User Manual - UnitPylot

Table of Contents

1. Introduction

Welcome to the User Manual for **UnitPylot**! This guide will help you understand how to use and get the most out of our extension.

2. Getting Started

2.1 Prerequisites

- Access to Copilot: To use any GitHub Copilot extension in Visual Studio Code, you need either an **active Copilot subscription** (such as Copilot Pro, Copilot Enterprise, or Copilot Business)
- **Visual Studio Code** should be installed
- **Copilot in Visual Studio Code**: Follow this if not yet set-up - http://code.visualstudio.com/docs/copilot/setup

2.2 Installation and Launching the Extension

1. Make sure you have the **GitHub Copilot activated in VS Code**. Follow the link above if you need assisstance.
2. Add our extension by searching for **"UnitPylot"** in the **VS Code Extension Marketplace**.
3. Download an example codebase to try out our extension from here: https://github.com/ucl-syseng-tools-for-vscode/example-codebases

To begin using UnitPylot, follow these steps:

1. Press `F5` *OR* open the **Command Palette (Shift + Command + P)** and run **Debug: Start Debugging**.
2. Open one of the projects within the **example-codebases** folder.
3. Run the `make.sh` file to create a **virtual environment (venv)** to run the project within *OR* ensure that you have the necessary dependencies installed by running: `pip install pytest pytest-cov pytest-json-report pytest-monitor`.

For further instuctions, please follow our installation procedure listed in our `README.md`.

3. Overview of our Features 🌟

📊 Test Performance & Coverage Insights

- **Granular Test Metrics Breakdown**: displays the structure of the project's test suite within a **tree view** highlighting,
  - passing/failing test cases,
  - *n* slowest tests,
  - *n* most memory intensive tests.
- **Line and Branch Coverage Display**: provides functionality that highlights untested areas of code within the editor, providing real-time feedback.
- **Test History Tracker**: tracks test performance with interactive graphs of pass/fail rates and coverage trends.
- **Exportable Logs**: saves test results and coverage trends into *json* or *markdown* formats.

🔁 Automated Test Optimisation & Debugging

UnitPylot offers **AI assitance** that provides suggestions to improve the following metrics, allowing them to be **accepted directly into corresponding files**:

- **Fix Failing Tests**: detects failure points and suggests fixes to improve test reliability.
- **Improve Coverage**: detects untested code such as edge cases or missed branches and suggests additional test cases.
- **Optimise Slowest Tests**: detects the *n* slowest tests and suggests explanations and improved test cases with faster execution time.
- **Optimise Memory-intensive Tests**: detects the *n* most memory intensive tests and suggests tests which use lesser memory.

🤖 AI-Powered Enhancements

- **Code Insights**: highlights vulnerabilities and suggests improvements in test cases to detect bottlenecks and prevent regressions.
- **Pydoc Generation**: generates documentation for test cases to enhance readability and maintainability.
- **AAA Chat Participant**: provides guidance on how to follow the best testing practices by adhering to the AAA design pattern.

⚙️ Smart Execution and Customisation

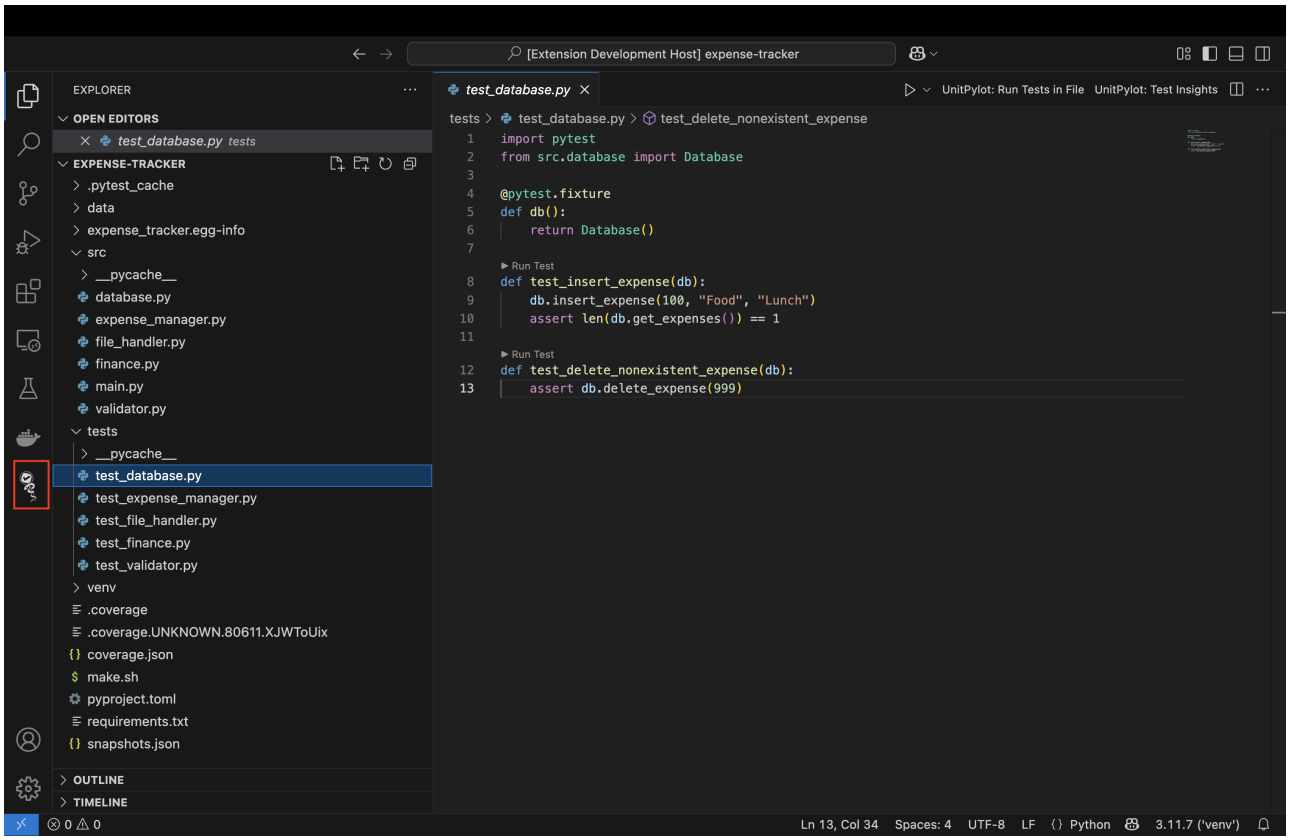
- **Customise *n***: allows user to chose the number of slowest and memory intensive tests to display dynamically.
- **Continuous Background Testing**: runs necessary tests automatically when changes are detected.
- **Refreshing Suite History**: allows user to customise whether to **periodically save snapshots** or **track changes based on file changes**.
- **Selective Test Execution**: allows running only relevant tests based on recent changes to shorten feedback loops.

4. Usage

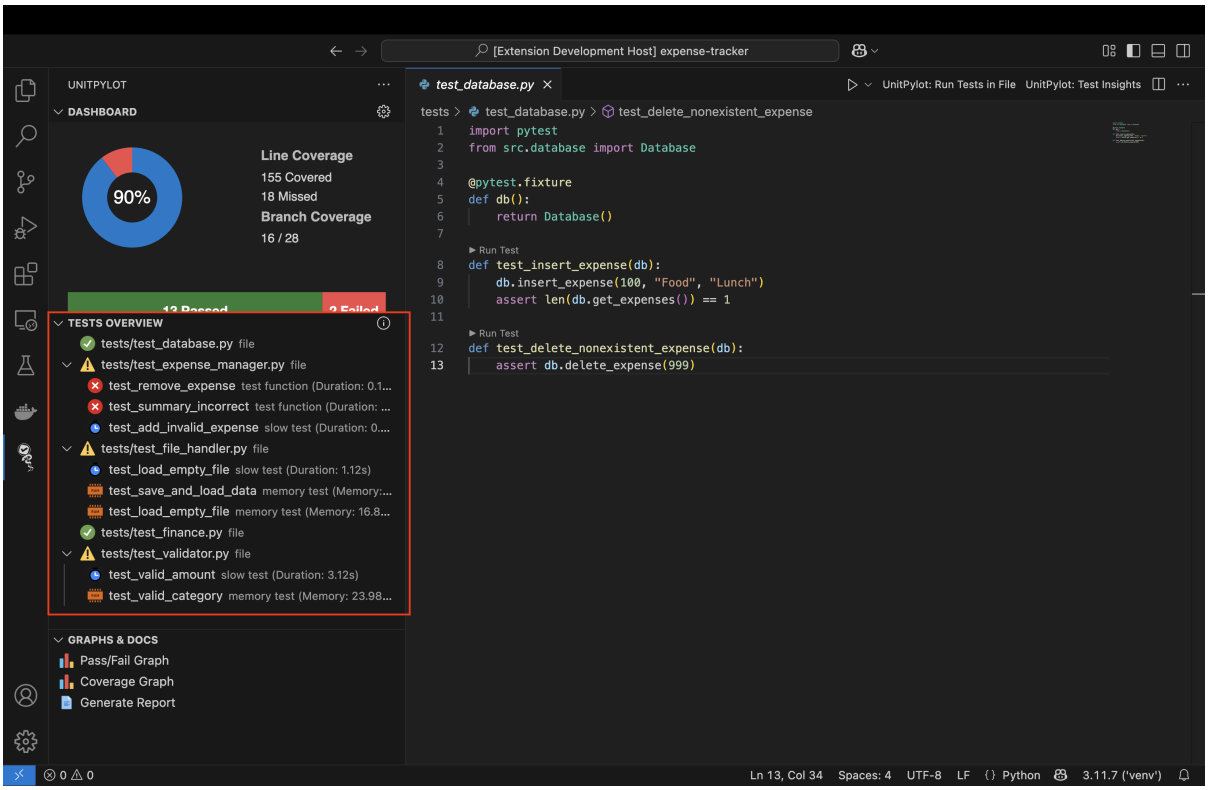
4.1 Usage Instructions 📖

🖥️ Dashboard View

- Locate and open the 🔧 icon on the left-side VSCode navigation bar.

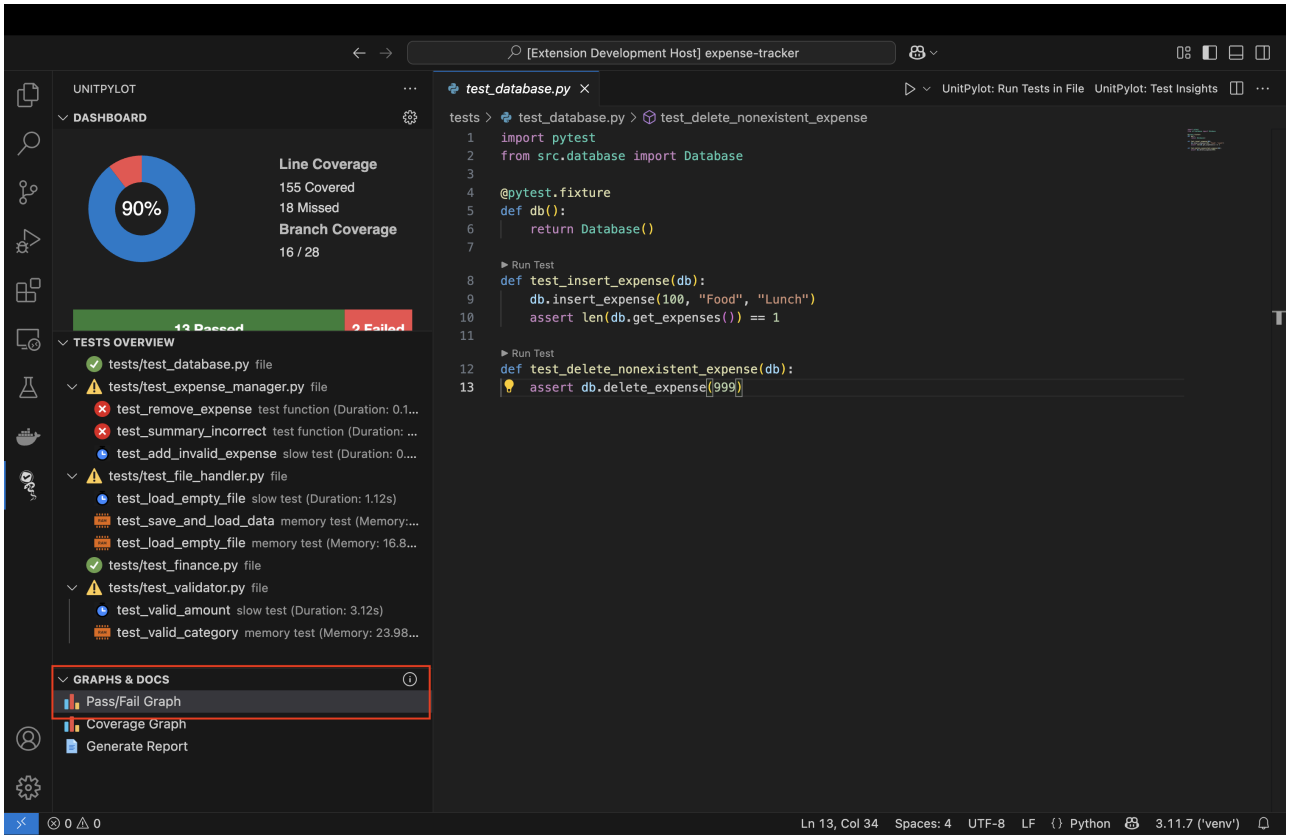


- Access the granular test suite view from the **dashboard view** under the **Tests Overview** collapsable view.
    - When you click on a test case, you are automatically navigated to the relevant code for that test.

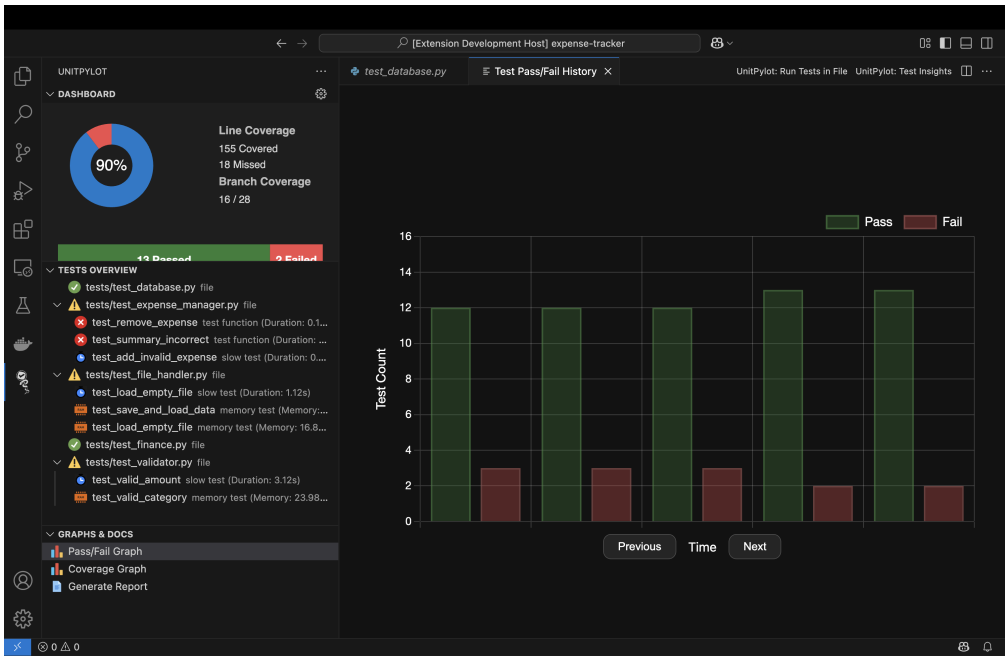


- Access the test history graphs to analyse the pass/fail rates and coverage trends over time from the **dashboard view** under the **Graphs & Docs** collapsable view.

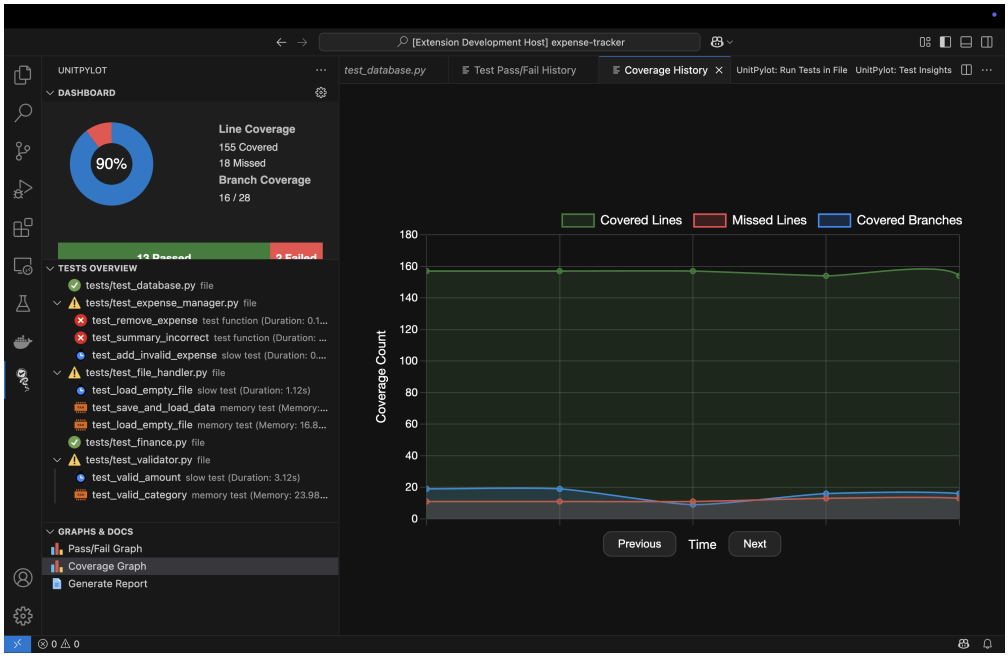
1. First press the relevant graph button on the Dashboard



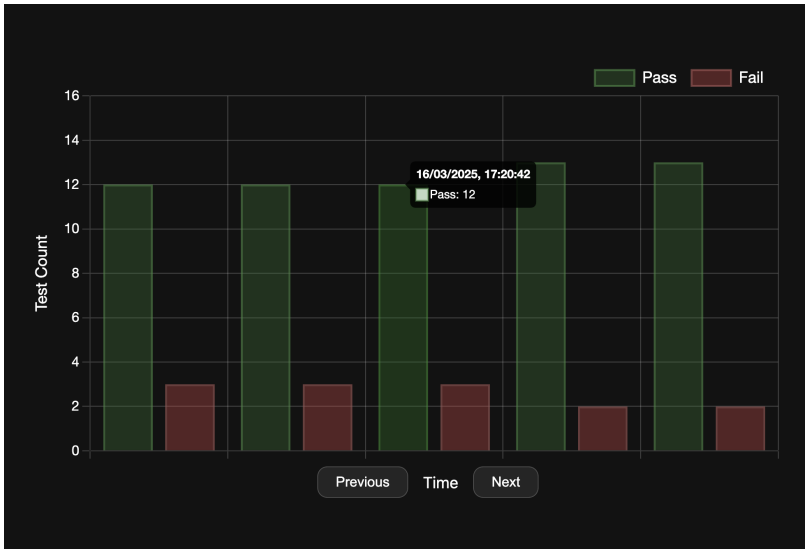
2. A new window will automatically open up, displaying the relevant graph. Use the previous or next buttons to move through the graph display over time. By hovering over a particular point in time, you will see further information, such as the timestamp and number of test cases or covered lies/branches.



Example of the Pass/Fail Graph



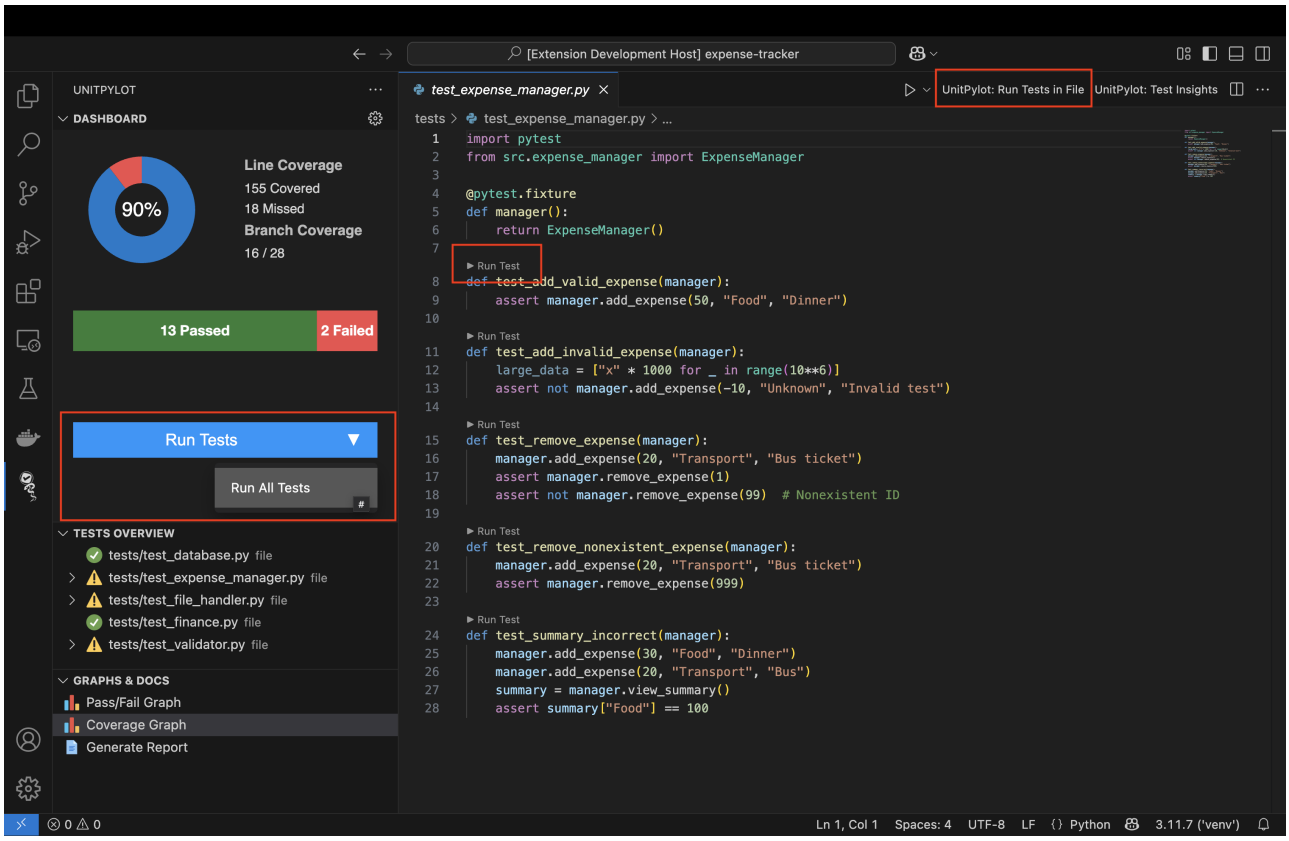
Example of the Coverage Graph



Hover over a data point to see more information!

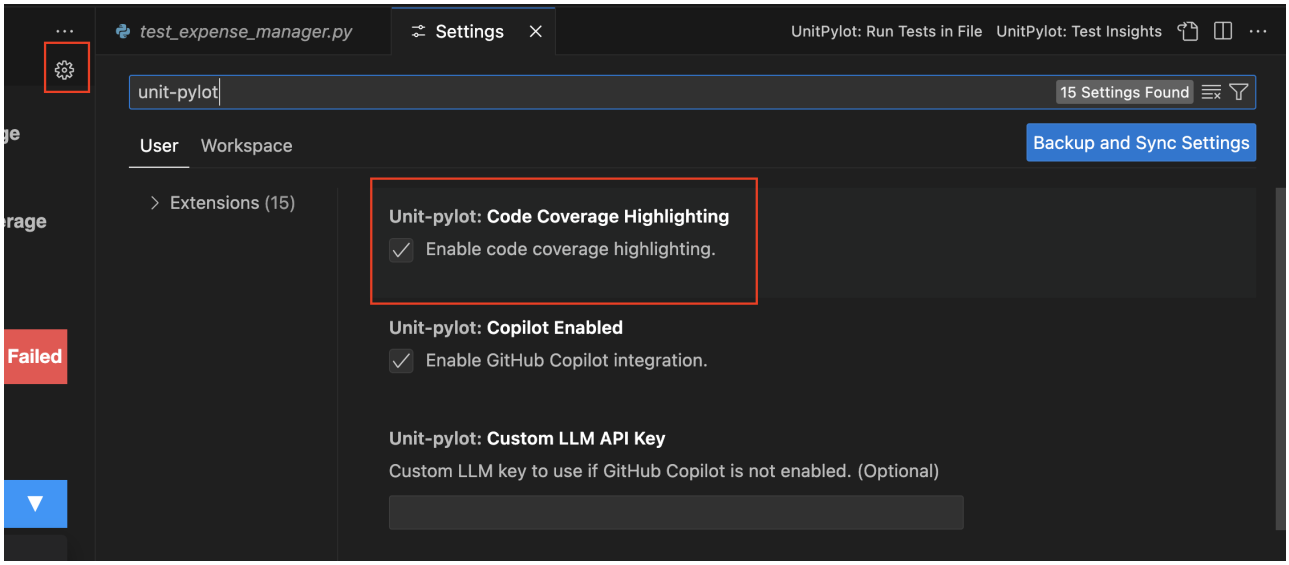
⏯️ Running Tests

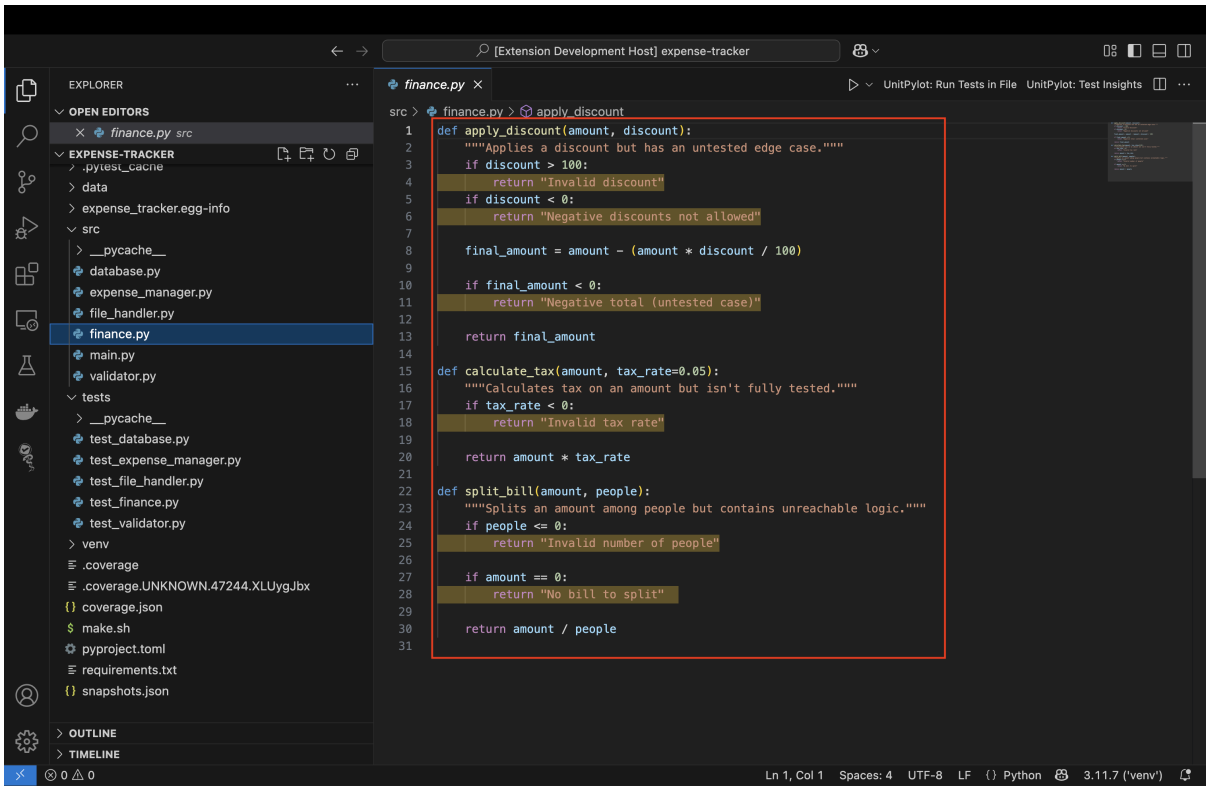
1. Open a Python file with **tests**.
2. Use the CodeLens links above each test function to run or debug particular tests *OR* click the **run tests / run all tests** button within the **dashboard view**.



📈 Viewing Test Coverage

1. Enable **code coverage highlighting** in the **settings**.
2. Run your tests to see the coverage data directly in the editor.



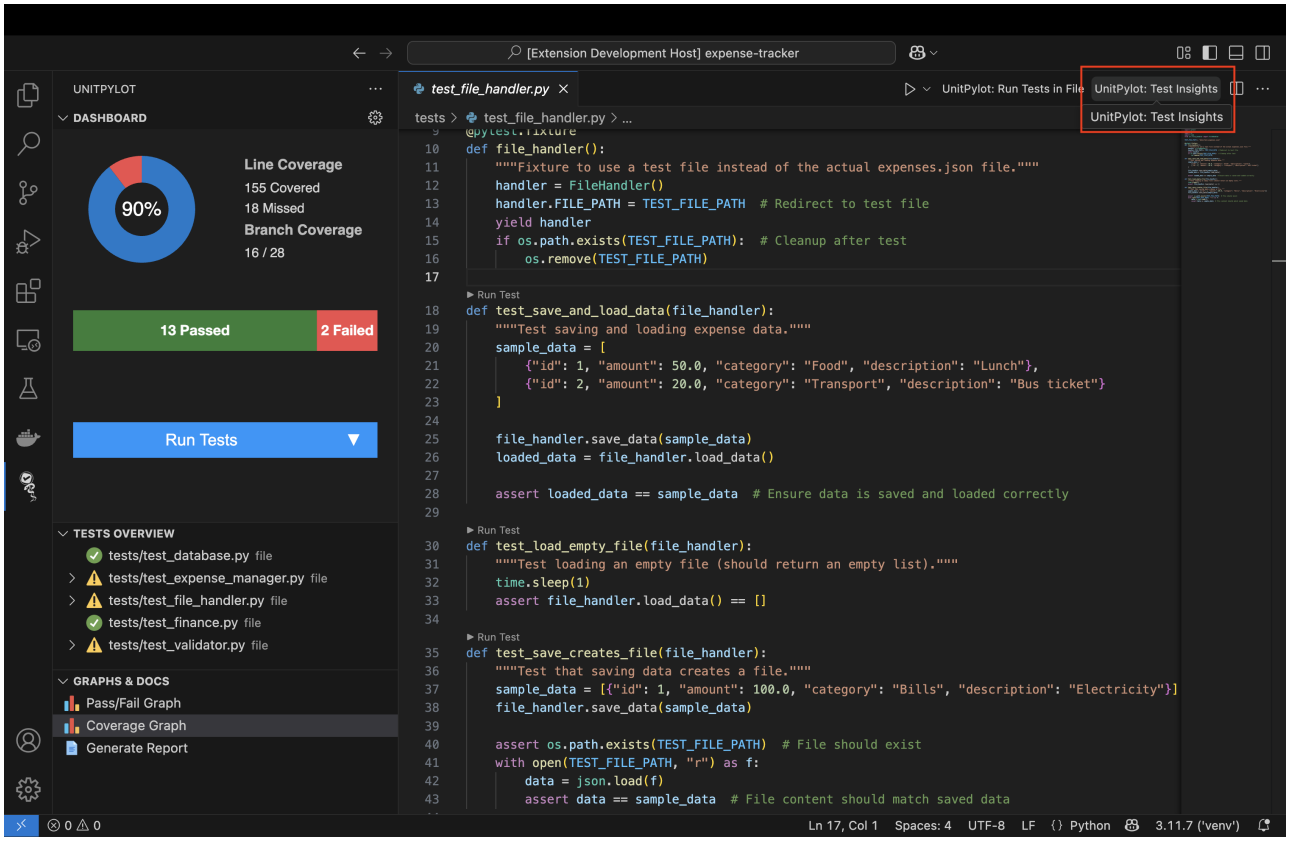


Example of coverage in-line highlighting within the editor

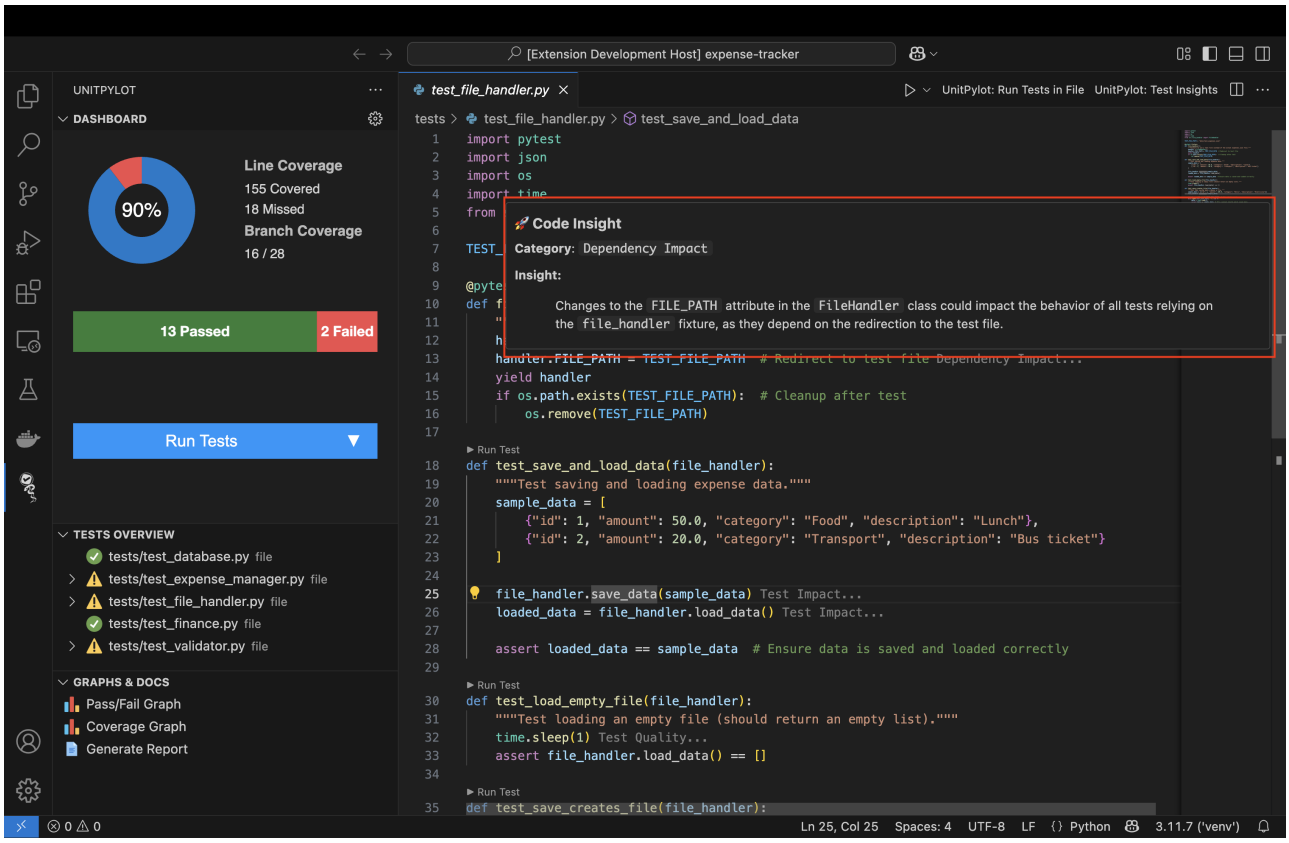
✅ Commands for Optimising Tests

Code Insights

1. Locate the **Code Insights** button on the top right next to the run button to generate code insights.



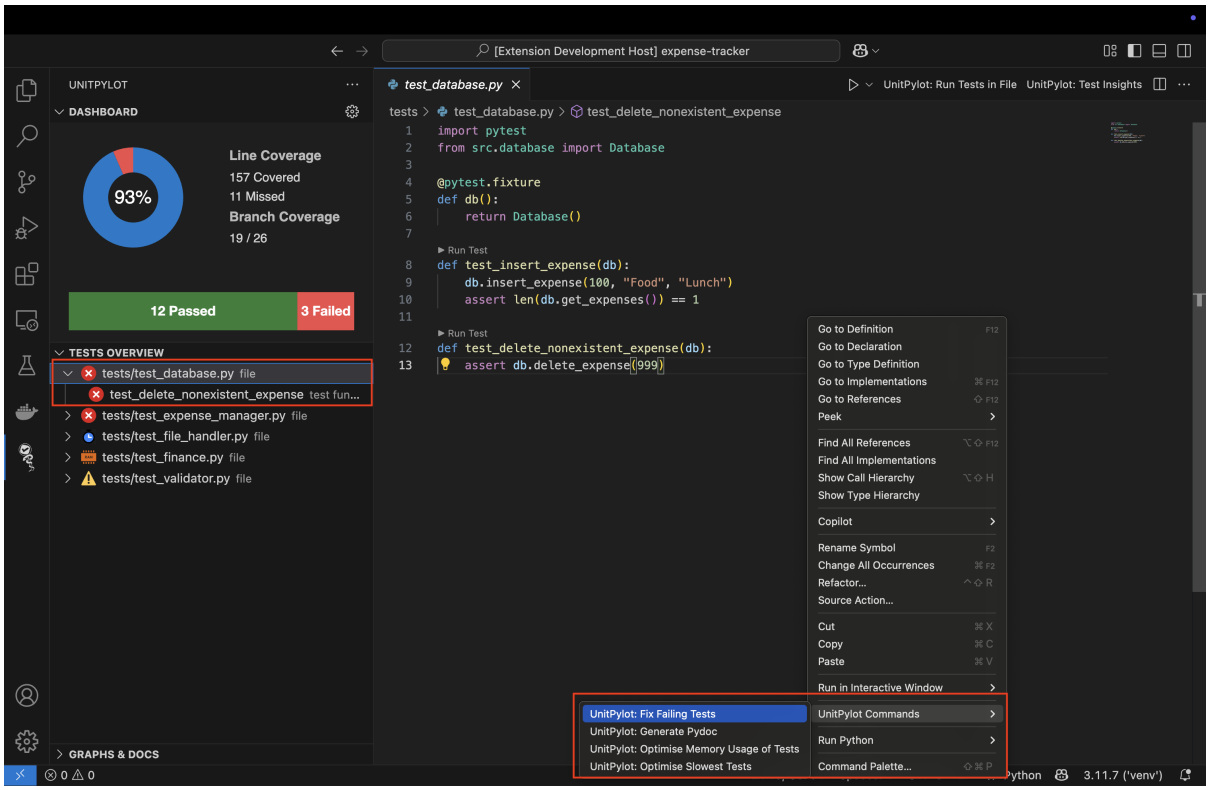
2. Hover over the annotation to view each insight! Annotations automatically disappear once you leave the file.



UnitPylot Commands

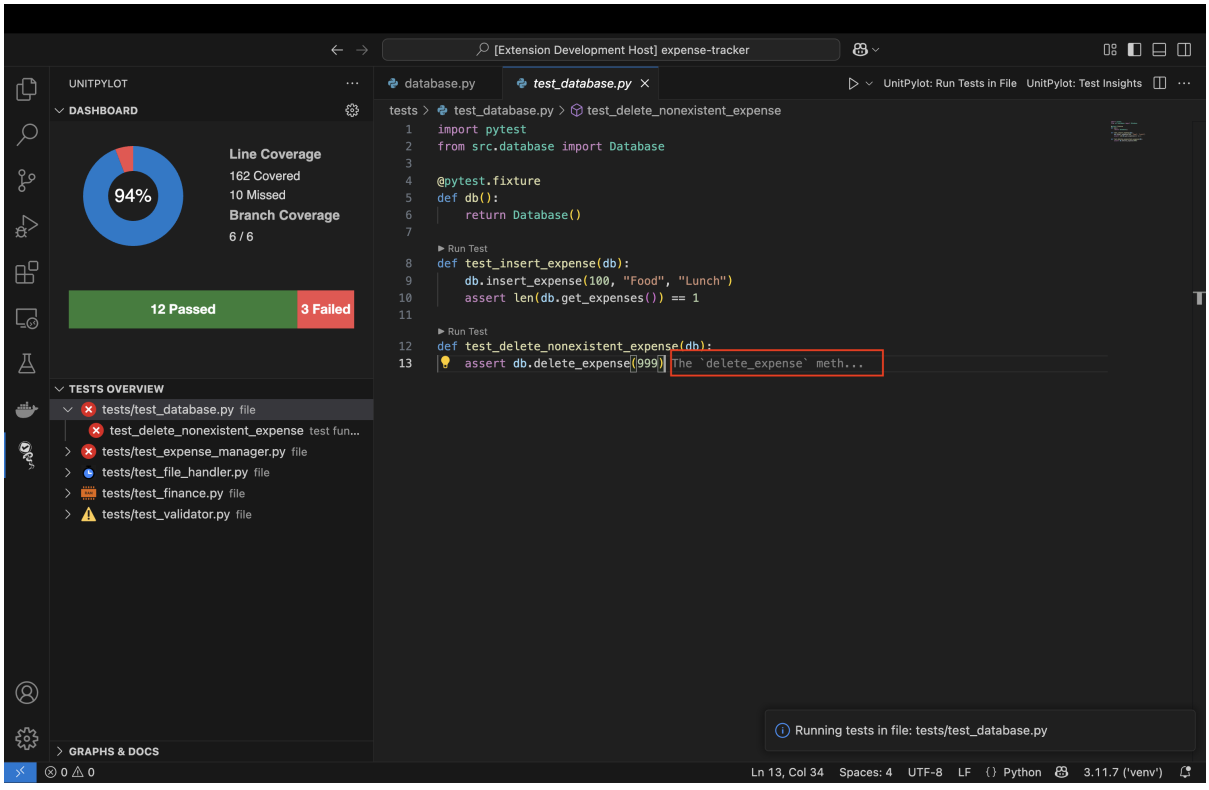
- Right click and navigate to the **UnitPylot Commands** sub-menu to find:
  - the fix coverage command when in a source file.
  - the fix failing, optimise slowest, optimise memory, and generate pydoc commands when the current editor is in a test file.

1. First open the sub-menu by **right-clicking** anywhere in the editor.



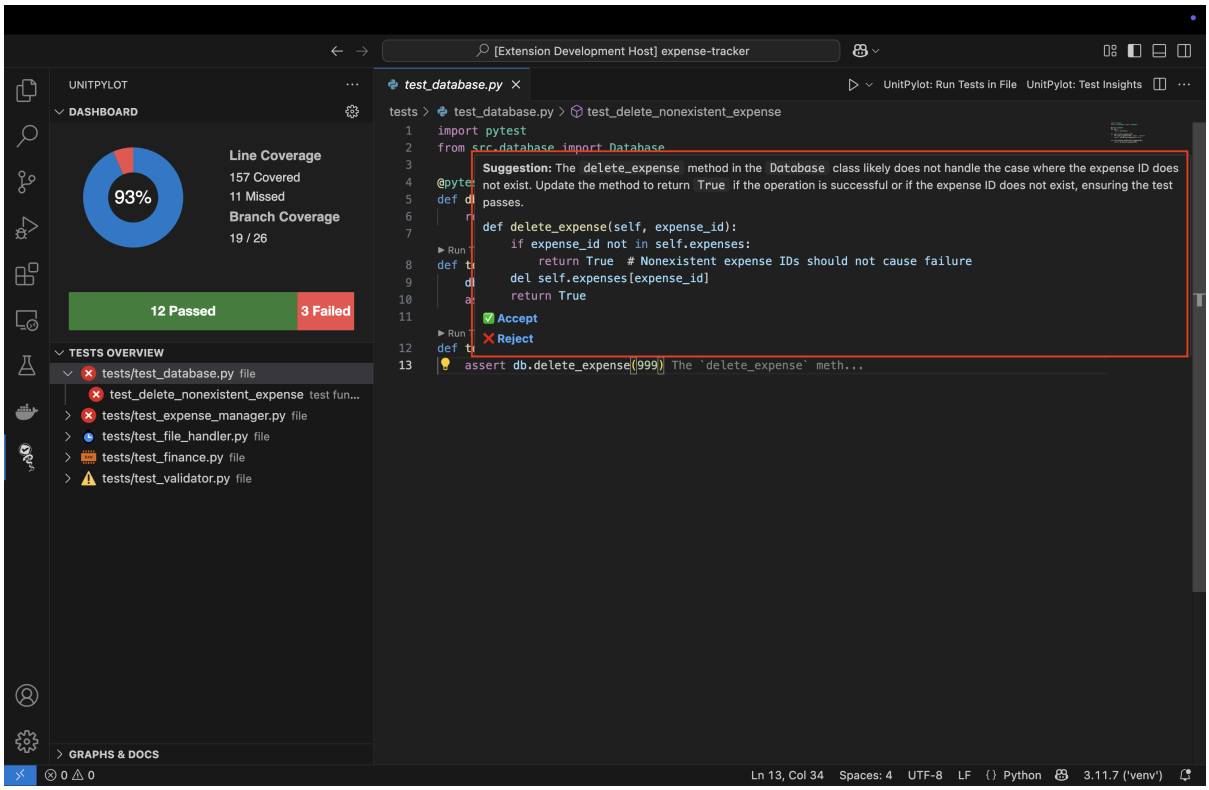
Accessing the command via the submenu (In this example, the `Fix Failing ` command is executed)

2. The relevant test case will then be annotated.



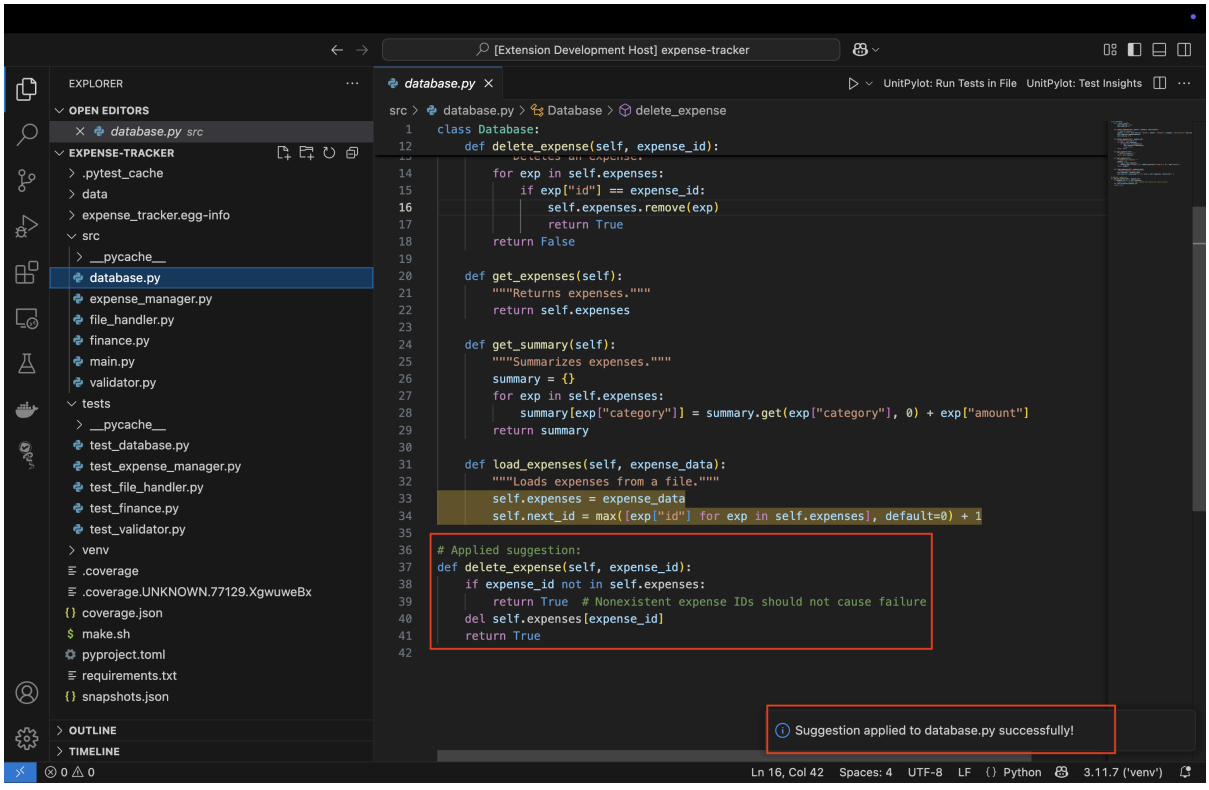
Suggestion is being displayed in-line

3. Hover over the annotation to see a pop-up with the option to accept or reject the suggestion.

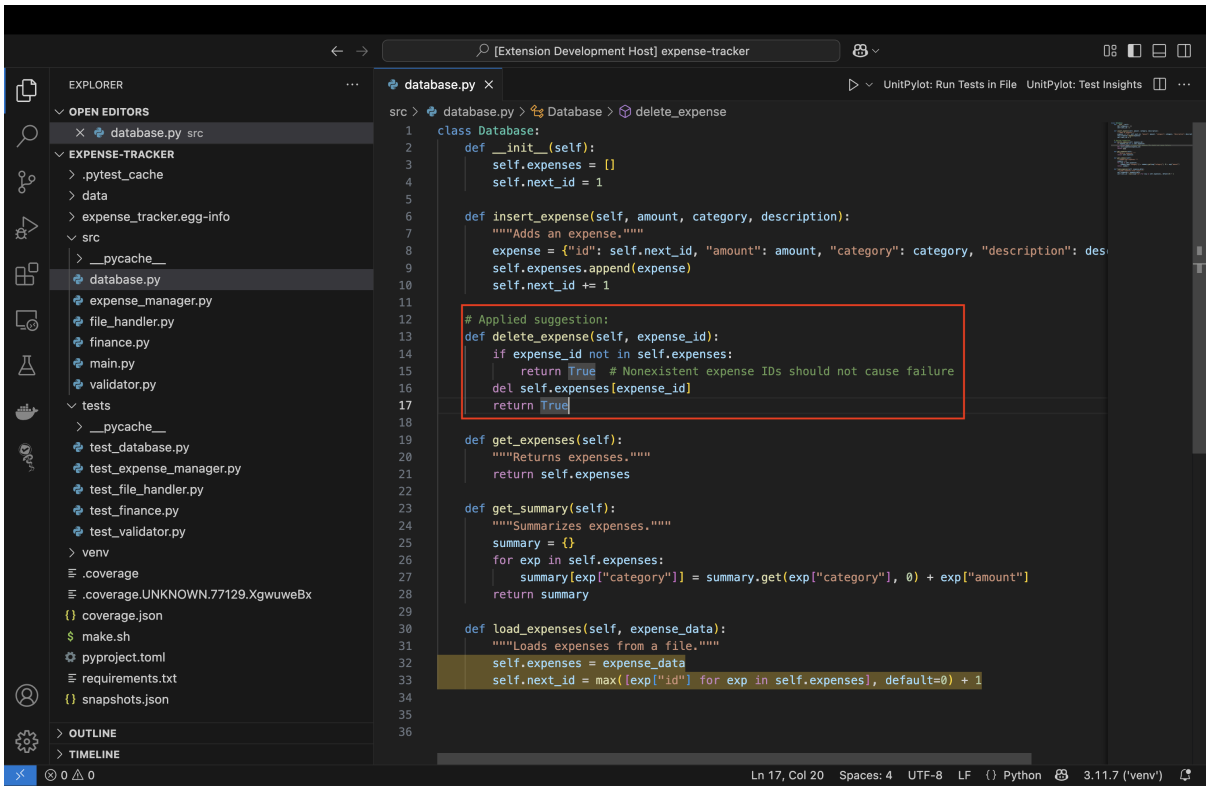


Viewing the suggestion

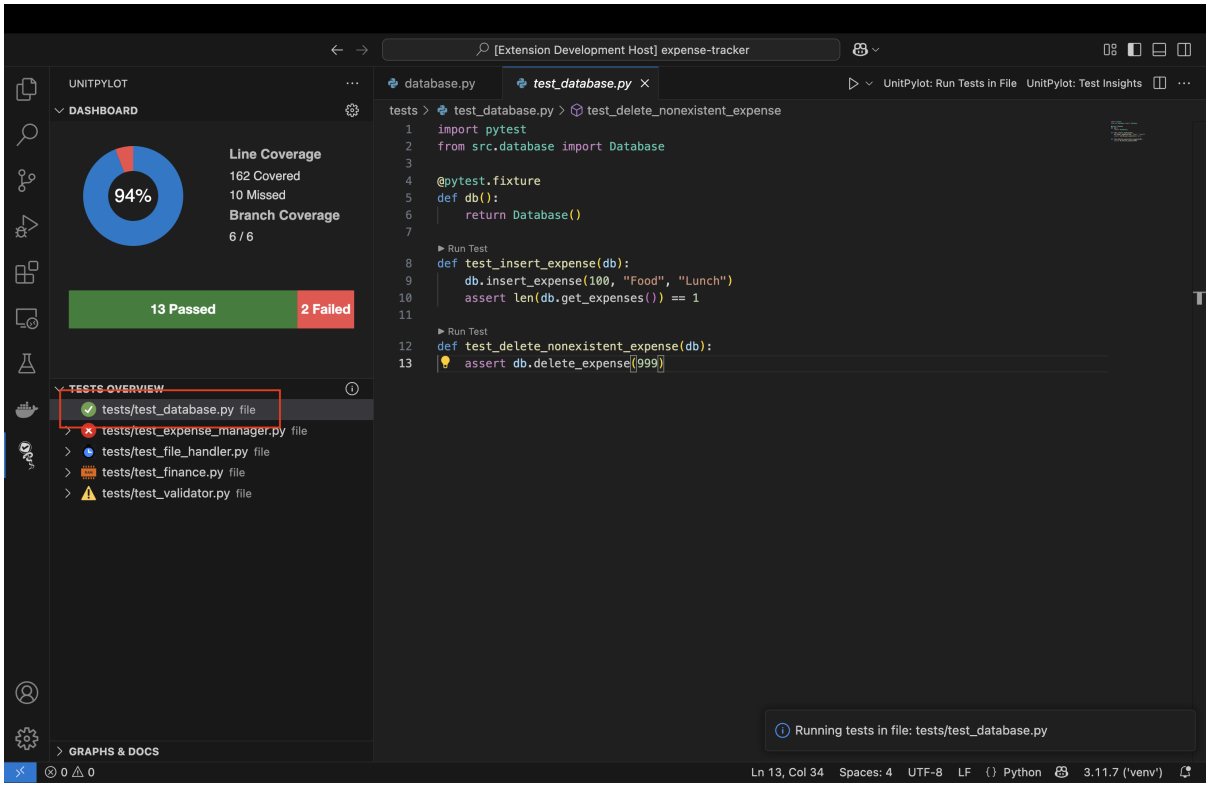
4. If accepted, the suggestion will be applied to the relevant file. View the file and refactor code if necessary. If rejected, the annotation disappears.



Accepting the suggestion



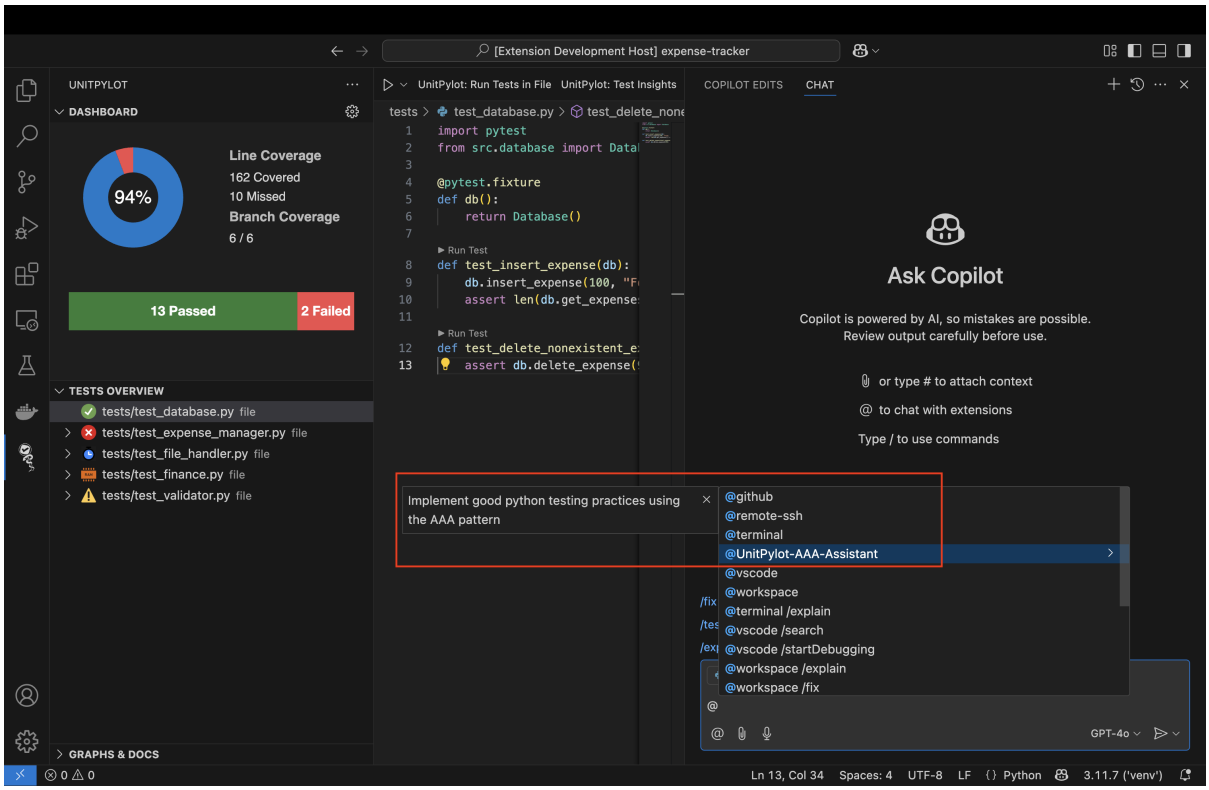
Refactor code as needed

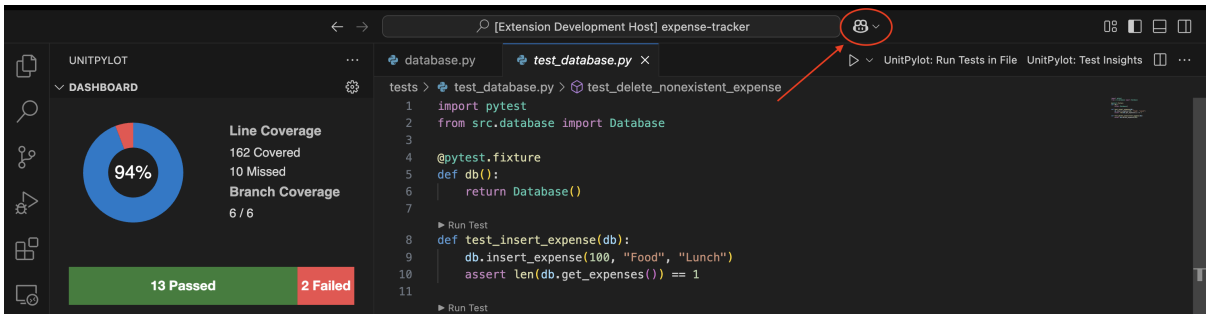


Updated Dashboard

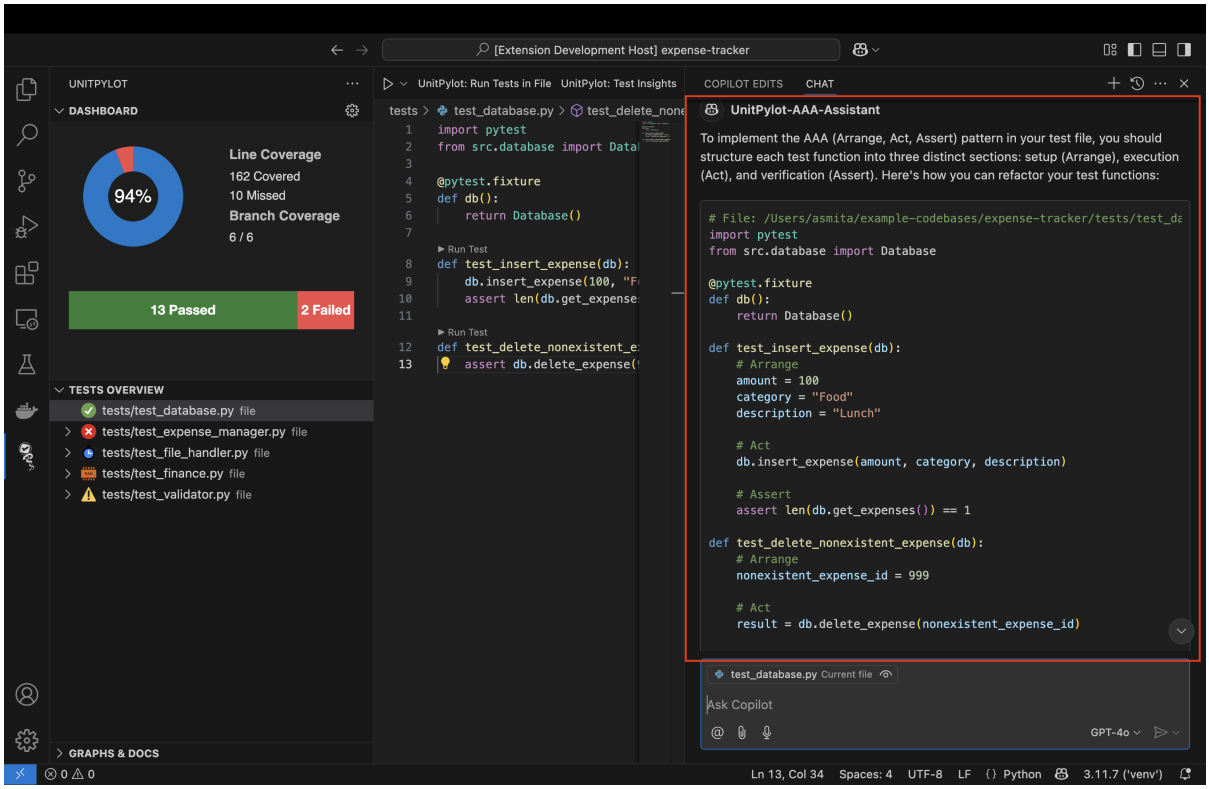
AAA Chat Participant

The Chat Participant can be accessed by clicking on the button as shown below. You much **search** for `@UnitPylot-AAA-Assistant` to use it.





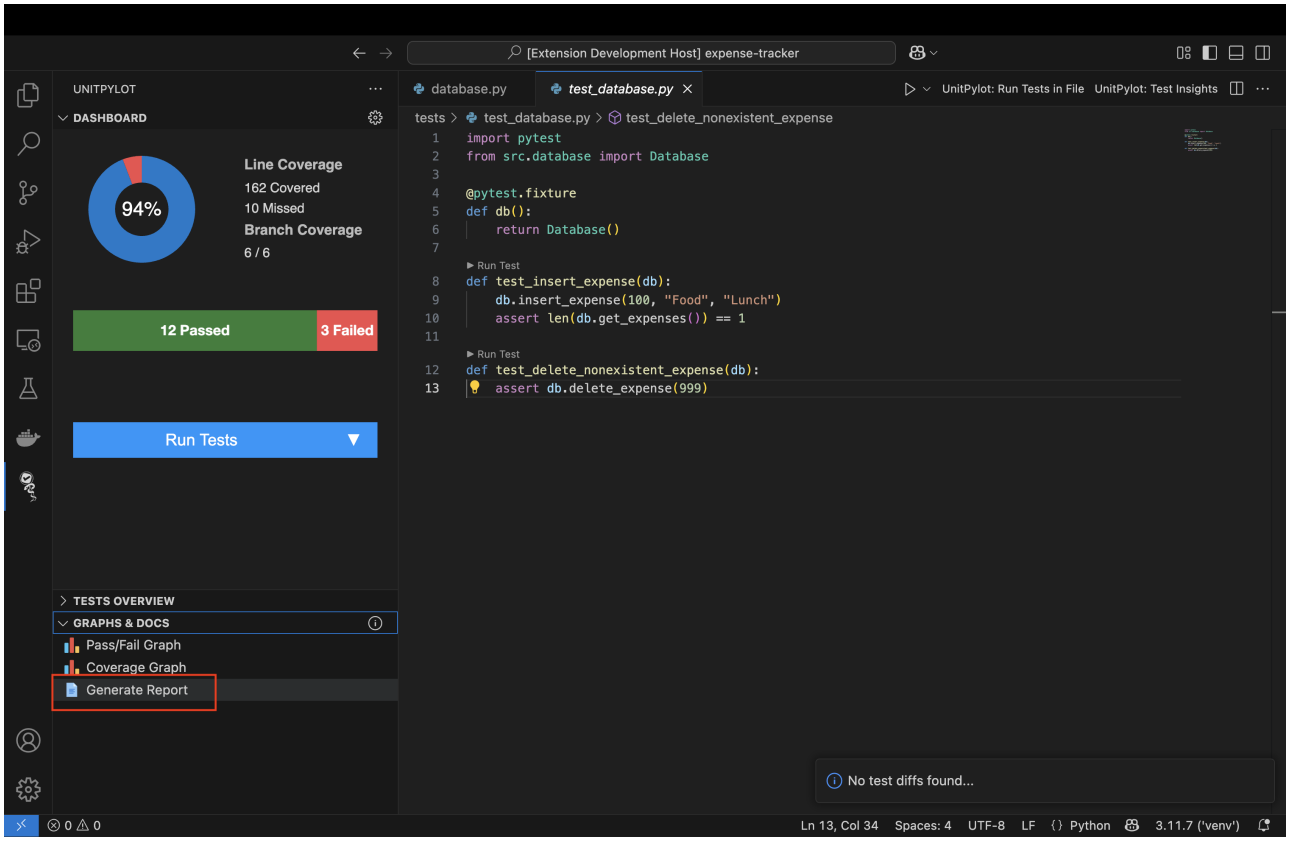
Interacting with the Participant



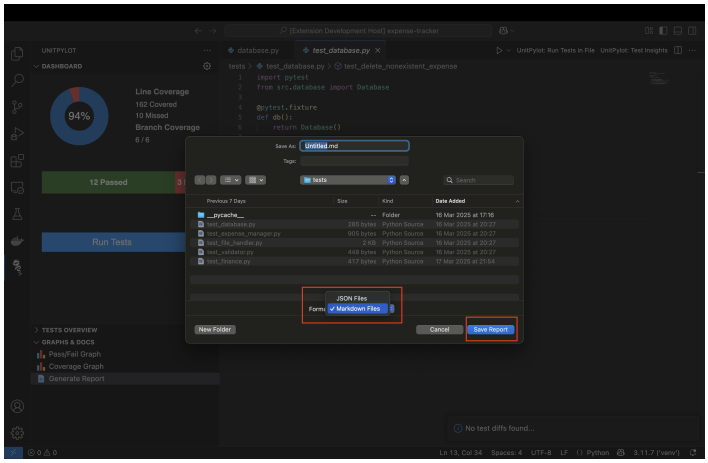
Example response

Generating a Report

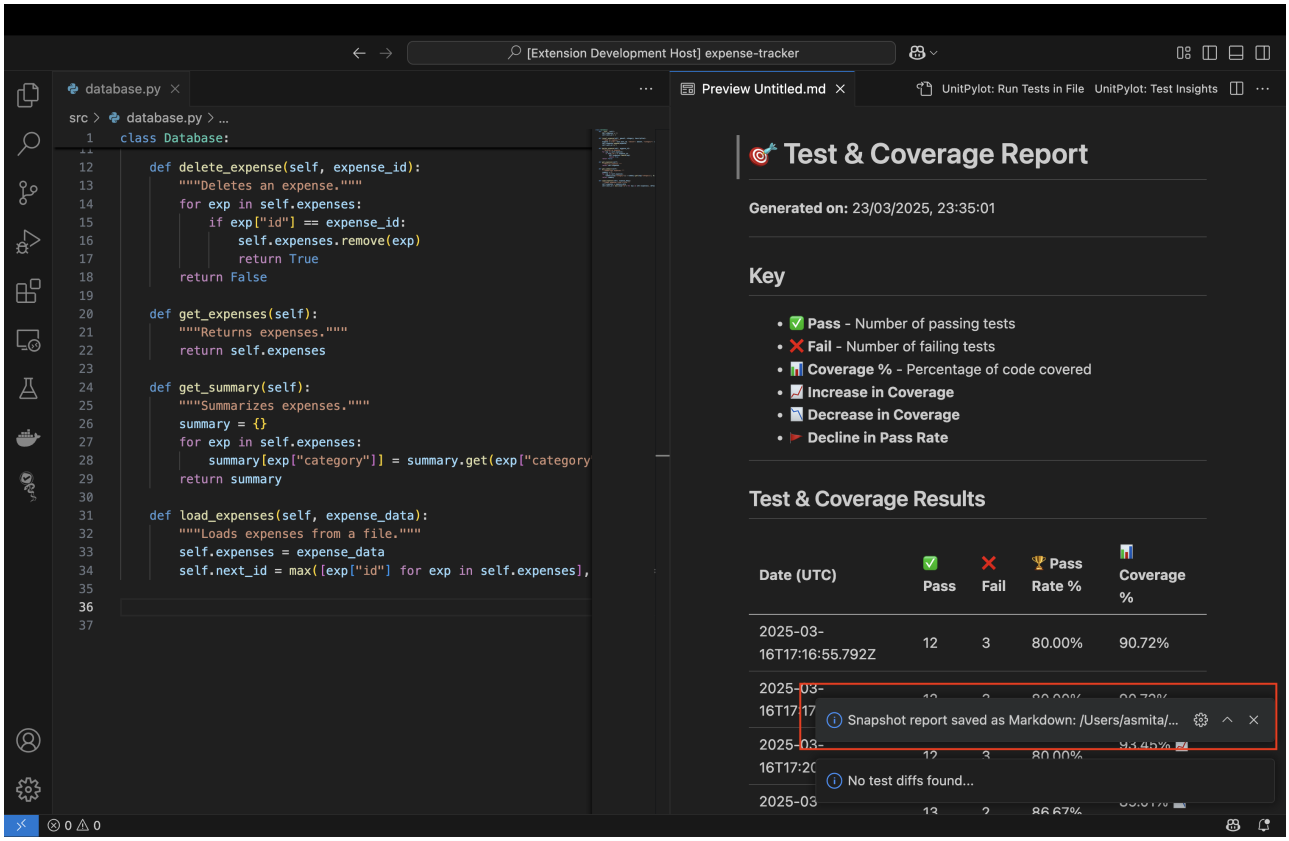
1. First click on the `Generate report` button.



2. Then specify the name, where you want to save the report and its format(`JSON` or `Markdown`).

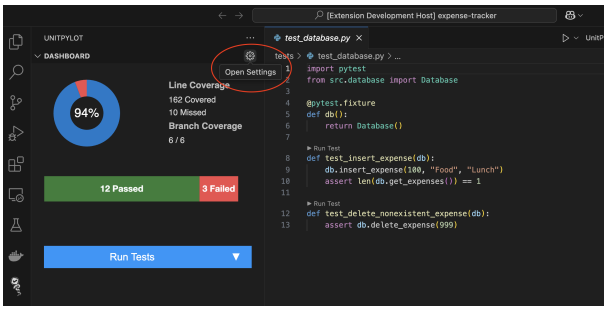


3. Upon success, a message will notify the report has been generated and saved!

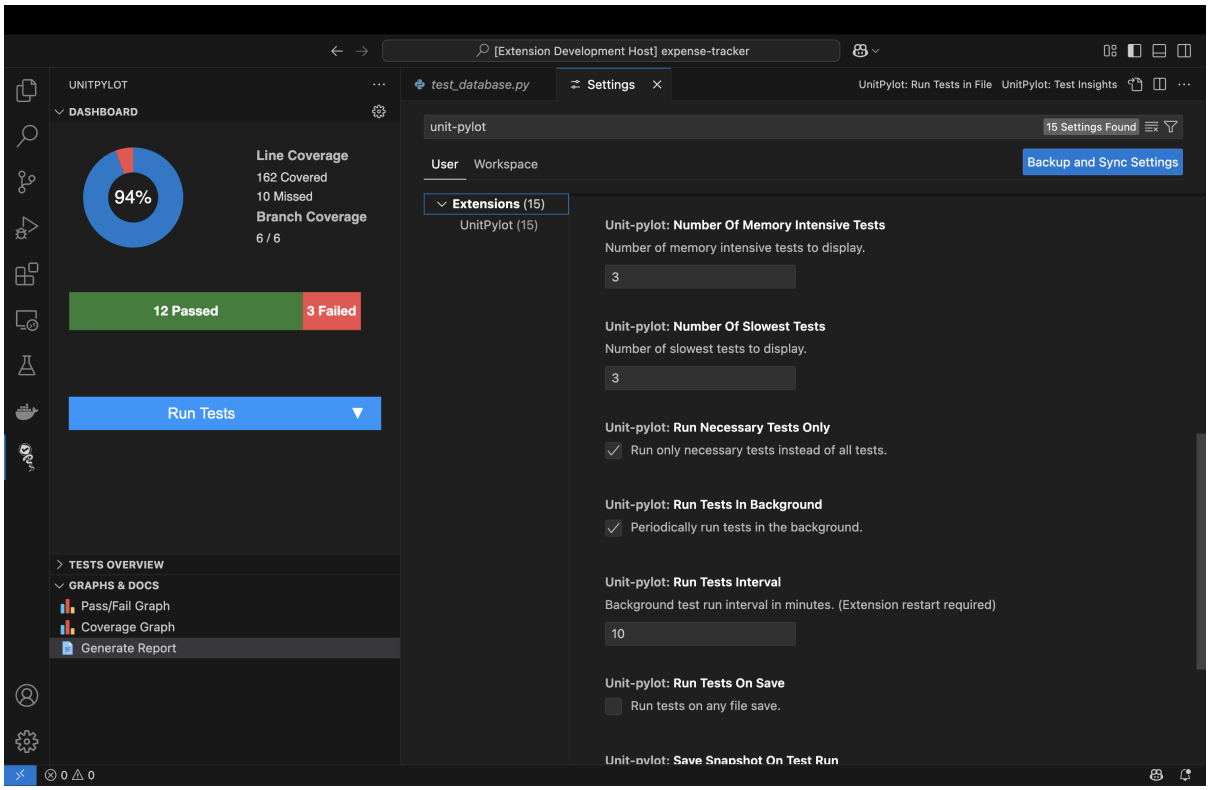


Settings Page

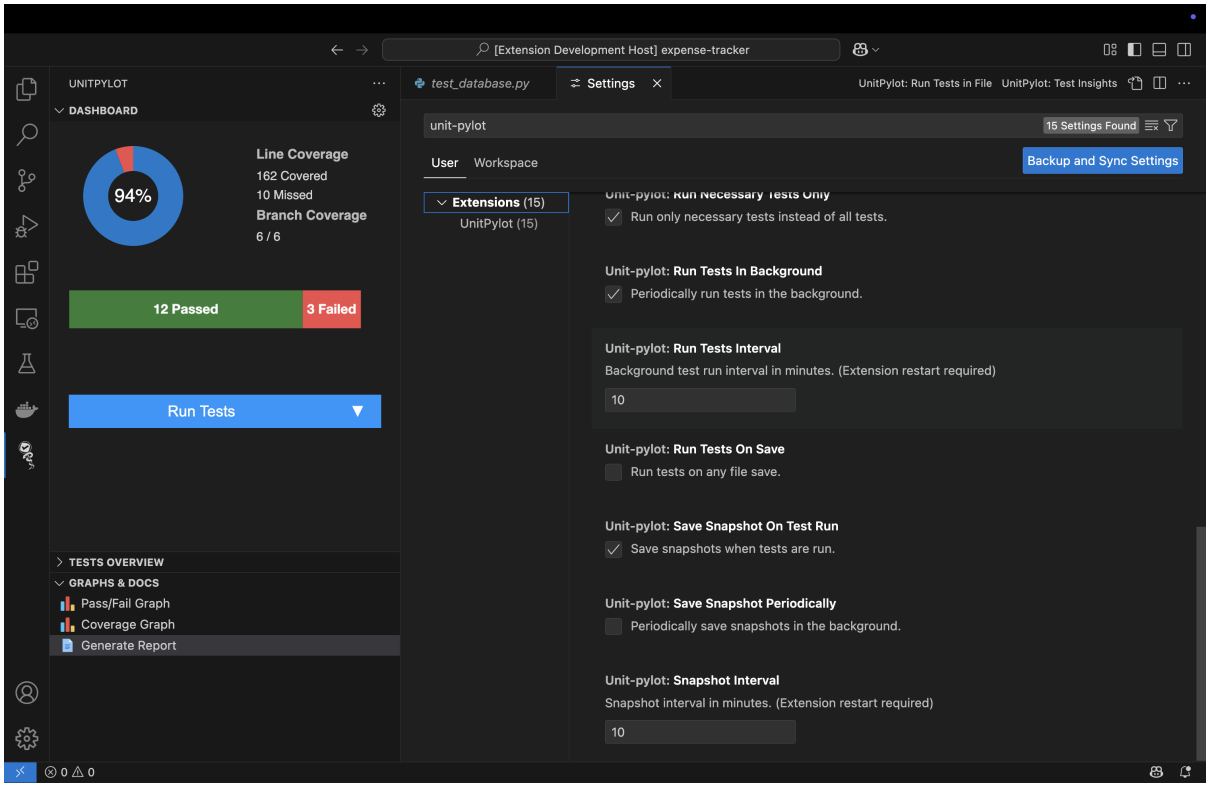
Clicking on the settings icon allows you to open the settings page.



Here you can change the number of slowest tests and memory intensive tests to be displayed on the Dashboard, along with how the dashboard metrics are saved. (i.e. every time you save the file, or every x amount of minutes.)



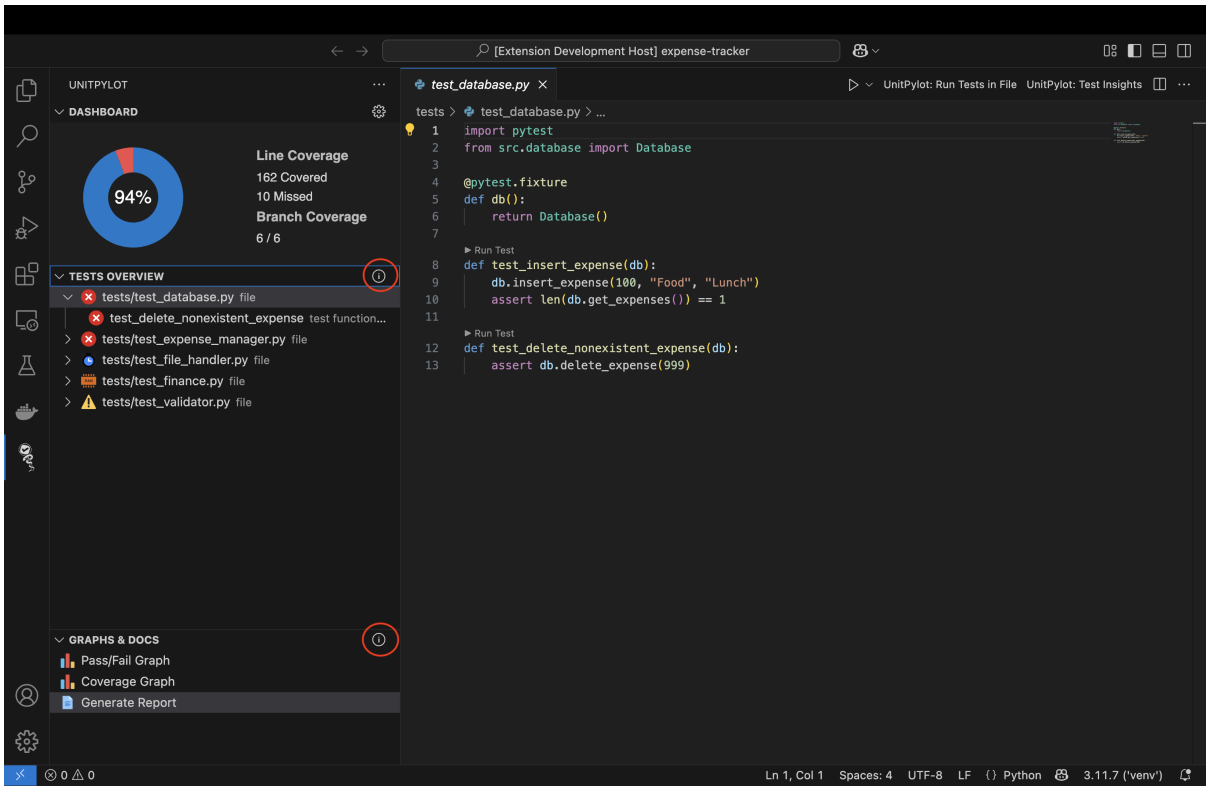
Settings page



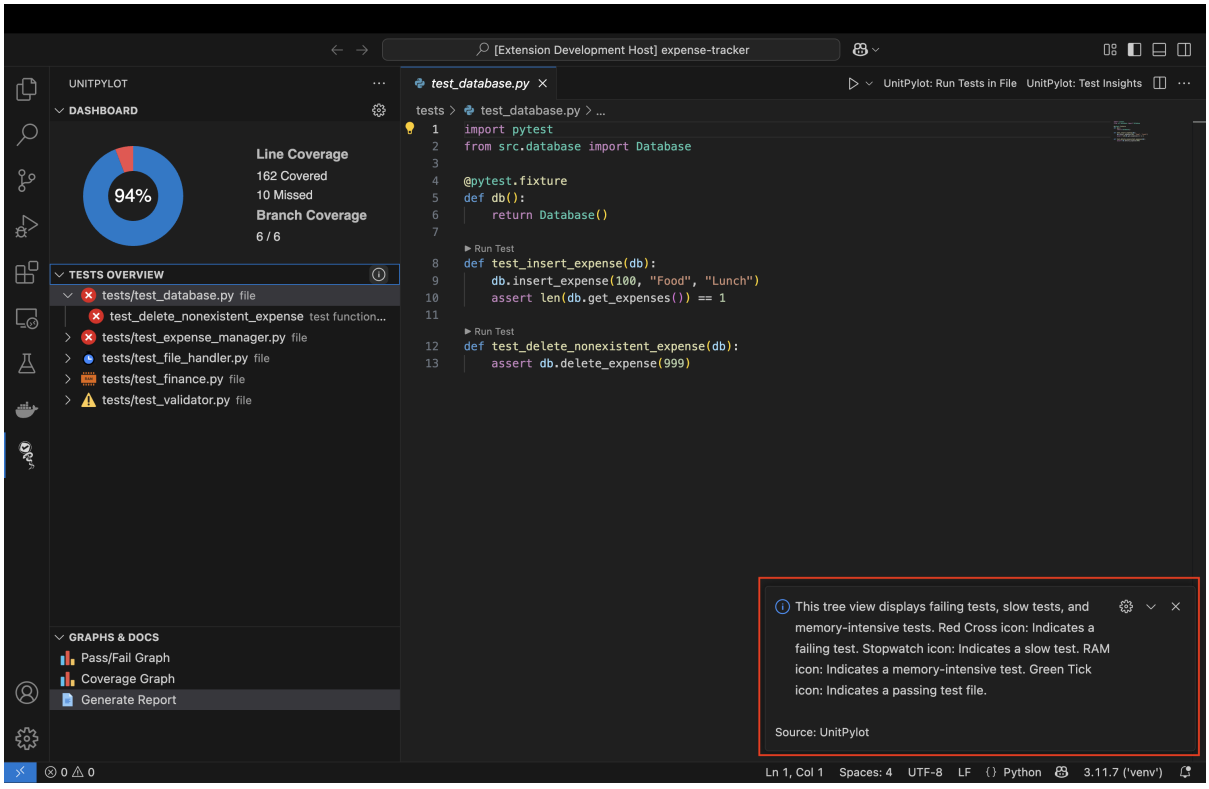
Settings page continued

Information Icons

Hover over the tooltips triggers for more information on each feature.



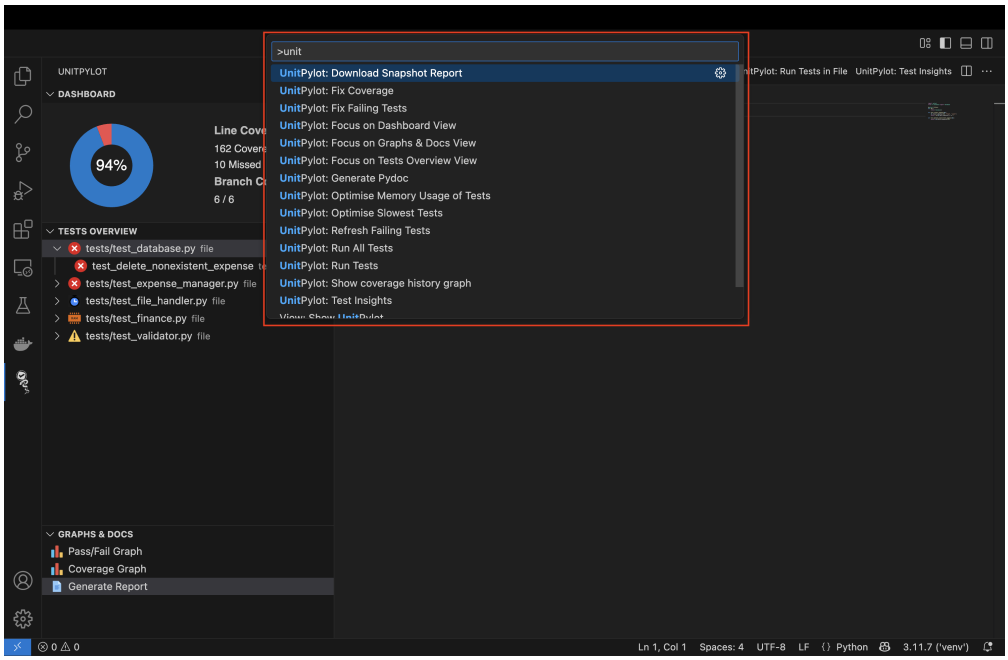
Click on a information tooltip



A pop-up will appear for key information about the feature

A Note on the Commands

All commands are also accessible from the **Command Palette**! Simply type `>UnitPylot` and they should all be listed!



Custom, Third-Party LLM Support

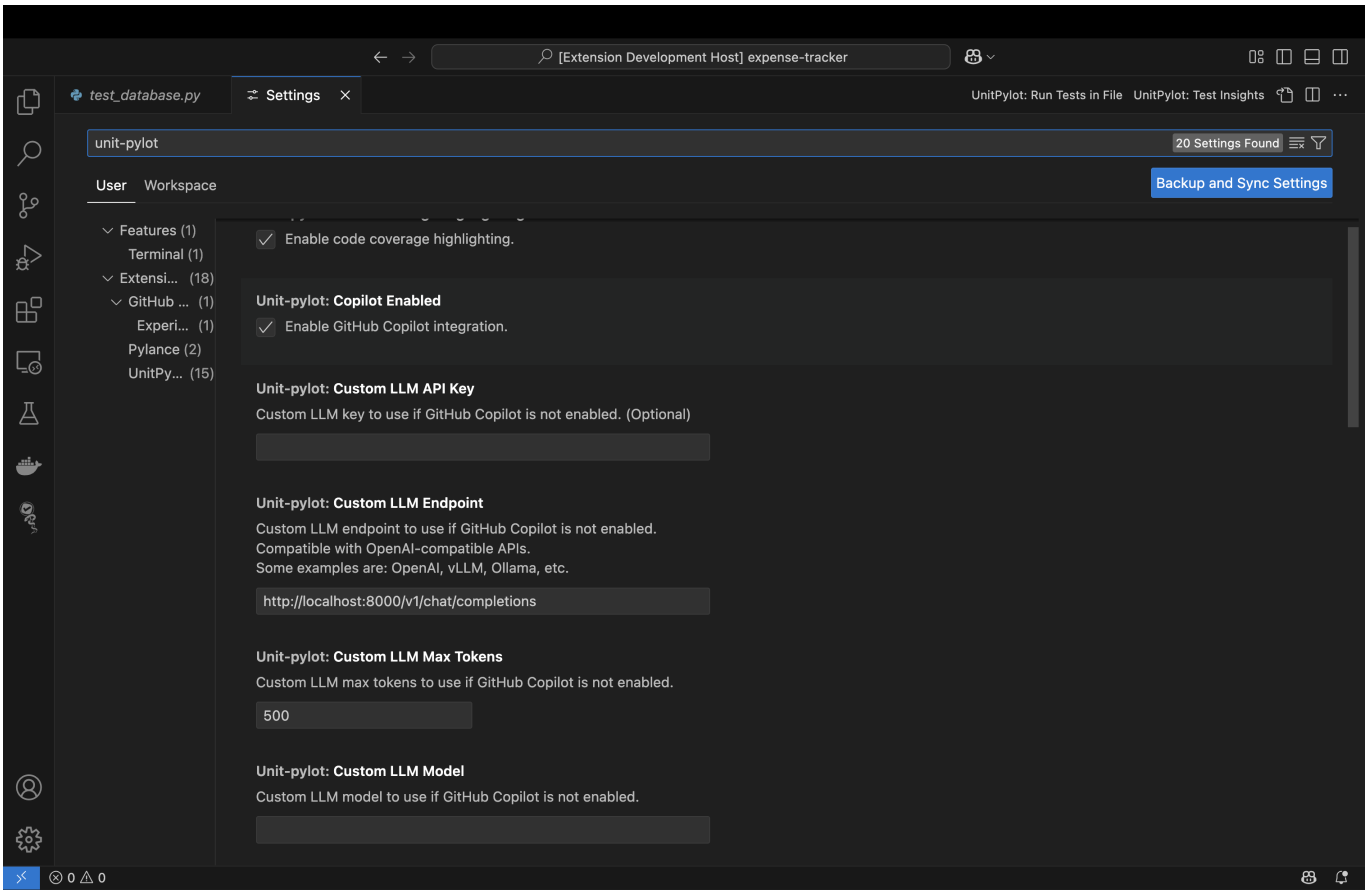
This extension **should** be used with GitHub Copilot, as this provides the best user experience for the developer. However there is the option to use a Third-party or Local LLM if the user wishes.

UnitPylot supports custom large language models (LLMs) through an OpenAI-style API. This allows you to integrate your own LLMs for test writing and optimisation.

To configure a custom LLM, set the following options in the VSCode settings:

- `unit-pylot.customLLM Endpoint`: The endpoint URL for your custom LLM API (e.g., `http://xxxxxx/v1/chat/completions`).
- `unit-pylot.customLLM Model`: The model name to use with your custom LLM.
- `unit-pylot.customLLM APIKey`: The API key for authenticating with your custom LLM.
- `unit-pylot.customLLM MaxTokens`: The maximum number of tokens to use for each request to your custom LLM.

Please note: The extension has not been designed with this in mind and we strongly discourage this mode of use as the accuracy of results for the optimisations will be impacted.

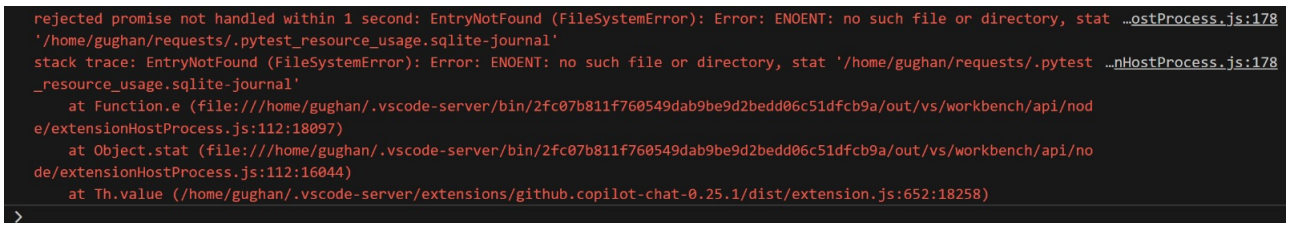


5. Troubleshooting

5.1 Common Issues

- If there is an issue in downloading the extension from the Visual Studio Code Marketplace, then you can also access it **locally**. Further instructions are included in the `README.md`.

- When building the extension from source and running via vscode, you may encounter error messages like these. This is because our extension deletes temporary files generated by `pytest` which may throw an error in other extensions.



  The example above shows an error message from the GitHub copilot chat extension but this can be ignored.

- Sometimes the `UnitPylot: Optimise Memory Usage of Tests` command may not optimise all test cases in the file, and only the most intensive one. This is because memory allocations fluctuate each time tests are run in the background. Please **re-run** the tests then **run the command again**.

6. Contact Information

For any questions, feedback, or support, feel free to reach out to the **UnitPylot team**:

- **Aaditya Kumar**: aaditya.kumar.23@ucl.ac.uk, aadityastyles@gmail.com
- **Asmita Anand**: asmita.anand.23@ucl.ac.uk, asmitaanand04@gmail.com
- **Gughan Ramakrishnan**: gughan.sowndravalli.23@ucl.ac.uk, rs.gughan@gmail.com
- **Swasti Jain**: swasti.jain.23@ucl.ac.uk, swasjn4@gmail.com