Capstone Project

CodeHouse Speedtest – AI Internet Speed Monitoring CLI

Project Overview

In this project, you will build an **AI-powered Command-Line Interface (CLI)** tool that:

- Monitors their computer's **upload**, **download**, and **ping** speeds.
- Uses **Poisson probability distribution** to **detect anomalies** in speed patterns.
- Displays real-time results in a beautiful CLI interface using the `rich` library.
- Logs performance data to a `.csv` file for later analysis.

Learning Objectives

By the end of the project, you will:

1. Understand and apply **Python CLI development** concepts.
2. Learn how to perform **network speed tests** using the `speedtest` library.
3. Analyze speed data using **NumPy, Pandas**, and **Statistics**.
4. Apply **Poisson Distribution** for real-world **AI-based anomaly detection**.
5. Present results neatly using **Rich Tables** and Progress Bars.
6. Save and interpret performance logs with **Pandas** and **CSV**.

Prerequisites

Before starting, ensure that:

- Python 3.8+ is installed.
- The following libraries are installed:

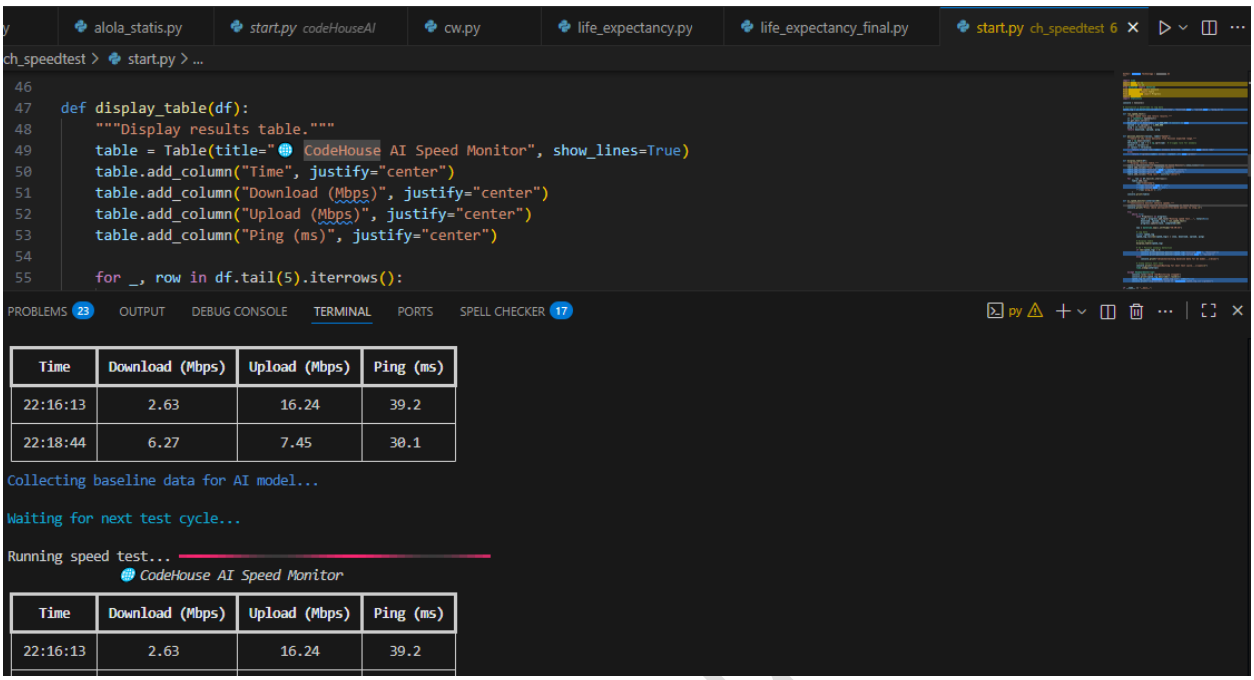
```
pip install speedtest-cli numpy pandas rich
```

Optional (for testing):

```
pip install notebook
```

CODEHOUSE CLOUD genius@codehouse.cloud www.codehouse.cloud
+2348037137902, +2347069374780

RESULT SAMPLE OF WORKING PROGRAM



Step-by-Step Project Development Plan

Phase 1: Project Setup

Goal: Set up the environment and understand the purpose of the project.

Steps:

1. Create a project folder named `codehouse_speedtest`.
2. Inside, create a Python script file named `speed_monitor.py`.
3. Write the header section with author, description, and imports.
4. Test your terminal to ensure you can run:
5. `python speed_monitor.py`
6. Research: “What is Poisson distribution and how is it used in anomaly detection?”

Deliverable:

- Project directory created and Python environment ready.
- **Written a short 1-page document explaining Poisson distribution in your own words.**

Phase 2: Implement Speed Testing

Goal: Run and log internet speed tests.

Steps:

1. Use `speedtest` to measure download, upload, and ping.
2. Return and print results in Mbps and ms.
3. Append test results to a Pandas DataFrame.
4. Save test results to a `.csv` file.

Deliverable:

- A working function `run_speed_test()` returning speed values.
- A CSV file showing test logs.

Phase 3: Data Logging & Visualization

Goal: Display data attractively in the console using `rich`.

Steps:

1. Create a `display_table(df)` function to show last 5 test results.
2. Use `rich.table.Table` to format results.
3. Use `rich.console.Console` for colored messages.
4. Use `rich.progress.Progress` to show a loading bar while testing.

Deliverable:

- CLI output with formatted table and progress bar.
- Screenshot of the CLI output.

Phase 4: AI-Powered Anomaly Detection

Goal: Apply Poisson distribution to detect abnormal speed readings.

Steps:

1. Implement the `poisson_monitor()` function.
2. Use the formula:

$$\text{threshold} = \lambda + 3\sqrt{\lambda}$$

   where λ = mean of the observed speeds.

3. Compare the latest speed test result to the threshold.
4. Print:
   - `[bold red]Anomaly detected[/bold red]` if it exceeds threshold.
   - `[green]Normal[/green]` if within normal range.

Deliverable:

- Working anomaly detection system.
- Test report showing detection of abnormal speed values.

Phase 5: Real-Time Monitoring

Goal: Automate continuous monitoring.

Steps:

1. Create a loop that runs tests every set interval (e.g., every 2 minutes).
2. Add `try-except KeyboardInterrupt` to safely stop the program.
3. On stop, save results and show summary statistics using `df.describe()`.
4. Allow configurable interval (via function parameter).

Deliverable:

- Fully working CLI that monitors speed continuously.
- CSV file with full session data.

Phase 6: Testing, Analysis & Documentation

Goal: Test and evaluate the program's performance.

Steps:

1. Run the program multiple times under different conditions (e.g., streaming, idle, downloads).
2. Plot results in using Pandas, seaborns and Matplotlib.
3. Write a final report including:
    - Overview of your implementation.
    - Sample outputs and screenshots.
    - Observations on anomalies detected.
    - Lessons learned and improvements for future.
4. Push your project, including the above documents to github.

Deliverable:

- Project report (PDF or DOCX).
- Final working code.
- Visualized performance graph.