# **Lab 9 – Documentation Generation:**

# **Automatic Documentation and Code Comments**

**Enrollment Number:2503A51L36**

**Name: Suhana Rehan**

**Assignment Number:9.4**

**Lab Objectives**

* Inline comments
* Docstrings
* Auto-documentation tools
* AI-assisted summarization

**Task 1: Automatic Code Commenting**

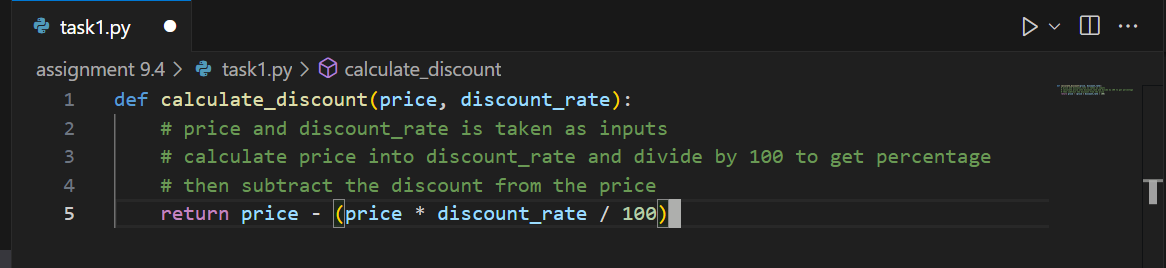
**Scenario:** You have been given a Python function without comments.

def calculate\_discount(price, discount\_rate):

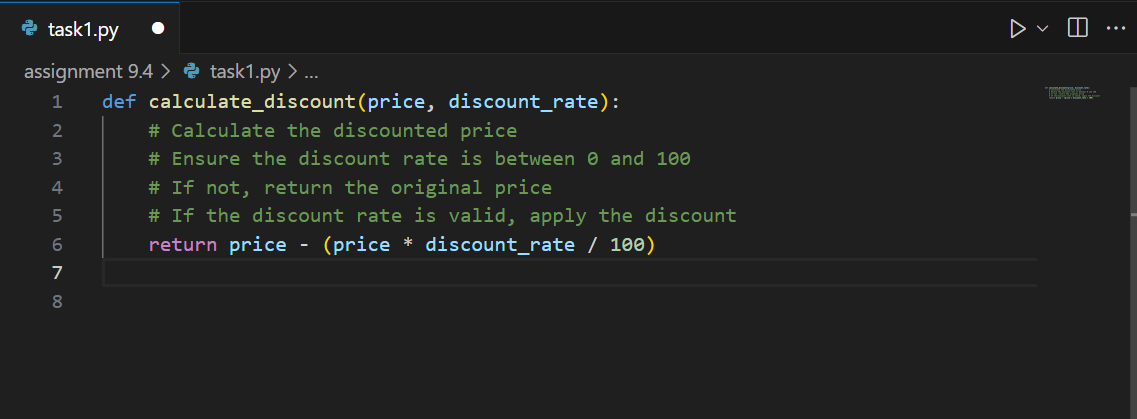
return price - (price \* discount\_rate / 100)

* Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.
* Modify the function so that it includes a docstring in Google-style or NumPy-style format.

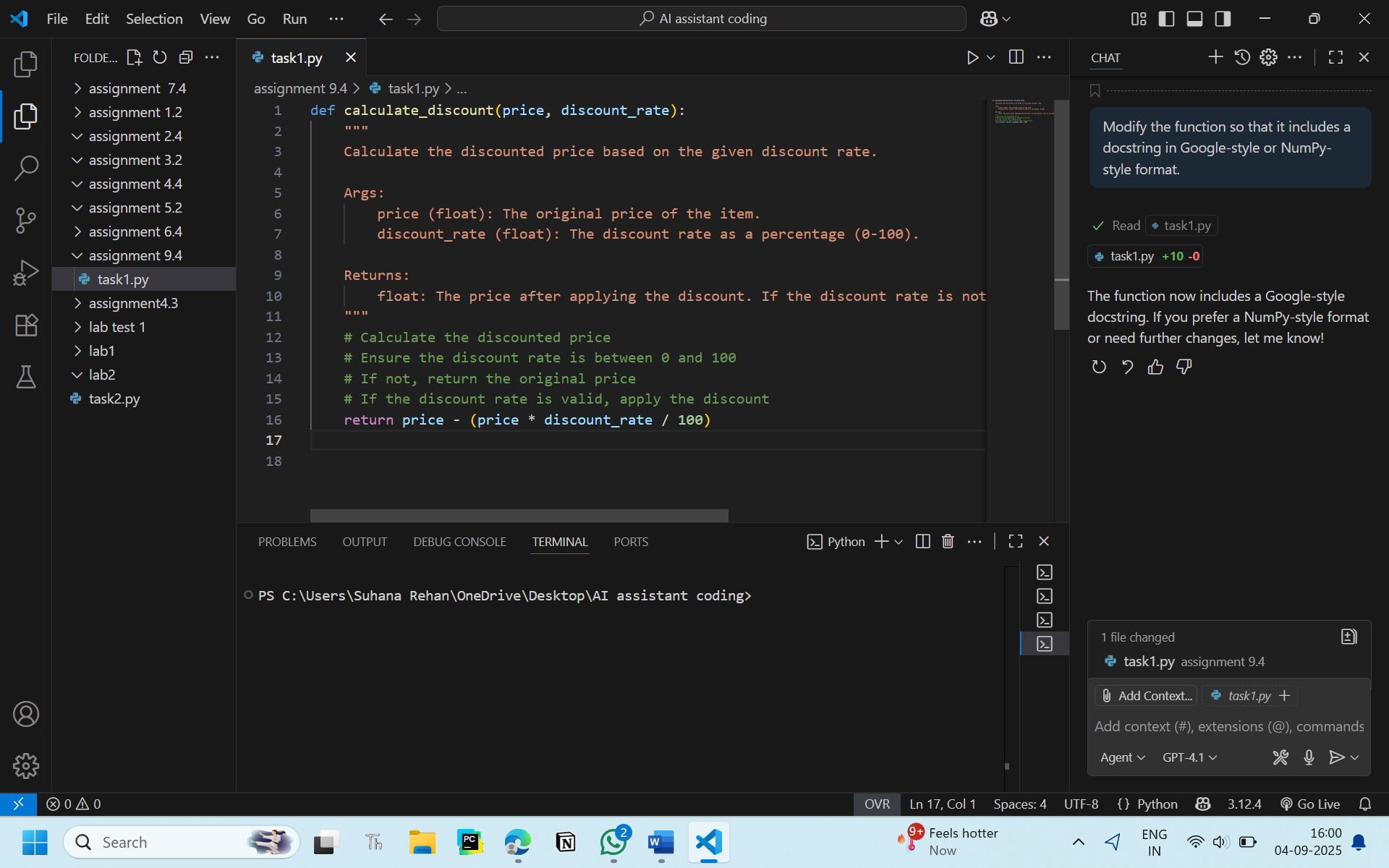
**Comment written by me:**

****

**Comment generated by AI:**

****

**Modified function and docstring generated by AI:**



**Observation:**

When I wrote comments myself, it took more time but they looked clear.  
AI comments were fast but not really on the main point.  
The docstring made the function look more proper and neat.  
So I feel manual comments are better, but AI can give a quick start.

**Task 2:** **API Documentation Generator**

**Scenario:** A team is building a **Library Management System** with multiple functions.

def add\_book(title, author, year):

# code to add book

pass

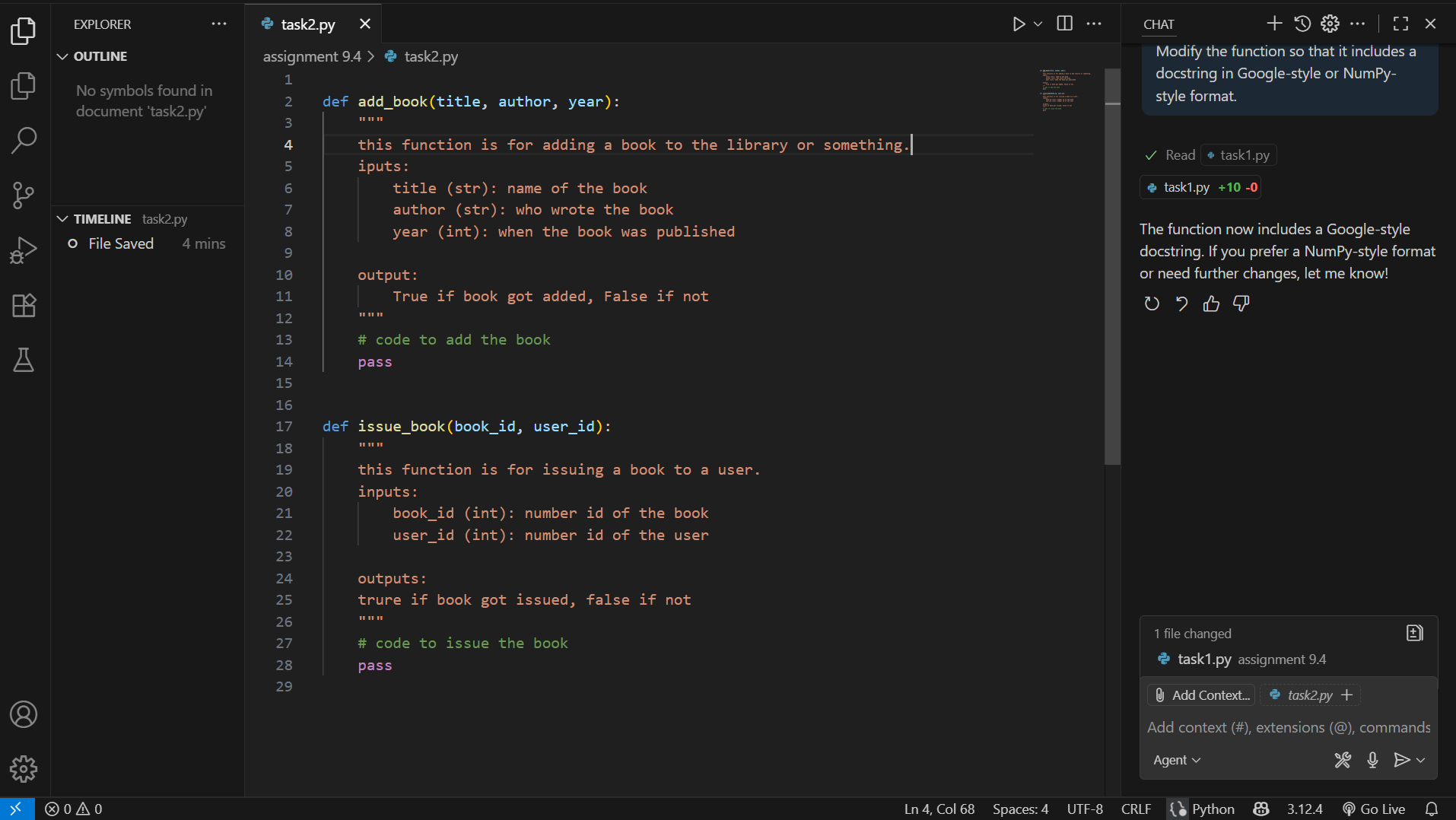
def issue\_book(book\_id, user\_id):

# code to issue book

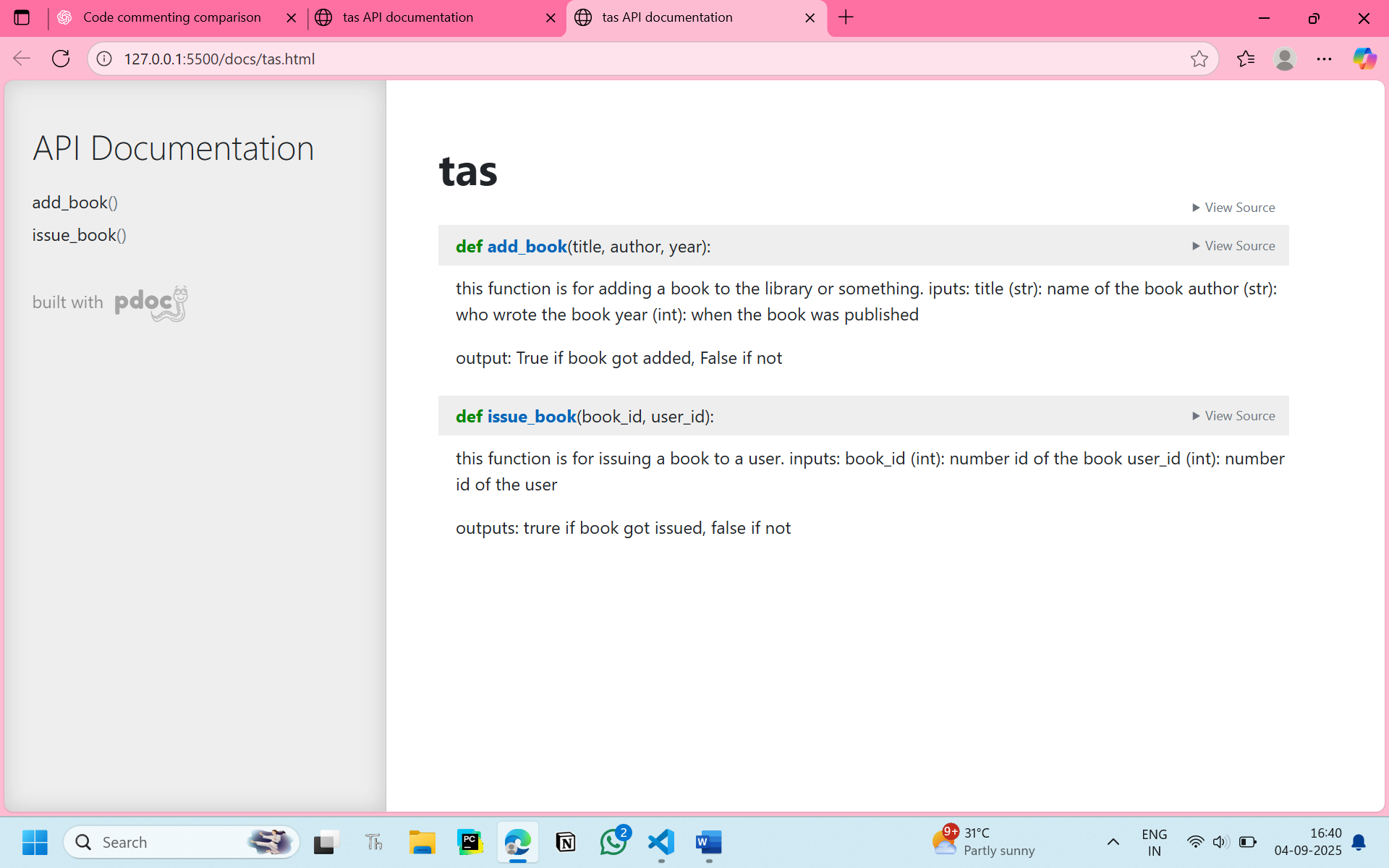
Pass

* Write a Python script that uses docstrings for each function (with input, output, and description).
* Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.

**Script written by me**

****

**Documentation generator tool (pdoc)**



**Task 3 : AI-Assisted Code Summarization**

**Scenario**: You are reviewing a colleague’s codebase containing long functions.

def process\_sensor\_data(data):

cleaned = [x for x in data if x is not None]

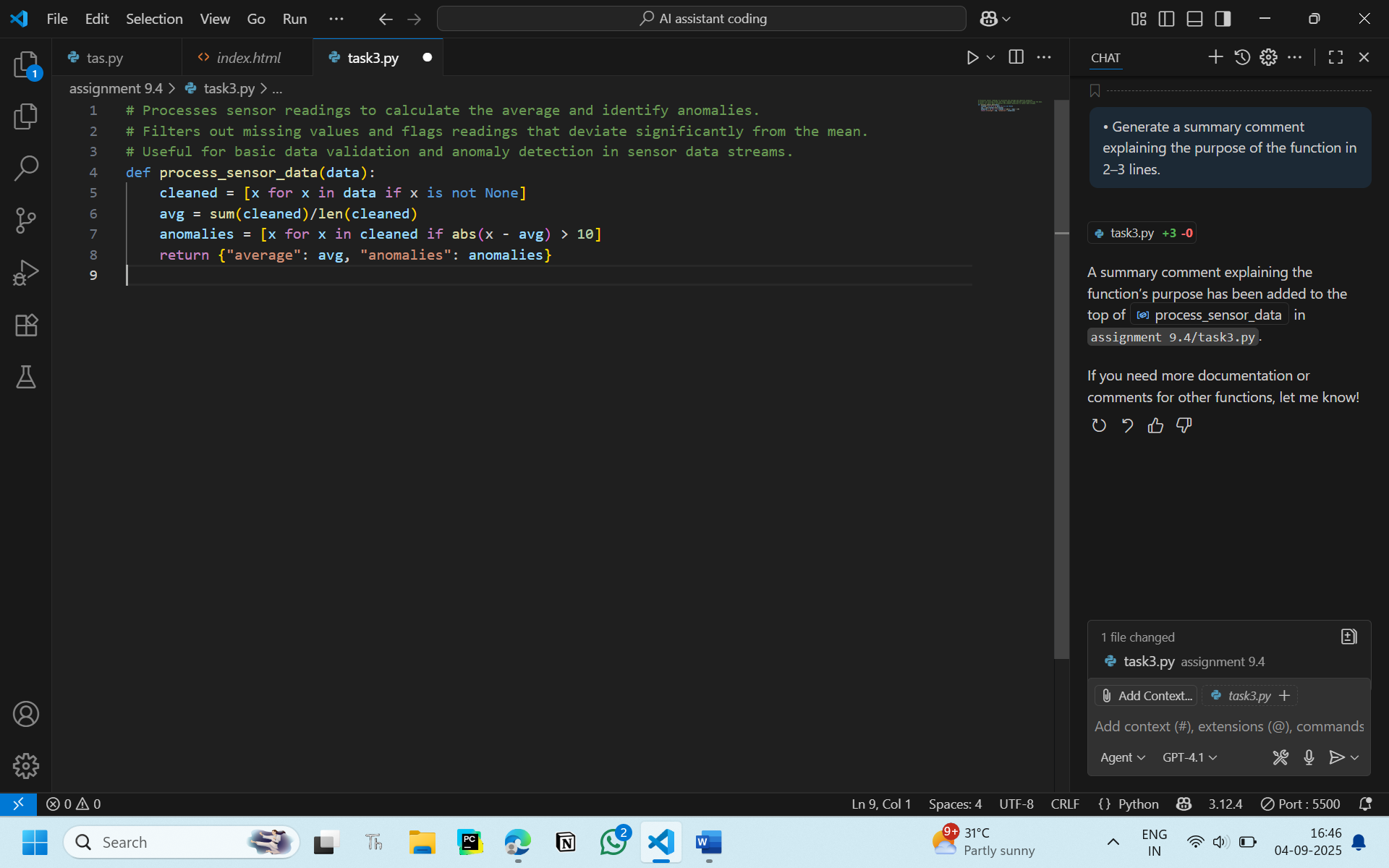
avg = sum(cleaned)/len(cleaned)

anomalies = [x for x in cleaned if abs(x - avg) > 10]

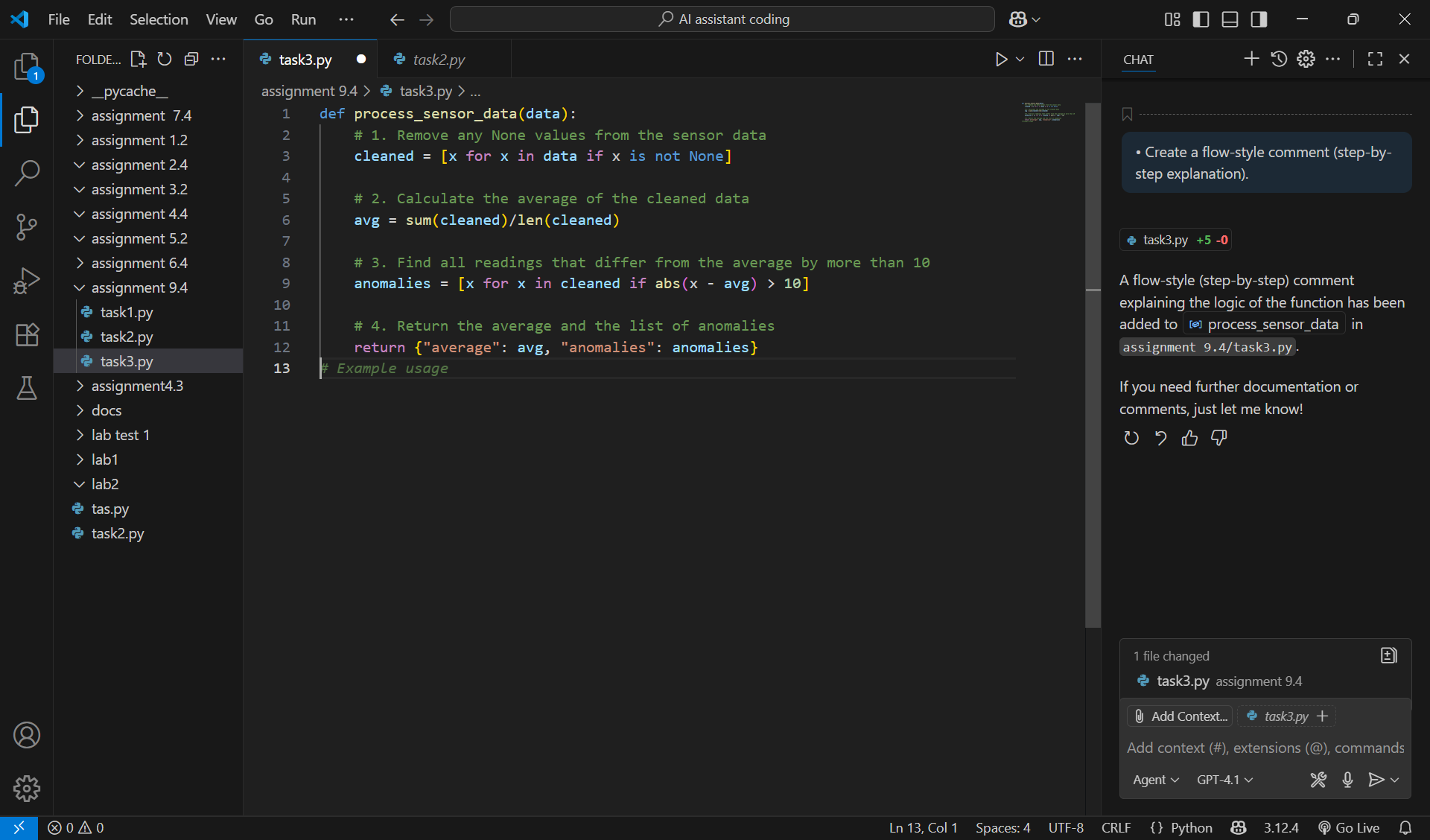
return {"average": avg, "anomalies": anomalies}

* Generate a summary comment explaining the purpose of the function in 2–3 lines.
* Create a flow-style comment (step-by-step explanation).
* Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios.

Summary comment explaining the purpose of the function in 2–3 lines:



AI Created a flow-style comment (step-by-step explanation):



Documentation:

This function can be useful in real-world applications where sensor data is collected continuously, such as weather monitoring, industrial machines, IoT devices, or healthcare equipment. By filtering out missing values, calculating the average reading, and identifying unusual spikes or drops, it helps detect faulty sensors, abnormal conditions, or sudden changes in the environment.

**Observation:**

The function demonstrates AI data preprocessing by cleaning data and finding anomalies

**Task 4:** Real-Time Project Documentation

**Scenario:** You are part of a project team that develops a Chatbot Application. The team needs documentation for maintainability.

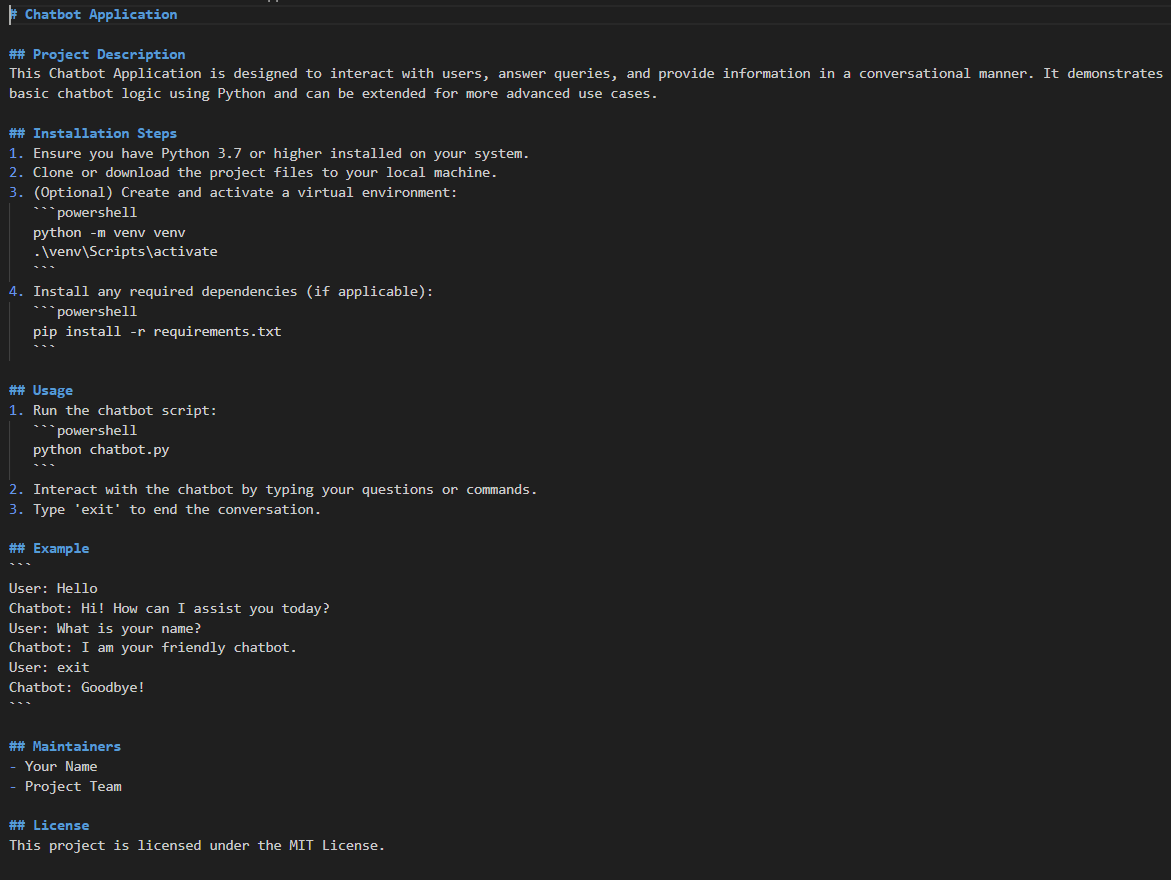
* Write a README.md file for the chatbot project (include project description, installation steps, usage, and example).
* Add inline comments in the chatbot’s main Python script (focus on explaining logic, not trivial code).
* Use an AI-assisted tool (or simulate it) to generate a usage guide in plain English from your code comments.

**Prompt:**

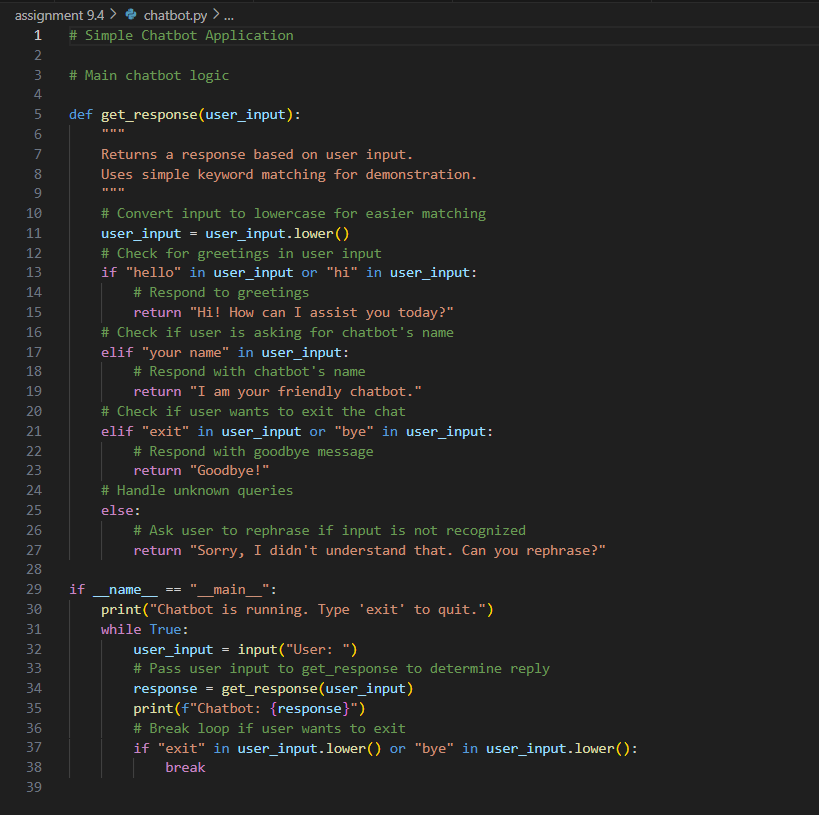
|  |
| --- |
|  |

|  |
| --- |
| Create documentation for a chatbot project: write a README.md with description, installation, usage, and example interaction; add inline comments explaining logic in the chatbot’s main Python script; simulate an AI-assisted tool to generate a plain-English usage guide from code comments. |

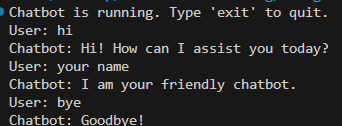
**Documents generated ( README )**



**Code:**



**Output:**



**Observation:**

The program implements a simple chatbot using Python functions and loops.

    2. It uses keyword matching to respond to greetings, name queries, and exit commands.

    3. The chatbot provides a default response for unrecognized inputs, prompting the user to rephrase.

    4. The main loop continues until the user types 'exit' or 'bye', demonstrating interactive conversation handling.

    5. The code is modular and easy to extend for more advanced chatbot features.