Release Document

Emilia-Romagna region exposure to potential risk and hazard to flooding and its impact on households

Sarmiento Ospina, Nataly Alejandra

Saud-Miño, Claudia Isabela

Wang, Xinmeng

Sayegh, John Cullen

Deliverable: RD

Title: Release Document

Authors: Sarmiento Ospina, N. A., Saud-Miño, C. I., Sayegh, J.C., Wang, X.

Version: 1.0

Date: 01/07/2024

Download page:

Purpose:

The script run_app.py aims to streamline the startup process of a web server and a Jupyter Notebook. By using Python's subprocess module, it ensures that both processes are initiated and monitored correctly, handling any interruptions gracefully.

Detailed Implementation:

1. **Starting the Server:**
    - Utilize subprocess.Popen to run webserver.py, initiating the server in a separate process.
2. **Checking Server Status:**
    - Implement a loop that periodically sends a GET request to the server's URL (e.g., http://localhost:5000) using the requests library.
    - If the server is not up, wait for a short duration (1 second) before retrying.
3. **Starting Jupyter Notebook:**
    - Once the server is confirmed to be running, use subprocess.Popen to start dashboard.ipynb (Jupyter Notebook) in another separate process.
4. **Process Management:**
    - The script waits for both the server and the Jupyter Notebook to keep running.
    - Catch KeyboardInterrupt (e.g., via Ctrl+C) to terminate both processes gracefully.

Explanation:

1. **Starting the Server:**
    - start_process function initializes a subprocess with the given command.
    - The server is started with server_command = ['python', 'webserver.py'].
2. **Checking Server Status:**
    - check_server function sends a GET request to the server URL.
    - The loop keeps retrying every second until the server responds successfully.
3. **Starting Jupyter Notebook:**
    - Once the server is confirmed running, the Jupyter Notebook is started with notebook_command = ['jupyter', 'notebook', 'dashboard.ipynb'].
4. **Process Management:**
    - The script sets up a signal handler (terminate_processes) to catch KeyboardInterrupt and terminate both processes gracefully.
    - signal.signal(signal.SIGINT, terminate_processes) registers this handler.

- The script then waits for both processes to complete using wait.

This script ensures that the server and Jupyter Notebook start sequentially and run concurrently, simplifying the application's startup process.