Test Details for test_plotter.py

This document outlines the comprehensive testing strategy applied to the Plotter class within plotter.py. The tests leverage pytest for its testing framework and unittest.mock for effective mocking, ensuring thorough validation of the Plotter's functionality without actual graphical rendering.

1. Mocking Strategy (mock_matplotlib Fixture)

The mock_matplotlib fixture is a cornerstone of this test suite, designed to isolate the Plotter class's logic from matplotlib's graphical output.

- **Purpose:** To prevent matplotlib from opening actual plot windows during test execution. This significantly speeds up tests, eliminates reliance on a display environment, and makes the test suite suitable for continuous integration (CI) pipelines.
- **Implementation Details:**
  - It uses patch('plotter.plt') to replace the matplotlib.pyplot object (aliased as plt in plotter.py) with a MagicMock. This means any call to plt from within the Plotter class's methods will interact with this mock object.
  - Crucially, it simulates the matplotlib object hierarchy:
    - mock_plt.figure.return_value = mock_fig: Ensures that when plt.figure() is called, it returns a mock figure object (mock_fig).
    - mock_fig.add_subplot.return_value = mock_ax: Configures mock_fig so that calling add_subplot() on it returns a mock axes object (mock_ax). This mimics how 3D plots are set up.
  - **Explicit Mocking:** Key matplotlib.pyplot functions (plot, title, xlabel, show, etc.) and matplotlib.axes methods (plot, set_title, etc.) that Plotter directly calls are individually mocked using MagicMock(). This allows precise assertion of their calls and arguments.
- **Benefit:** Enables tests to verify *what* matplotlib functions are called and with *what arguments*, rather than attempting to validate visual output.

2. Dummy Data Fixtures

To ensure self-contained and repeatable tests, controlled dummy data is used.

- **dummy_elevation_profile:**
  - **Purpose:** Provides a basic, valid ElevationProfile instance.
  - **Details:** It creates a list of Point objects with latitude, longitude, and elevation, then initializes an ElevationProfile with these points. This profile is then used across various tests.
- **another_dummy_elevation_profile:**
  - **Purpose:** Supplies a second, distinct ElevationProfile instance.

    - **Details:** Similar to dummy_elevation_profile, but with different (though equally simple) point data. This is vital for testing scenarios involving multiple profiles (e.g., comparison plots).

3. Test Cases (TestPlotter Class)

The TestPlotter class is structured to systematically test each component of the Plotter class.

a. Initialization Tests (test_init_*)

These tests cover the behavior of the Plotter constructor (__init__) under different input conditions.

- **test_init_no_profiles**:
    - **Check:** Initializes Plotter without any profiles.
    - **Assertion:** plotter.profiles dictionary is empty.
- **test_init_with_single_profile_auto_named**:
    - **Check:** Initializes Plotter with one ElevationProfile instance (without an explicit name).
    - **Assertion:** The profile is added to plotter.profiles with the auto-generated name "Profile 1".
- **test_init_with_single_profile_named**:
    - **Check:** Initializes Plotter with one (ElevationProfile, name) tuple.
    - **Assertion:** The profile is added with the specified name.
- **test_init_with_single_profile_named_none**:
    - **Check:** Initializes Plotter with a profile provided with None as its name.
    - **Assertion:** The profile is correctly auto-named "Profile 1". **(Edge Case)**
- **test_init_with_multiple_profiles_auto_named**:
    - **Check:** Initializes Plotter with multiple ElevationProfile instances.
    - **Assertion:** Profiles are auto-named sequentially (e.g., "Profile 1", "Profile 2").
- **test_init_with_multiple_profiles_mixed_names**:
    - **Check:** Initializes Plotter with a combination of auto-named and explicitly named profiles.
    - **Assertion:** All profiles are added correctly with their respective names.

b. Unique Name Generation Tests (_generate_unique_name)

These tests specifically target the internal helper method _generate_unique_name.

- **test_generate_unique_name_empty**:
    - **Check:** Calls _generate_unique_name on an empty Plotter.
    - **Assertion:** Returns "Profile 1".
- **test_generate_unique_name_existing**:
    - **Check:** Calls _generate_unique_name when "Profile 1" already exists.

    - **Assertion:** Returns "Profile 2".
- **test_generate_unique_name_with_gap**:
    - **Check:** Calls _generate_unique_name when names like "Profile 1" and "Profile 3" exist, creating a gap.
    - **Assertion:** Correctly identifies and returns "Profile 2" (the first available sequential name). **(Edge Case)**

c. Adding Profiles Tests (add_profiles)

These tests validate the add_profiles method's ability to handle various inputs for adding profiles.

- **test_add_profiles_single_auto_name**:
    - **Check:** Adds a single ElevationProfile instance.
    - **Assertion:** Profile is auto-named "Profile 1".
- **test_add_profiles_single_named**:
    - **Check:** Adds a single (ElevationProfile, name) tuple.
    - **Assertion:** Profile is added with the provided name.
- **test_add_profiles_multiple_named**:
    - **Check:** Adds multiple profiles, each with an explicit name.
    - **Assertion:** All profiles are correctly added with their specified names.
- **test_add_profiles_mixed_input**:
    - **Check:** Adds a mix of ElevationProfile instances (auto-named) and (ElevationProfile, name) tuples (explicitly named) in a single call.
    - **Assertion:** All profiles are added and named correctly.
- **test_add_profiles_duplicate_name_overwrites**:
    - **Check:** Adds a profile with a name that already exists.
    - **Assertion:** The new profile replaces the old one under the same name, and the total count of profiles remains correct (i.e., no new entry is created if the name exists). **(Edge Case)**
- **test_add_profiles_empty_string_name_auto_named**:
    - **Check:** Adds a profile where the name in the tuple is an empty string "".
    - **Assertion:** The profile is correctly auto-named. **(Edge Case)**
- **test_add_profiles_none_name_auto_named**:
    - **Check:** Adds a profile where the name in the tuple is None.
    - **Assertion:** The profile is correctly auto-named. **(Edge Case)**
- **test_add_profiles_invalid_input**:
    - **Check:** Tests various invalid inputs (e.g., plain string, tuple with non-ElevationProfile first element, MagicMock instead of ElevationProfile).
    - **Assertion:** ValueError is raised with the expected error message. **(Edge Case)**

d. Setting Profiles Tests (set_profiles)

These tests ensure the set_profiles method correctly replaces the entire set of profiles.

- **test_set_profiles_valid**:
  - **Check:** Sets profiles using a valid dictionary mapping names to ElevationProfile instances.
  - **Assertion:** plotter.profiles is completely replaced by the new dictionary.
- **test_set_profiles_empty_dict**:
  - **Check:** Sets profiles with an empty dictionary.
  - **Assertion:** All existing profiles are removed, and plotter.profiles becomes empty. **(Edge Case)**
- **test_set_profiles_invalid_type**:
  - **Check:** Attempts to set profiles using a non-dictionary input (e.g., a list).
  - **Assertion:** TypeError is raised. **(Edge Case)**
- **test_set_profiles_invalid_value**:
  - **Check:** Attempts to set profiles with a dictionary containing non-ElevationProfile objects as values.
  - **Assertion:** ValueError is raised, indicating an invalid profile type. **(Edge Case)**

e. Plotting Method Tests (plot_distance_vs_elevation and plot_3d_lat_lon_elevation)

These tests confirm that the plotting methods correctly interact with the mocked matplotlib library without producing actual plots. The focus is on verifying method calls and their arguments.

- **test_*_no_profiles (for both 2D and 3D plots)**:
  - **Check:** Calls plotting methods when Plotter contains no profiles.
  - **Assertion:** A "No profiles to plot." message is printed to standard output (captured by capsys). Crucially, no matplotlib functions (plt.figure, plt.plot, plt.show, etc., or their ax equivalents) are called. **(Edge Case)**
- **test_plot_distance_vs_elevation_single_profile**:
  - **Check:** Calls the 2D plotting method with a single profile and custom labels/title.
  - **Assertion:** mock_matplotlib.figure, mock_matplotlib.plot, mock_matplotlib.title, mock_matplotlib.xlabel, mock_matplotlib.ylabel, mock_matplotlib.grid, mock_matplotlib.legend, mock_matplotlib.tight_layout, and mock_matplotlib.show are all called exactly once with the expected arguments.
- **test_plot_distance_vs_elevation_multiple_profiles**:
  - **Check:** Calls the 2D plotting method with multiple profiles.
  - **Assertion:** mock_matplotlib.figure is called once. mock_matplotlib.plot is called once for *each* profile (verified by call_count and call_args_list to check actual data passed).

Other matplotlib methods are called once.

- **test_plot_3d_lat_lon_elevation_single_profile**:
  - **Check:** Calls the 3D plotting method with a single profile and custom labels/title.
  - **Assertion:** mock_matplotlib.figure is called once. mock_fig.add_subplot is called with projection="3d". mock_ax.plot is called once with latitude, longitude, and elevation data. mock_ax.set_title, mock_ax.set_xlabel, mock_ax.set_ylabel, mock_ax.set_zlabel, mock_ax.legend, mock_matplotlib.tight_layout, and mock_matplotlib.show are all called exactly once with the expected arguments.
- **test_plot_3d_lat_lon_elevation_multiple_profiles**:
  - **Check:** Calls the 3D plotting method with multiple profiles.
  - **Assertion:** mock_matplotlib.figure and mock_fig.add_subplot are called once. mock_ax.plot is called once for *each* profile (verified by call_count and call_args_list). Other relevant mock_ax and mock_matplotlib methods are called once.

This detailed testing approach ensures that the Plotter class is robust, handles various valid and invalid inputs gracefully, and correctly interacts with its matplotlib dependency.

```
PS D:\BS. CS & AI\6th sem project\backend\src\gpx_dataengine> python -m pytest test_plotter.py
============================================= test session starts =============================================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: D:\BS. CS & AI\6th sem project\backend\src\gpx_dataengine
collected 27 items

test_plotter.py ...........................                                                          [100%]

============================================== 27 passed in 1.67s =============================================
```