Test Report: elevation_api.py

1. Introduction

This report details the unit testing performed on the elevation_api.py module. This module contains two classes, ElevationAPI and OpenElevationAPI, which are responsible for fetching elevation data from external web services based on geographical coordinates. The primary goal of these tests is to ensure the reliability and robustness of these API clients under various conditions, including successful responses and different failure scenarios.

2. Testing Approach

All tests were implemented using Python's built-in unittest framework. A key aspect of our testing strategy was the extensive use of **mocking** to isolate the ElevationAPI and OpenElevationAPI classes from their external dependencies (i.e., actual web API calls).

- **Mocking requests.post**: The unittest.mock.patch decorator was used to replace requests.post with a Mock object. This allowed us to:
  - Prevent actual HTTP requests from being made, making tests fast and reliable.
  - Control the return values of requests.post to simulate different API responses (success, various error codes).
  - Simulate network issues by making requests.post raise exceptions (e.g., requests.ConnectionError).
  - Verify that requests.post was called with the correct URL and payload.
- **Mocking sys.stdout**: The unittest.mock.patch decorator combined with io.StringIO was used to capture any output printed to the console (via print() statements) by the elevation_api functions. This allowed us to assert that the correct error messages were displayed to the user during various failure scenarios.

3. Tested Components

The following classes and their get_elevations methods were thoroughly tested:

- ElevationAPI
- OpenElevationAPI

4. Detailed Test Scenarios and Edge Cases Covered

For both ElevationAPI and OpenElevationAPI, the following test scenarios and edge cases were covered:

4.1. Handling Empty List of Points

- **Purpose**: To ensure the get_elevations method gracefully handles situations where no geographical points are provided as input.

- **Edge Case Covered**: Input list points is empty.
- **Expected Behavior**:
  - The method should return an empty list ([]).
  - No actual API call (requests.post) should be attempted.

4.2. Successful API Response

- **Purpose**: To verify that the get_elevations method correctly processes a successful API response and extracts the elevation data as expected.
- **Mocking Strategy**:
  - requests.post is mocked to return a Mock object where ok is True.
  - The content of the mock response is set to a JSON string representing a successful API response with sample elevation data. (The JSON structure was tailored to each API: {"height": [...]} for ElevationAPI and {"results": [{"elevation": ...}]} for OpenElevationAPI).
- **Assertions**:
  - The returned list of elevations matches the expected values from the mocked response.
  - requests.post was called exactly once with the correct API URL and the appropriately formatted JSON payload for the given points (e.g., {"shape": [...]} for ElevationAPI, {"locations": [...]} for OpenElevationAPI).

4.3. Unsuccessful API Response (HTTP Errors)

- **Purpose**: To test the error handling when the external API returns an HTTP error code (e.g., 400 Bad Request, 404 Not Found, 500 Internal Server Error).
- **Edge Case Covered**: API returns a non-2xx status code.
- **Mocking Strategy**:
  - requests.post is mocked to return a Mock object where ok is False and a sample status_code is set (e.g., 404 or 500).
  - sys.stdout is mocked to capture printed error messages.
- **Assertions**:
  - The method should return an empty list ([]).
  - The correct API-specific error message ("API request failed: Data could not be retrieved" or "API request failed: open-elevation data could not be retrieved") is printed to sys.stdout.

4.4. requests.ConnectionError

- **Purpose**: To ensure the method gracefully handles network connectivity issues (e.g., no internet connection, DNS resolution failure).
- **Edge Case Covered**: A requests.ConnectionError is raised during the API call.

- **Mocking Strategy**:
  - requests.post.side_effect is set to requests.ConnectionError("Test error message"), causing the mock to raise this specific exception when called.
  - sys.stdout is mocked to capture printed error messages.
- **Assertions**:
  - The method should return an empty list ([]).
  - The correct API-specific connection error message ("Connection error: Data could not be retrieved" or "Connection error: open-elevation data could not be retrieved") is printed to sys.stdout.

4.5. Other Unexpected Exceptions

- **Purpose**: To verify the module's robustness against any other unforeseen runtime errors that might occur during the API call or subsequent data processing (e.g., malformed JSON that isn't caught by response.ok).
- **Edge Case Covered**: A generic Exception is raised.
- **Mocking Strategy**:
  - requests.post.side_effect is set to a generic Exception("Some unexpected issue").
  - sys.stdout is mocked to capture printed error messages.
- **Assertions**:
  - The method should return an empty list ([]).
  - The generic unexpected error message, including the exception details ("Unexpected error: Some unexpected issue"), is printed to sys.stdout.

5. Conclusion

All developed test cases for ElevationAPI and OpenElevationAPI in elevation_api.py have passed successfully. This indicates that both API client classes are robustly handling expected successful responses, various API-level failures, network issues, and general unexpected errors. The mocking strategy effectively isolated the unit under test, leading to reliable and fast tests.