Name: Aashika pandey
Roll Number: 150096725010
Subject: Python

Case Study 17: COVID-19 Data Visualization Dashboard

1. Introduction

The COVID-19 Data Visualization Dashboard is a Python-based mini-project that analyses city-wise COVID-19 case statistics and presents them as a text dashboard and a comparative chart. The system reads data from a CSV file, performs health metrics calculations, identifies critical cities, and visualizes trends using Matplotlib.[file:image.jpg]

2. Objectives

- To design a simple health dashboard that shows real-time style statistics for multiple cities.
- To apply object-oriented concepts (classes, methods) for data modelling and analysis.
- To practise file handling using CSV input and generate summary reports.
- To implement basic automation features using decorators and lambda expressions.
- To visualise COVID-19 trends using a multi-bar chart and a growth-rate trend line.[file:image.jpg]

---

3. System Requirements

3.1 Hardware Requirements

- Any standard computer or laptop.
- Minimum 4 GB RAM recommended.
- At least 200 MB free disk space for Python and libraries.

3.2 Software Requirements

- Operating System: Windows / macOS / Linux.

- Python 3.x installed.
- Python libraries:
    - matplotlib
    - Standard library modules: csv, os.
- Code editor (VS Code / PyCharm / any IDE).

---

4. System Design

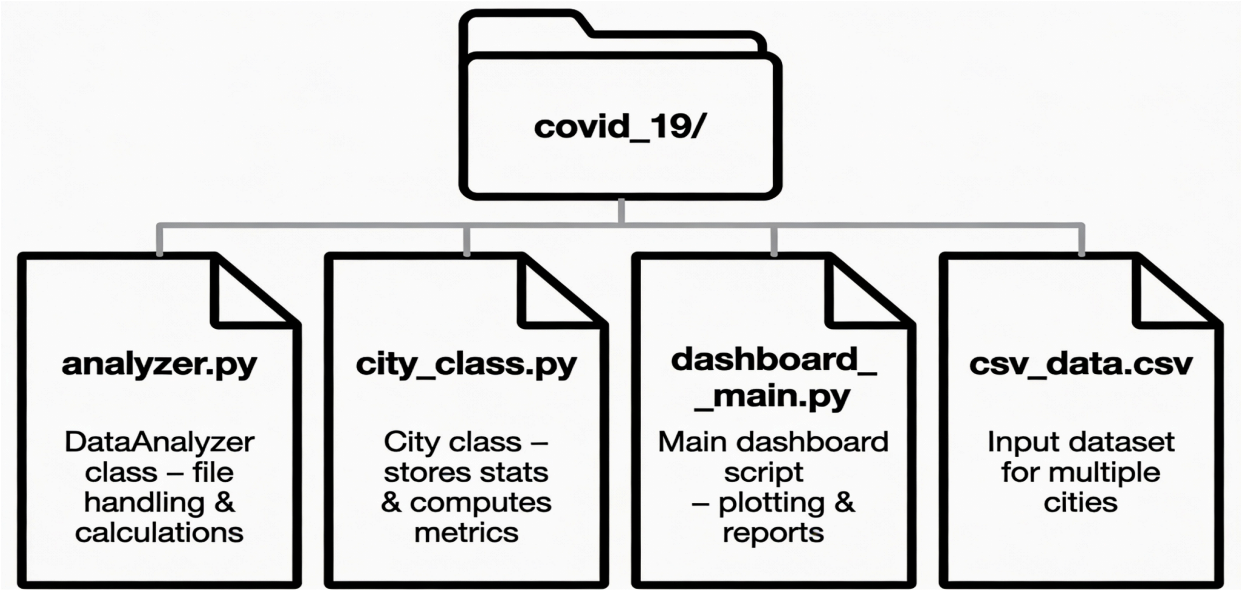
4.1 Module Description

1. `city_class.py` – City Module
    - Represents a single city and its COVID-19 statistics.
    - Attributes: name, confirmed, recovered, deaths, active, prev_confirmed.
    - Methods:
        - recovery_rate() – calculates recovery percentage.
        - mortality_rate() – calculates mortality percentage.
        - growth_rate() – calculates growth based on previous confirmed cases.
        - severity_level() – classifies the city as High / Medium / Low severity.
        - __str__() – returns formatted information for the dashboard.
2. analyzer.py – DataAnalyzer Module
    - Handles file reading and high-level calculations.
    - Responsibilities:
        - Build a safe path to **csv_data.csv** using **os.path.dirname(__file__)** and **os.path.join()**.
        - **load_data()** – reads each row from the CSV file and creates *City* objects.
        - **critical_cities(threshold)** – returns names of cities with active cases above threshold.
        - **average_recovery_rate()**, **average_mortality_rate()**, **average_growth_rate()** – compute dataset-level averages.[file:Screenshot-2025-12-17-at-12.58.11-AM.jpg]
3. **dashboard_main.py** – Dashboard & Visualization Module
    - Contains the main program entry point.
    - Uses a decorator @dashboard_header to print a formatted dashboard title automatically.
    - Creates a DataAnalyzer object, loads data, and prints per-city statistics (recovery, mortality, growth rate, severity).

- Generates a stacked multi-bar chart of Confirmed, Recovered, and Active cases with a red growth-rate line on a secondary Y-axis.
- Contains a Report class that prints a summary of average recovery, mortality, and growth rates.[file:Screenshot-2025-12-17-at-1.05.47-AM.jpg]

4. csv_data.csv – Data File
    - Stores historical COVID-19 data for multiple cities.
    - Sample columns: *City, Confirmed, Recovered, Deaths, Active, PrevConfirmed.*
    - Can be updated to simulate new days or new cities.

4.2 Folder Structure



---

5. Implementation Details

5.1 Object-Oriented Programming

- The project follows a modular OOP design:
    - The city acts as the data model.
    - DataAnalyzer is the analysis layer.
    - Report + dashboard_main.py form the presentation layer.
- Encapsulation is achieved by keeping calculations inside methods of each class, improving reusability and readability.

5.2 Automation Features

- Decorator:
  - **dashboard_header** decorator wraps the **run_dashboard()** function and prints a common heading before execution.
  - Shows how cross-cutting concerns can be added without changing business logic.
- Lambda Expression:
  - **fmt_percent = lambda x: f"{x:.2f}%"** is used to format numeric rates into neat percentages.
  - Keeps display logic concise and demonstrates lambda usage

5.3 Data Analysis Logic

- Recovery, mortality, and growth rates are calculated using simple percentage formulas based on confirmed, recovered, death, and previous confirmed counts.
- Severity level is derived from active cases with fixed thresholds to highlight cities that need more attention.

5.4 Visualization

- Matplotlib is used to generate:
  - Stacked bar chart for Confirmed, Recovered, Active cases per city.
  - Secondary Y-axis red line showing growth rates as trend indicators.
  - Legends, axis labels, and titles are added for better interpretation.[file:Screenshot-2025-12-17-at-1.05.47-AM.jpg]

---

6. Execution Flow

1. The user runs **dashboard_main.py.**
2. Decorator prints the dashboard header.
3. **DataAnalyzer** object is created and loads data from `csv_data.csv`.
4. **City** objects are created for each row and stored in a list.
5. The program prints city-wise stats, severity levels, and critical cities.
6. The multi-bar chart and growth-rate trend line are displayed.
7. The **Report** class prints overall summary statistics

---

7. Conclusion

The COVID-19 Data Visualization Dashboard successfully demonstrates how Python can be used to build a small but complete analytic tool. It combines file handling, object-oriented concepts, decorators, lambda expressions, and data visualisation to analyse city-level COVID-19 data. The system helps users quickly compare cities, identify critical areas, and understand key health metrics through both textual and graphical outputs.

GitHub Repository

Link to the project repository:

GitHub Link:

References

Python Official Documentation – https://docs.python.org/

Pandas Documentation – https://pandas.pydata.org/docs/

NumPy Documentation – https://numpy.org/doc/

Matplotlib Documentation – https://matplotlib.org/stable/

World Health Organization (WHO) – BMI and health statistics resources: https://www.who.int/