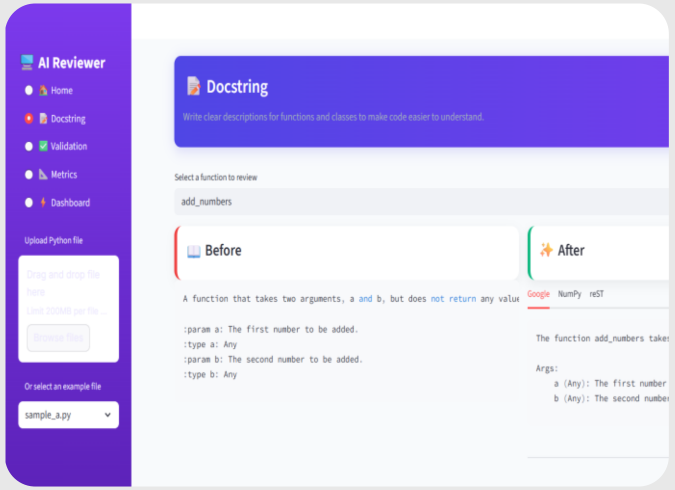
AI Code Reviewer

Smart Code Analysis & Documentation



Docstring

Write clear descriptions for functions and classes to make code easier to understand.

Select a function to review

add_numbers

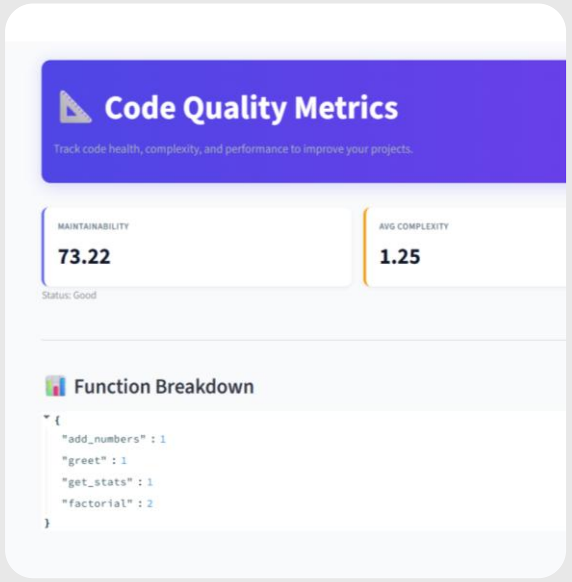
Before

```
A function that takes two arguments, a and b, but does not return any value

:param a: The first number to be added.
:type a: Any
:param b: The second number to be added.
:type b: Any
```

After

```
The function add_numbers takes

Args:
    a (Any): The first number
    b (Any): The second number
```

AI Code Reviewer



Code Quality Metrics

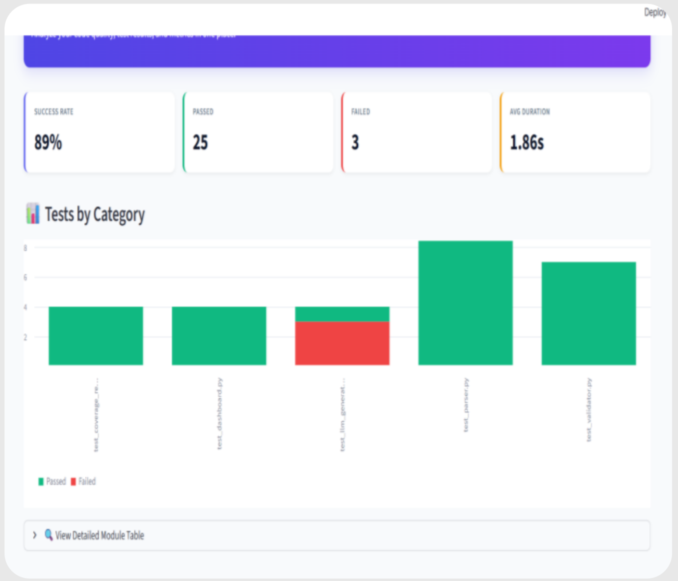
Track code health, complexity, and performance to improve your projects.

MAINTAINABILITY	AVG COMPLEXITY
73.22	1.25

Status: Good

Function Breakdown

```
{
  "add_numbers" : 1
  "greet" : 1
  "get_stats" : 1
  "factorial" : 2
}
```



SUCCESS RATE	PASSED	FAILED	AVG DURATION
89%	25	3	1.86s

Tests by Category

Presented By:

Suman Kumari

PROBLEM STATEMENT

- Many Python projects have poor code quality because of missing docstrings, high complexity, and no proper validation.
- and students often find it hard to manually check code for documentation rules, complexity, and maintainability.
- Existing tools are either complex, slow, or do not give clear visual feedback.

📘 Introduction

- ➢ The **AI Code Reviewer** is a web-based tool that helps developers automatically analyze Python code.
- ➢ It uses AI and static analysis to generate docstrings, check coding standards, measure complexity, and calculate maintainability.
- ➢ This tool makes code review faster, easier, and more accurate, especially for students and beginner developers.

Project Objectives

Improve Code Quality

Check Python code automatically to find missing docstrings, errors, and bad practices.
Help developers write clean and readable code.

Automate Code Review

Use AI to generate docstrings and analyze code complexity.
Reduce manual effort and save time.

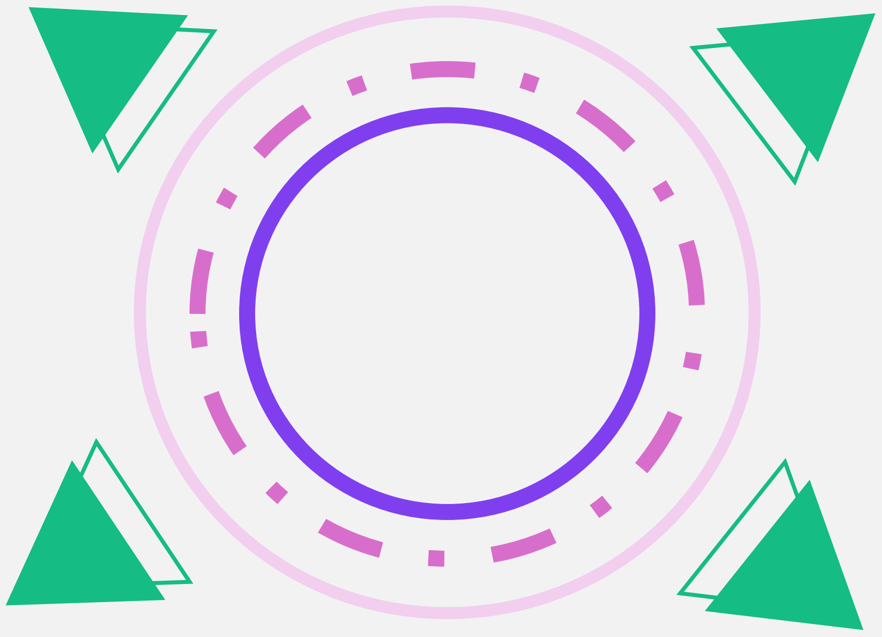
Clear Metrics & Insights

Show complexity, maintainability, and coverage in a simple dashboard.
Make results easy to understand for students and developers.

Scope of Project

Analyze Python code automatically

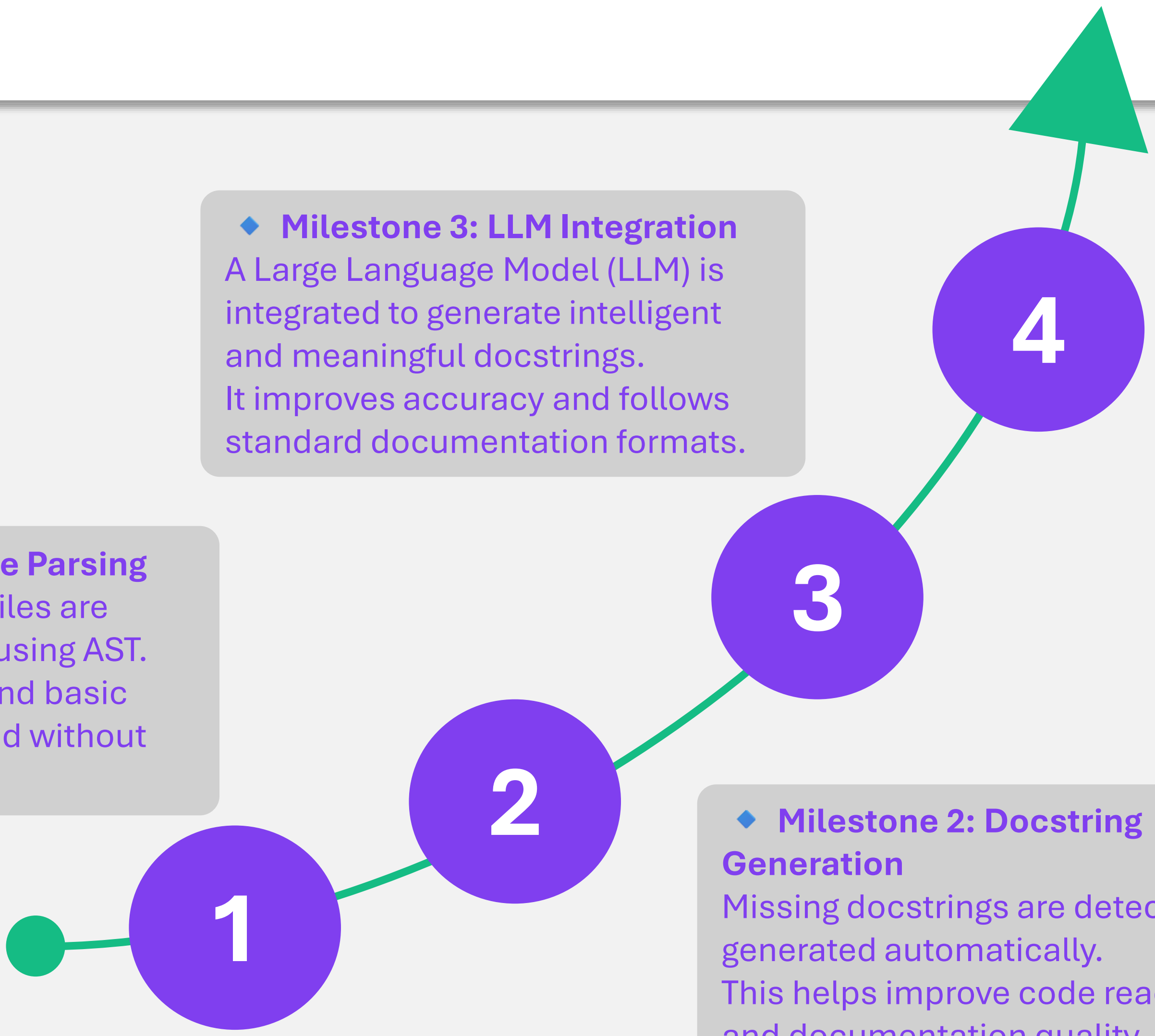
Generate and validate docstrings using AI



Measure code complexity and maintainability

Show clear results in a simple dashboard

Project Timeline



◆ **Milestone 1: Code Parsing**
In this stage, Python files are scanned and parsed using AST. Functions, classes, and basic structure are identified without errors.

◆ **Milestone 2: Docstring Generation**
Missing docstrings are detected and generated automatically.
This helps improve code readability and documentation quality.

◆ **Milestone 3: LLM Integration**
A Large Language Model (LLM) is integrated to generate intelligent and meaningful docstrings.
It improves accuracy and follows standard documentation formats.

◆ **Milestone 4: Dashboard Development**
A web dashboard is created to display metrics like complexity and maintainability.
Results are shown in a clear and user-friendly way.

Required Tools

Python
main programming language

LLM (GROQ Model)
For docstring generation

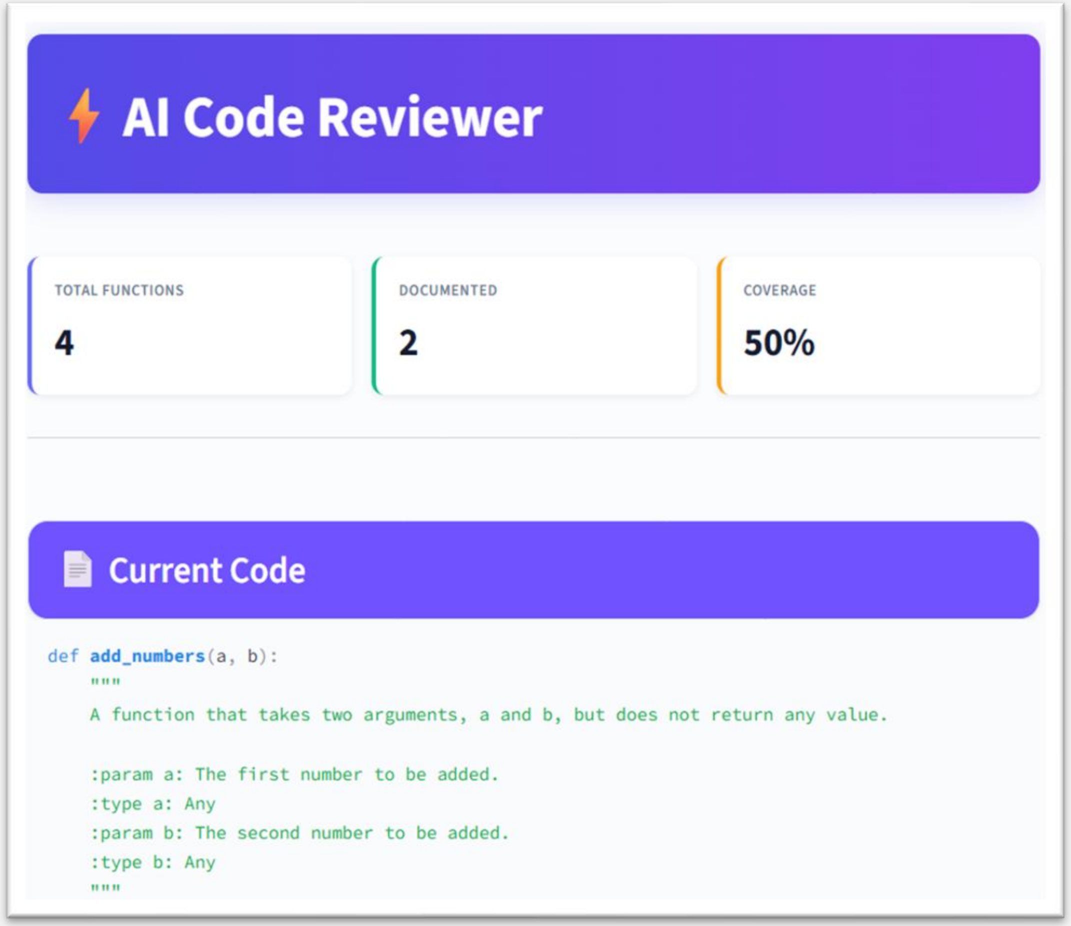
Streamlit
For dashboard and UI

PyTest
Used to test the Python code.

Home Section

This section displays the scanned file details, including the total number of functions, available docstrings, and overall documentation coverage.

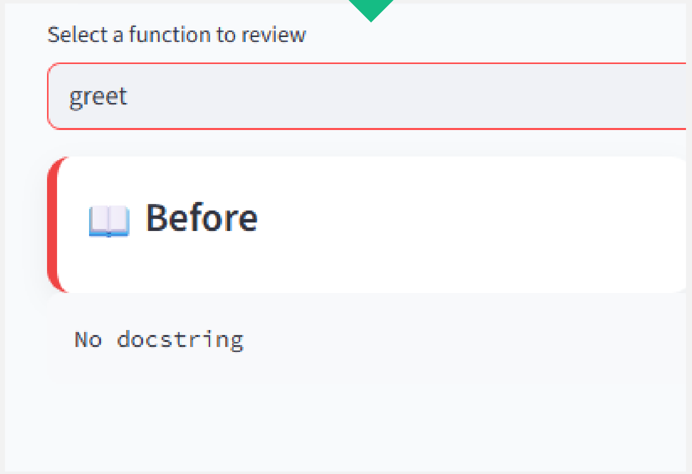
This section shows the scanned source code of the selected file.



```
def add_numbers(a, b):
    """
    A function that takes two arguments, a and b, but does not return any value.

    :param a: The first number to be added.
    :type a: Any
    :param b: The second number to be added.
    :type b: Any
    """
```

Docstring Section

Before, you can see that the function had no docstring earlier.



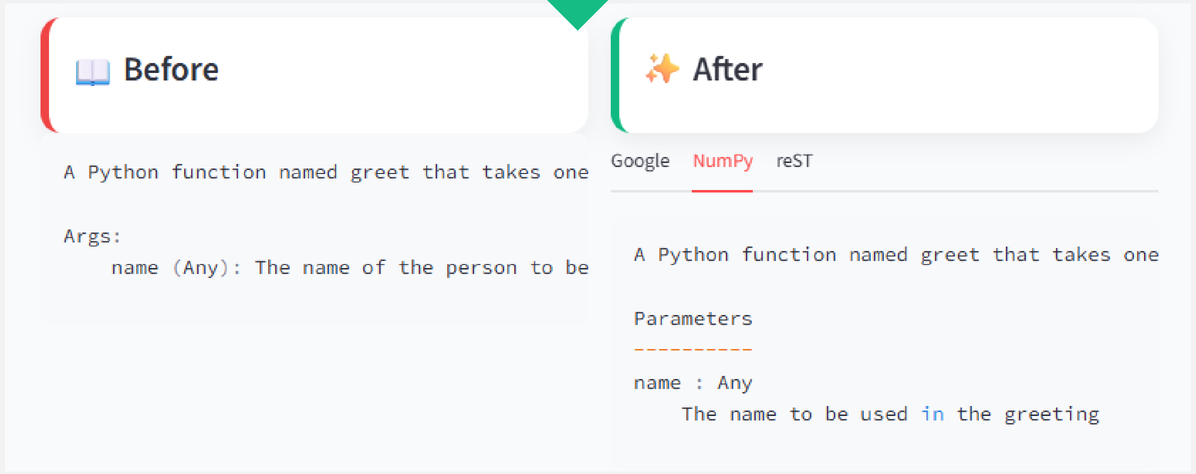
Select a function to review

greet

📖 Before

```
No docstring
```

After applying the selected style, the docstring is automatically generated.



📖 Before

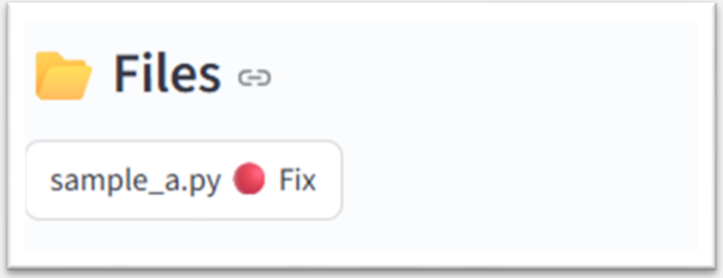
```
A Python function named greet that takes one

Args:
    name (Any): The name of the person to be
```

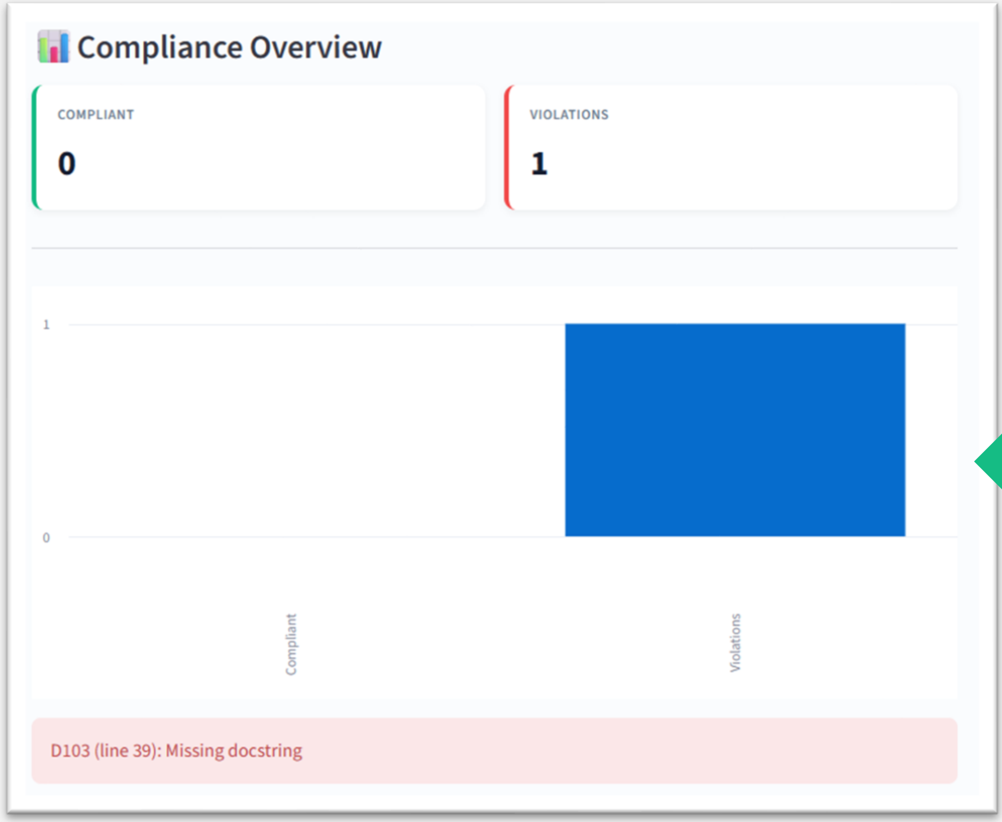
✨ After

```
A Python function named greet that takes one

Parameters
----------
name : Any
    The name to be used in the greeting
```

Validation Section

Before, you see this, onClick





D103 (line 39): Missing docstring

Now, it shows 0 complaints and 1 violation. ✅
complaint → singular works if you mean a single type or count.

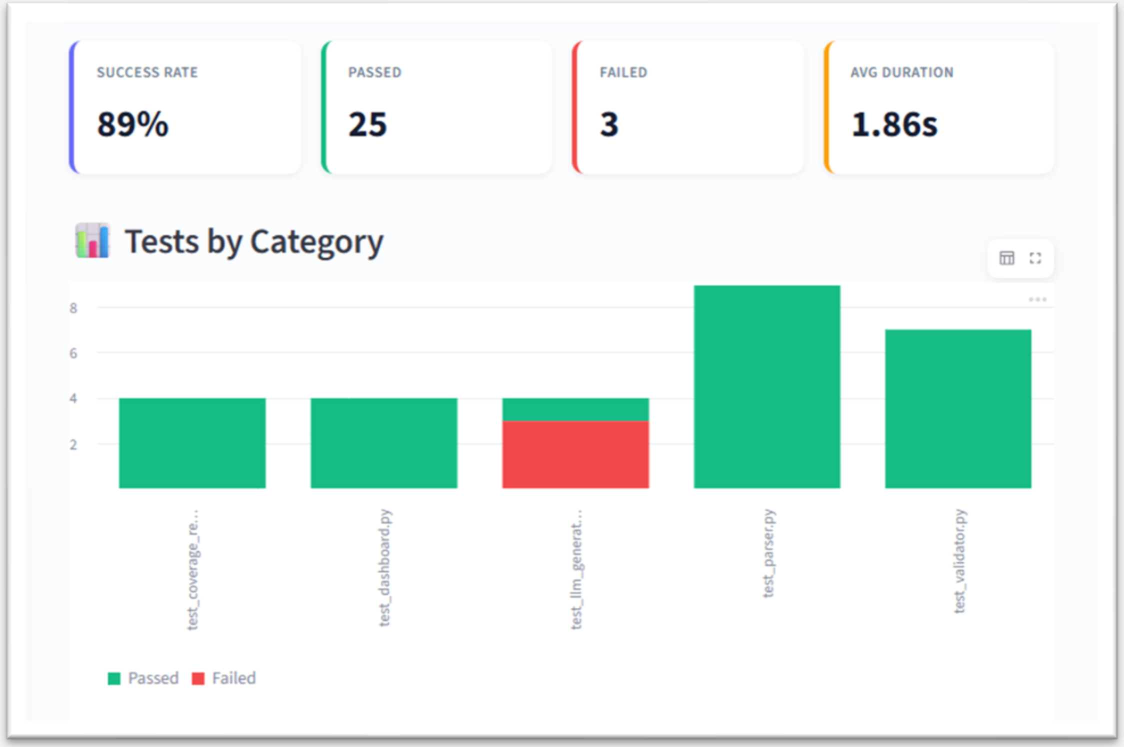
violations → keep plural if the count can be more than one, but since it's 1, you can also say **violation**.

Dashboard Section

The goal is to make sure the dashboard is reliable, user-friendly, and provides correct insights.

Test Summary: 89% success rate (25 passed, 3 failed), avg duration 1.86s.

Overall, the dashboard is mostly functional, with a few issues to address.

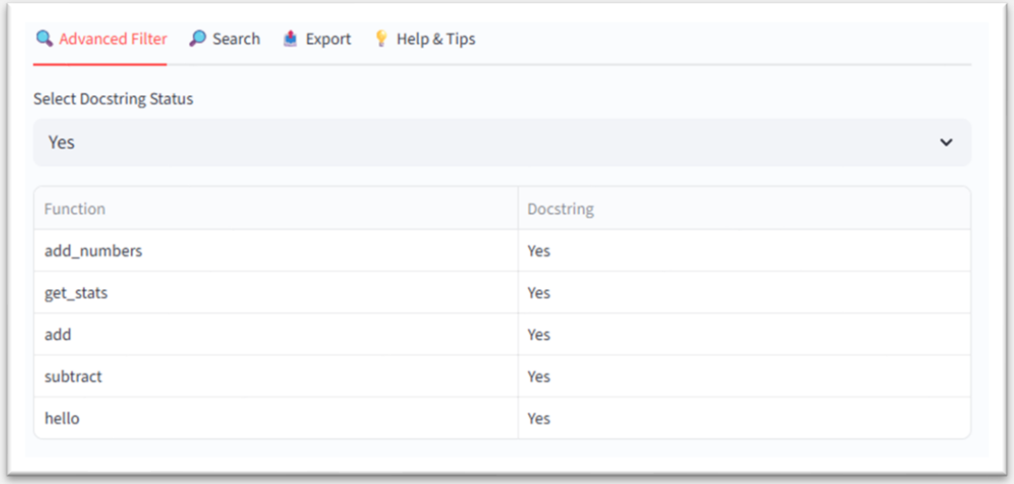


SUCCESS RATE	PASSED	FAILED	AVG DURATION
89%	25	3	1.86s

Tests by Category

Dashboard Section

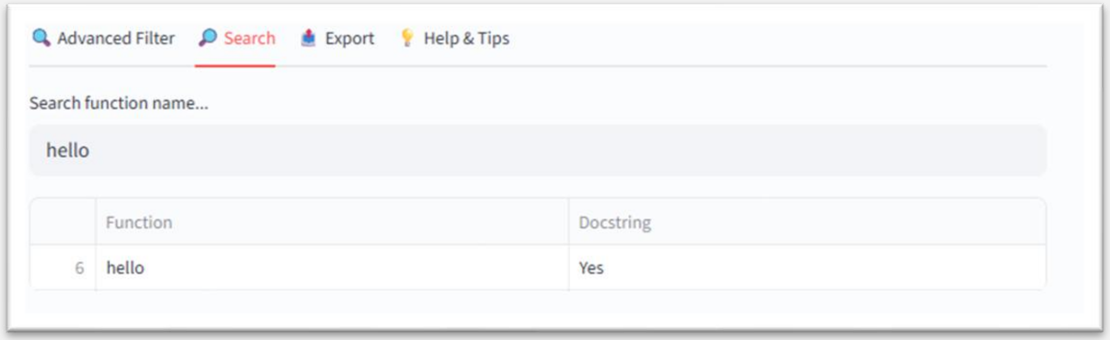
🔍 **Advanced Filter:** Allows users to filter data based on specific conditions.



Select Docstring Status

Yes

Function	Docstring
add_numbers	Yes
get_stats	Yes
add	Yes
subtract	Yes
hello	Yes



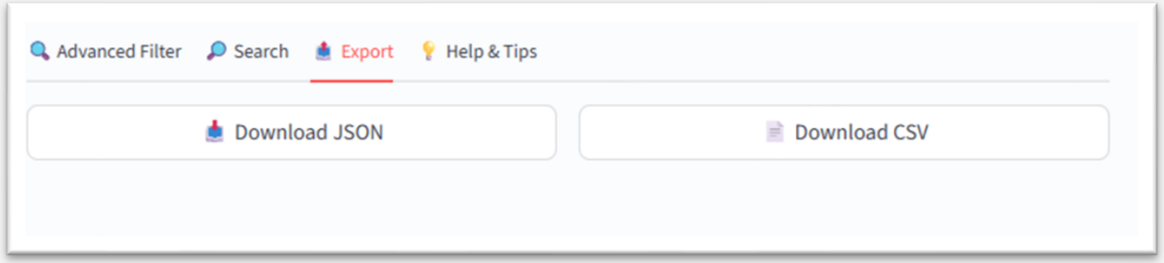
Search function name...

hello

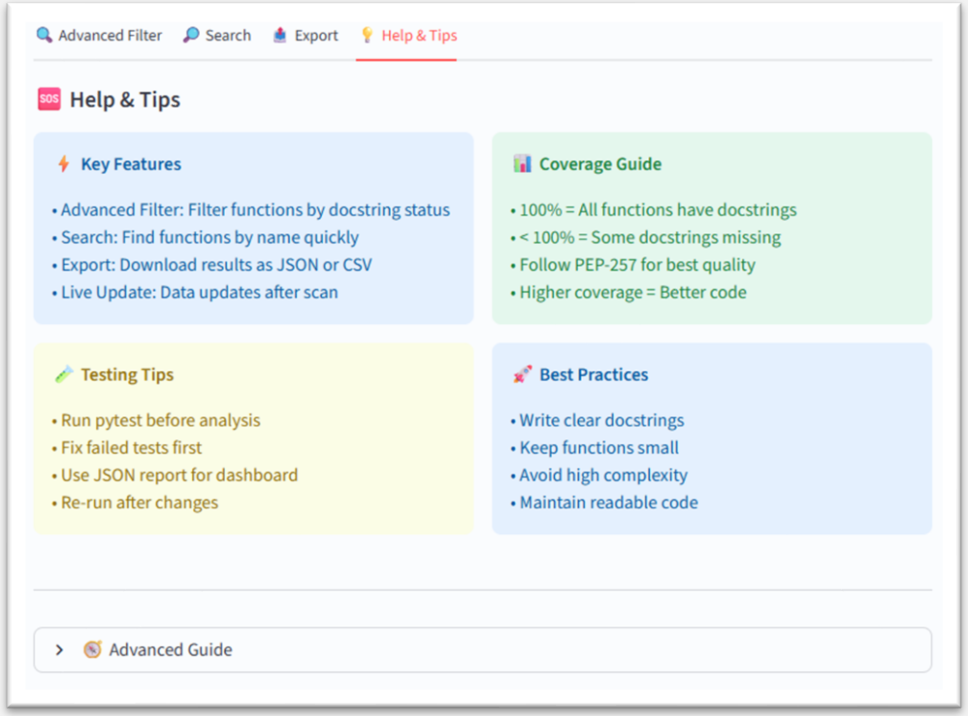
	Function	Docstring
6	hello	Yes

🔎 **Search:** Quickly find data or entries in the dashboard.

Dashboard Section



Export: Download or save data for offline use.



Help & Tips

Key Features

- Advanced Filter: Filter functions by docstring status
- Search: Find functions by name quickly
- Export: Download results as JSON or CSV
- Live Update: Data updates after scan

Coverage Guide

- 100% = All functions have docstrings
- < 100% = Some docstrings missing
- Follow PEP-257 for best quality
- Higher coverage = Better code

Testing Tips

- Run pytest before analysis
- Fix failed tests first
- Use JSON report for dashboard
- Re-run after changes

Best Practices

- Write clear docstrings
- Keep functions small
- Avoid high complexity
- Maintain readable code

💡 **Help & Tips:** Provides guidance and tips for using the dashboard effectively.

Conclusion

- The AI-powered code reviewer effectively analyzes code for maintainability, complexity, and risk.
- It provides detailed insights into functions, metrics, and potential issues, helping developers improve code quality quickly.
- Dashboard features like filtering, search, and export make it easy to use, and test results show it is reliable with minor improvements possible.