions:

Data:

Use a CSV file named `patients_data.csv` with the following structure (generate random values if needed):

PatientID, Age, Gender, Weight, Height, BP_Systolic, BP_Diastolic, Cholesterol, Diabetes, Smoker

Task Breakdown:

1. **Data Ingestion & Cleaning** (Pandas):
    a. Load the CSV using Pandas.
    b. Handle any missing values (drop or fill).

    c. Create a new column `BMI = Weight / (Height/100)^2`.
2. **Object-Oriented Structure**:
    a. Create a `Patient` class with attributes (ID, age, gender, etc.).
    b. Create a `PatientAnalyzer` class with methods:
        i. `get_average_bmi()`
        ii. `get_gender_distribution()`
        iii. `get_high_risk_patients()` – return patients with BMI > 30, High BP, or Cholesterol > 240
3. **Data Visualization**:
    a. Create a **seaborn barplot**: Average BMI by gender.
    b. Create a **matplotlib histogram**: Age distribution.
    c. Save both graphs as `.png` files.
4. **FastAPI Implementation**:
    a. Endpoint: `/stats/bmi` – returns average BMI.
    b. Endpoint: `/stats/gender-distribution` – returns gender count.
    c. Endpoint: `/risk/high` – returns high-risk patient list.
    d. Endpoint: `/charts/bmi-gender` – serves the BMI barplot as image.
    e. Endpoint: `/charts/age-dist` – serves the age histogram.

📦 Deliverables:

- Python file(s) with FastAPI app and class definitions.
- Charts saved locally and accessible via API.
- Well-commented code and clear README with instructions to run the API.

Assignment 2: EduAI – Student Grade Intelligence Engine

Problem Statement:

You are building a data-driven dashboard for a university. It will handle **student performance data**, extract patterns using Python and OOP, and expose APIs that can later be integrated into a frontend dashboard.

Objectives:

- Use **Python functions and classes** effectively.
- Use **Pandas/NumPy** to calculate and transform data.
- Visualize **performance trends**.
- Serve insights using **FastAPI** APIs.

Tasks & Instructions:

Data:

Simulate `students_scores.csv` with:

StudentID, Name, Gender, Semester, Subject, Marks

Include at least 5 subjects and 50 students (randomly assign).

Task Breakdown:

1. **Data Aggregation & Transformation**:
    a. Use Pandas to:
        i. Pivot to get average marks per student.
        ii. Create grade buckets: A (85+), B (70–84), C (50–69), F (<50)
        iii. Count number of students in each grade per subject.
2. **Object-Oriented Implementation**:
    a. Create a `Student` class.

    b. Create a GradeBook class with methods:
        i. `get_top_performers(n)`
        ii. `get_subject_average(subject)`
        iii. `get_failures_by_subject()`
3. **Visualization**:
    a. Seaborn heatmap of student performance (Student vs Subject).
    b. Matplotlib pie chart of overall grade distribution.
4. **FastAPI Implementation**:
    a. Endpoint: `/students/top/{n}` – returns top n students.
    b. Endpoint: `/subjects/average/{subject}` – returns average mark in that subject.
    c. Endpoint: `/subjects/failures` – returns count of failures per subject.
    d. Endpoint: `/visuals/heatmap` – serve seaborn heatmap.
    e. Endpoint: `/visuals/grades-pie` – serve pie chart.

📦 Deliverables:

- Clean and modular OOP-based implementation.
- API routes as described.
- Saved heatmaps and pie charts.
- README with example usage and instructions.

🧰 Tools Required:

- Python 3.x
- Pandas, NumPy
- Seaborn, Matplotlib
- FastAPI, Uvicorn
- Optional: Faker (to simulate realistic names)