# ASSIGNMENT-9.5

NAME: T.Sai TAnuj

BATCH:15

ROLLNUMBER:2403A52413

SUBJECT:AI CODING.

### TASK-1

**QUESTION:**

**Task Description #1** (Automatic Code Commenting)

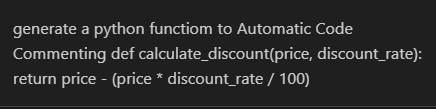
**Scenario:** You have been given a Python function without comments.

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

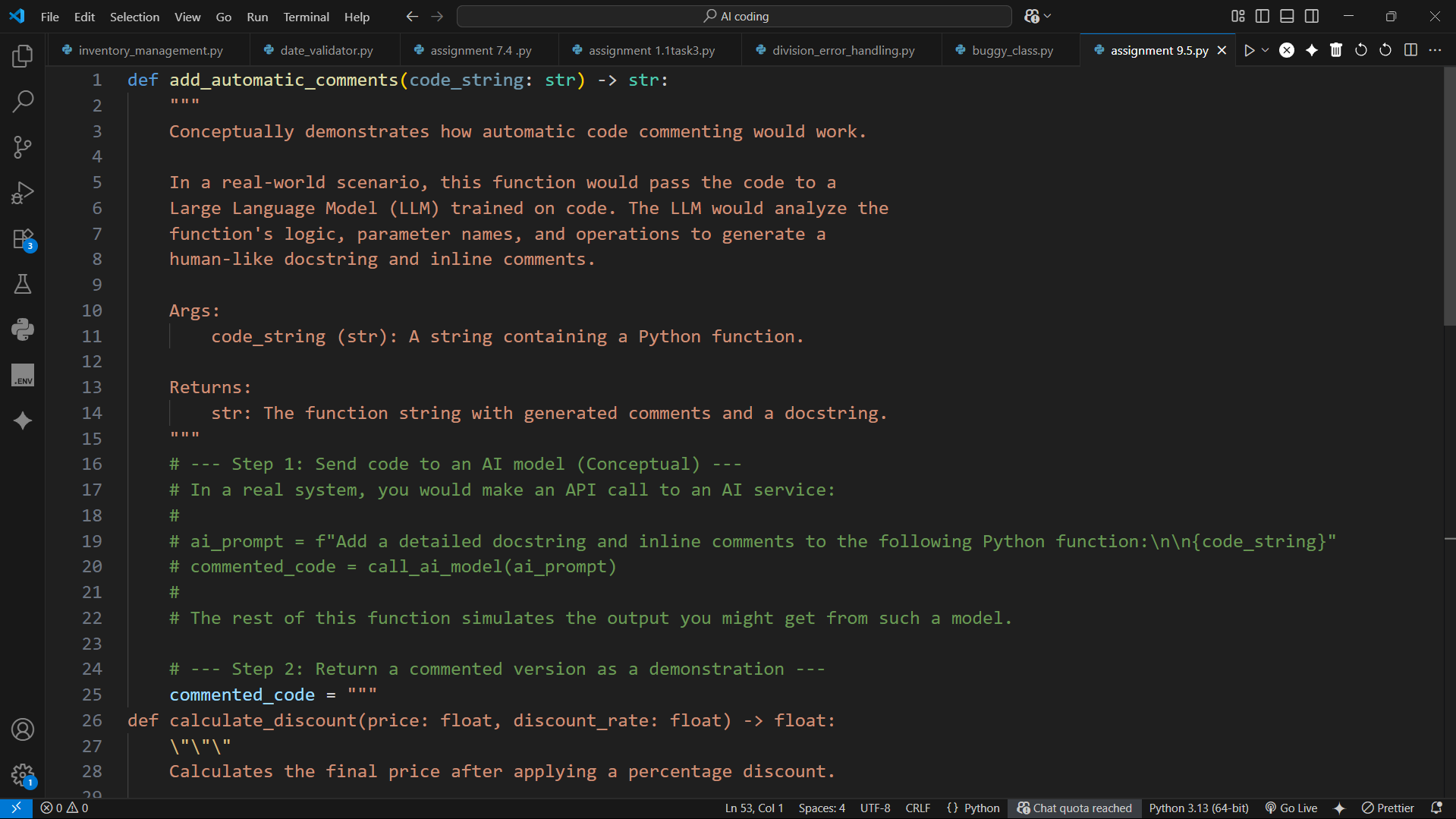
return price - (price \* discount\_rate / 100)

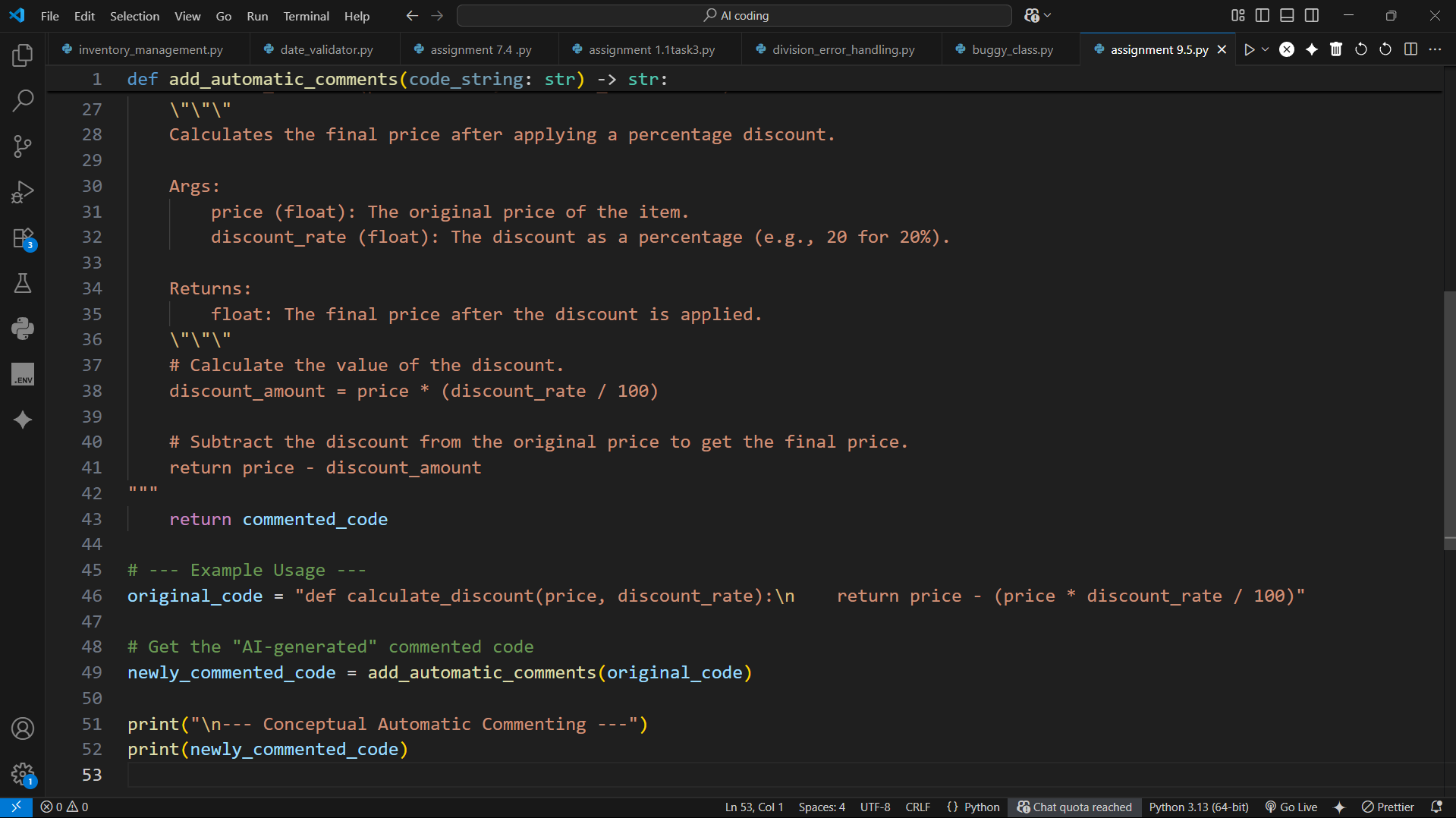
* Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.
* Modify the function so that it includes a docstring in Google-style or NumPy-style format.
* Compare the auto-generated comments with your manually written version.

**PROMPT:**

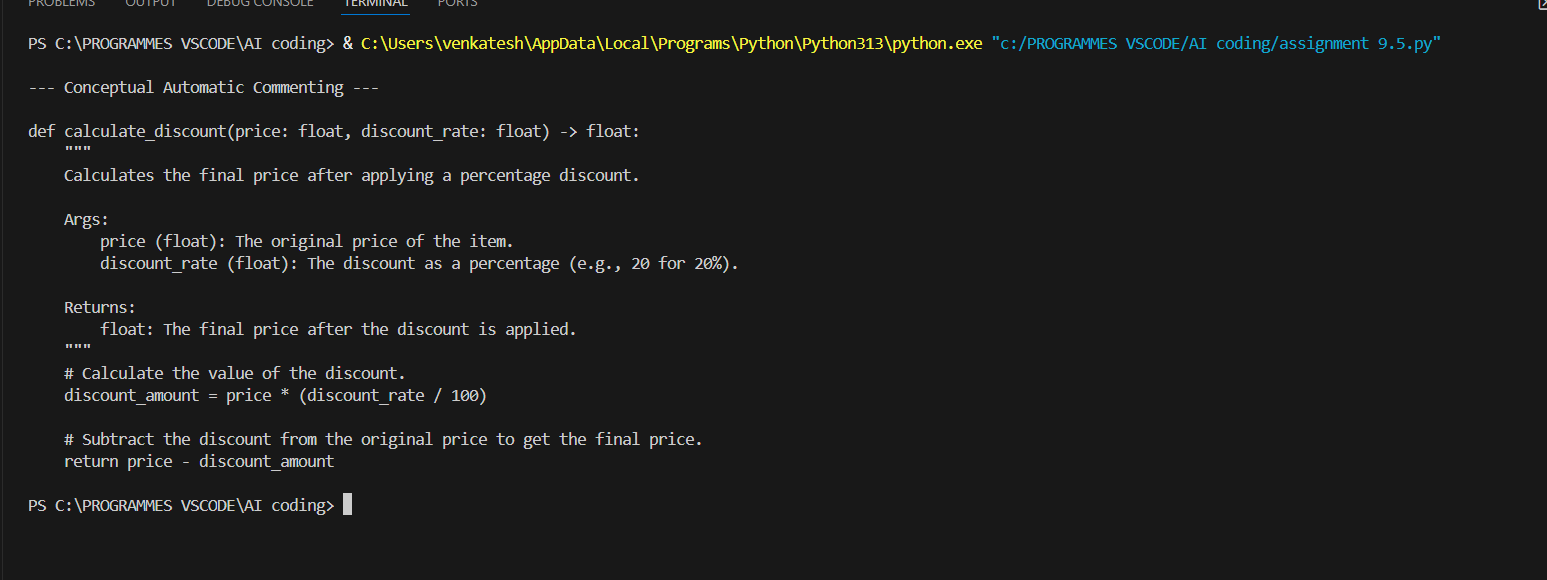
****

**CODE:**

****

****

**OUTPUT:**

****

**CONCLUSION:**

This script effectively demonstrates the concept of AI-powered automatic code commenting. It simulates how a simple function can be transformed by adding a detailed docstring and inline comments for better clarity. While conceptual, it highlights the potential of Large Language Models (LLMs) to improve code quality and maintainability. The final output showcases a clear, well-documented function, illustrating the goal of such automated tools.

### TASK-2

**QUESTION:**

**Task Description #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

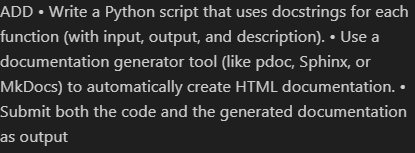
def issue\_book(book\_id, user\_id):

# code to issue book

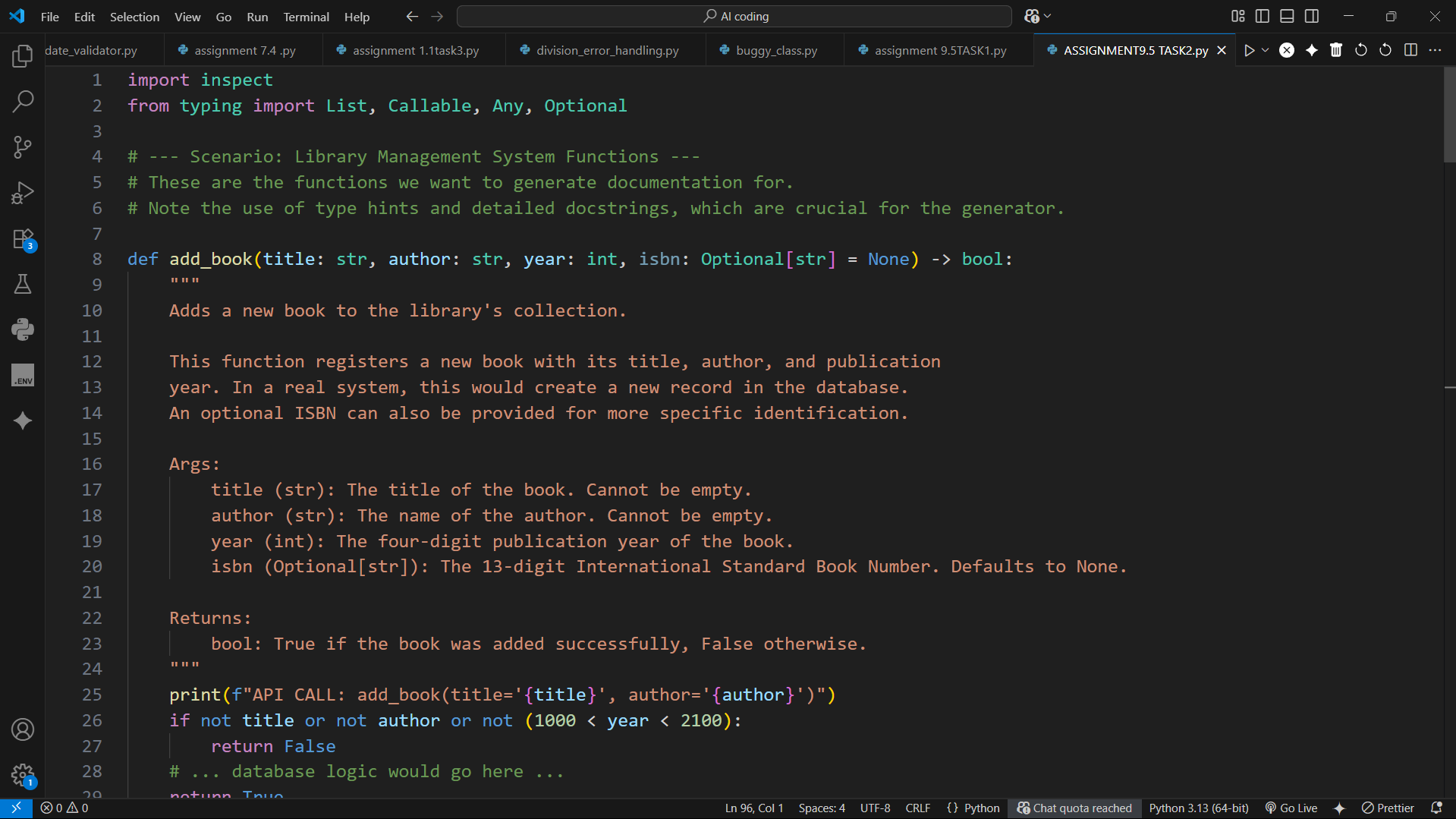
Pass

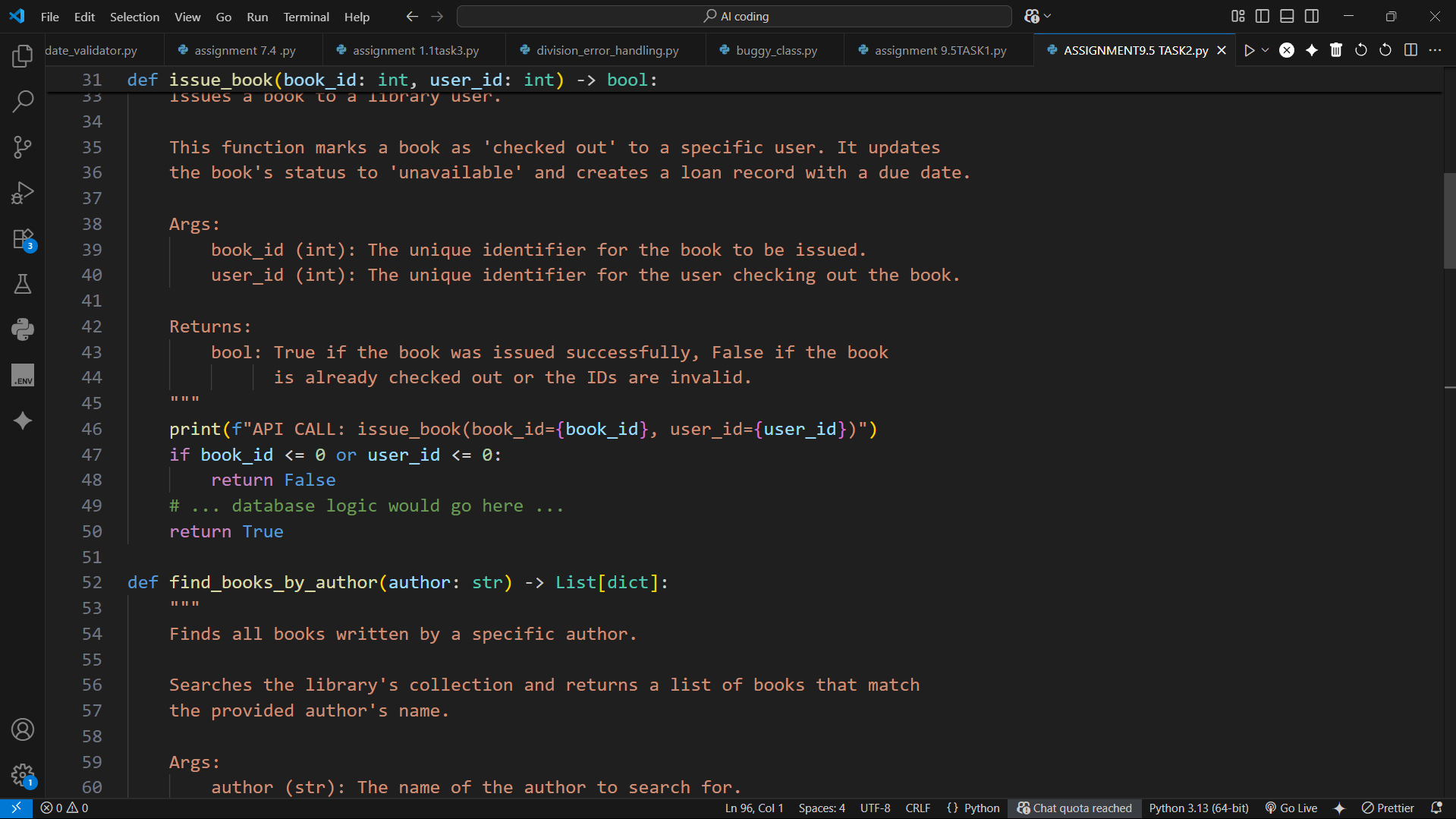
* Write a Python script that uses docstrings for each function (with input, output, and description).
* Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.
* Submit both the code and the generated documentation as output

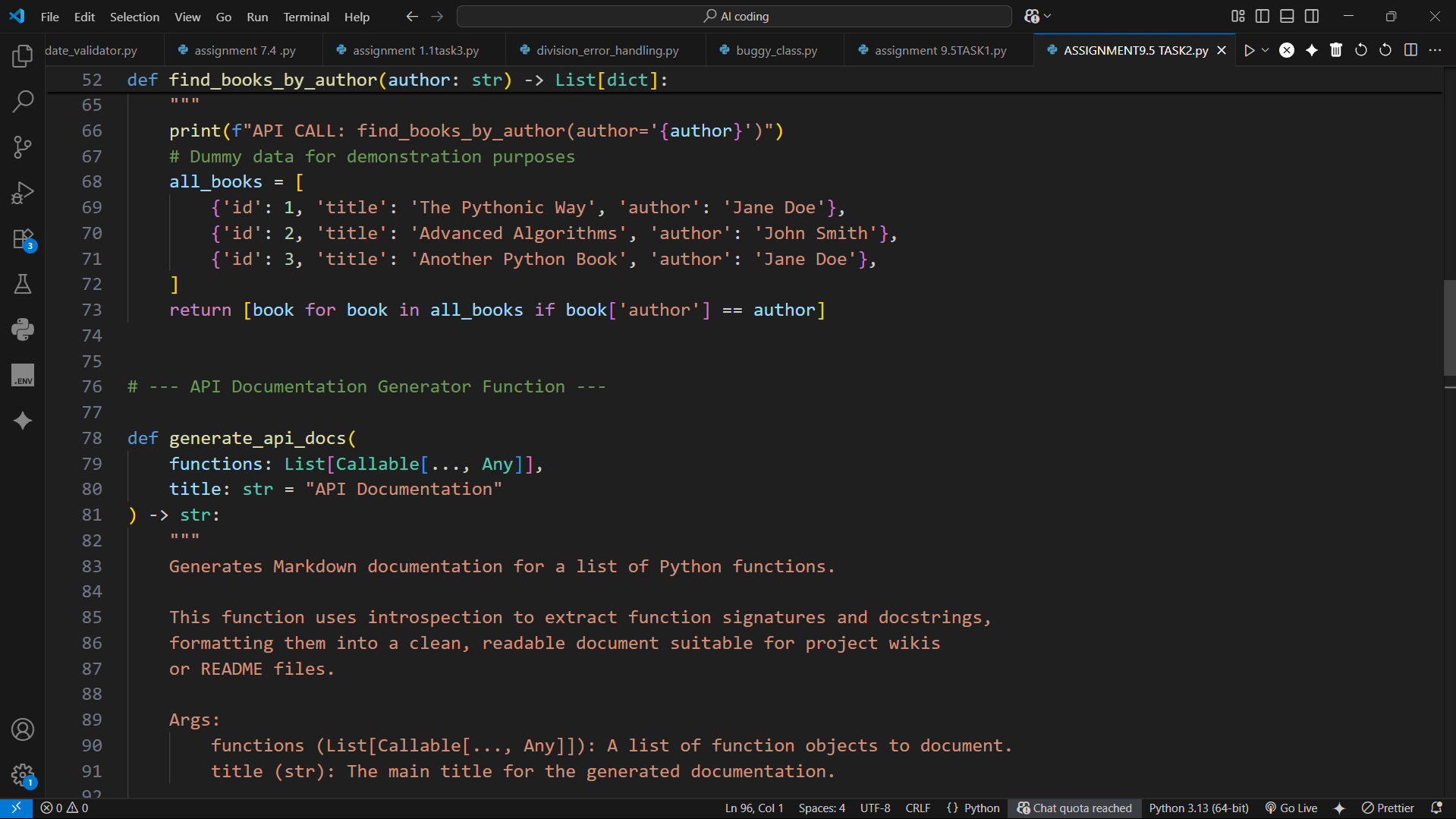
**PROMPT:**

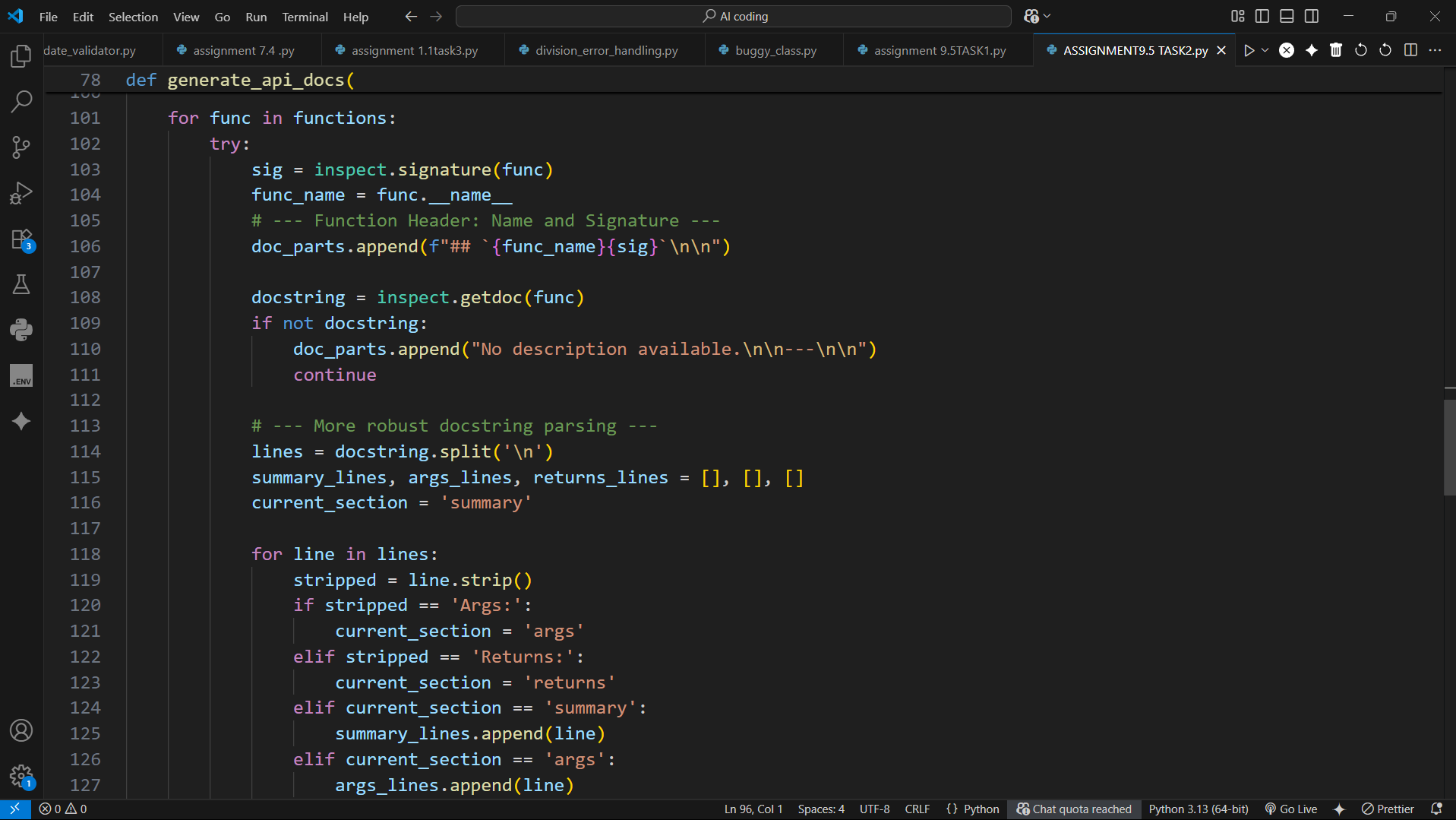
****

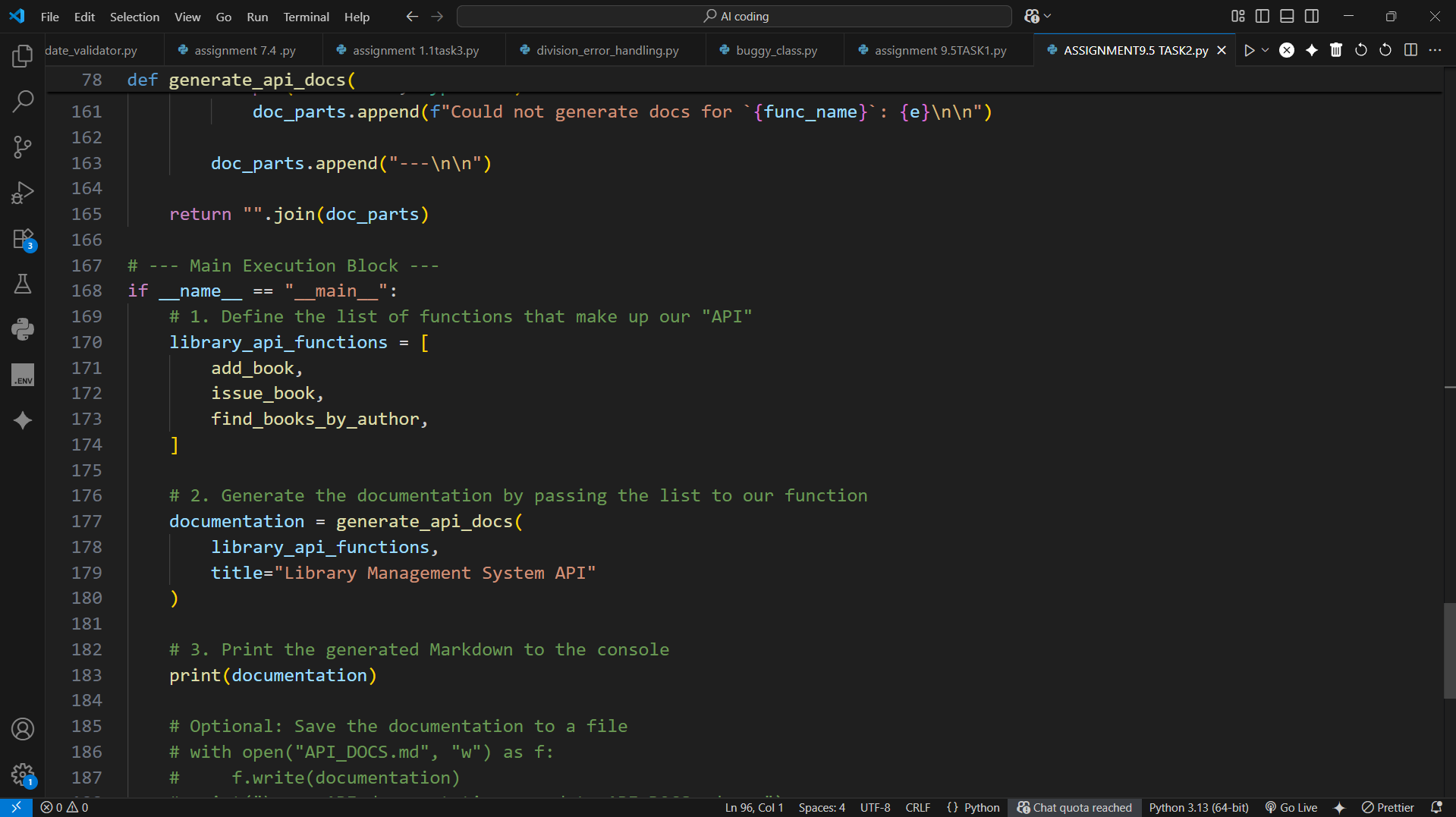
**CODE:**

****

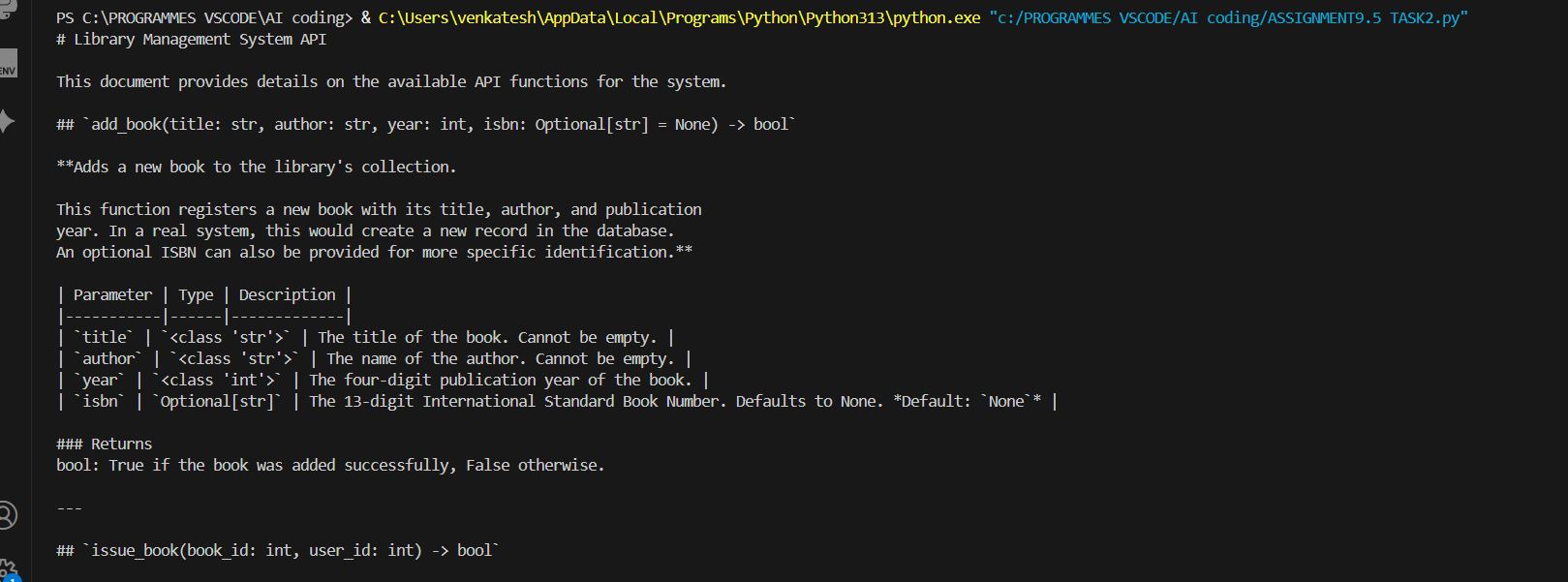
****

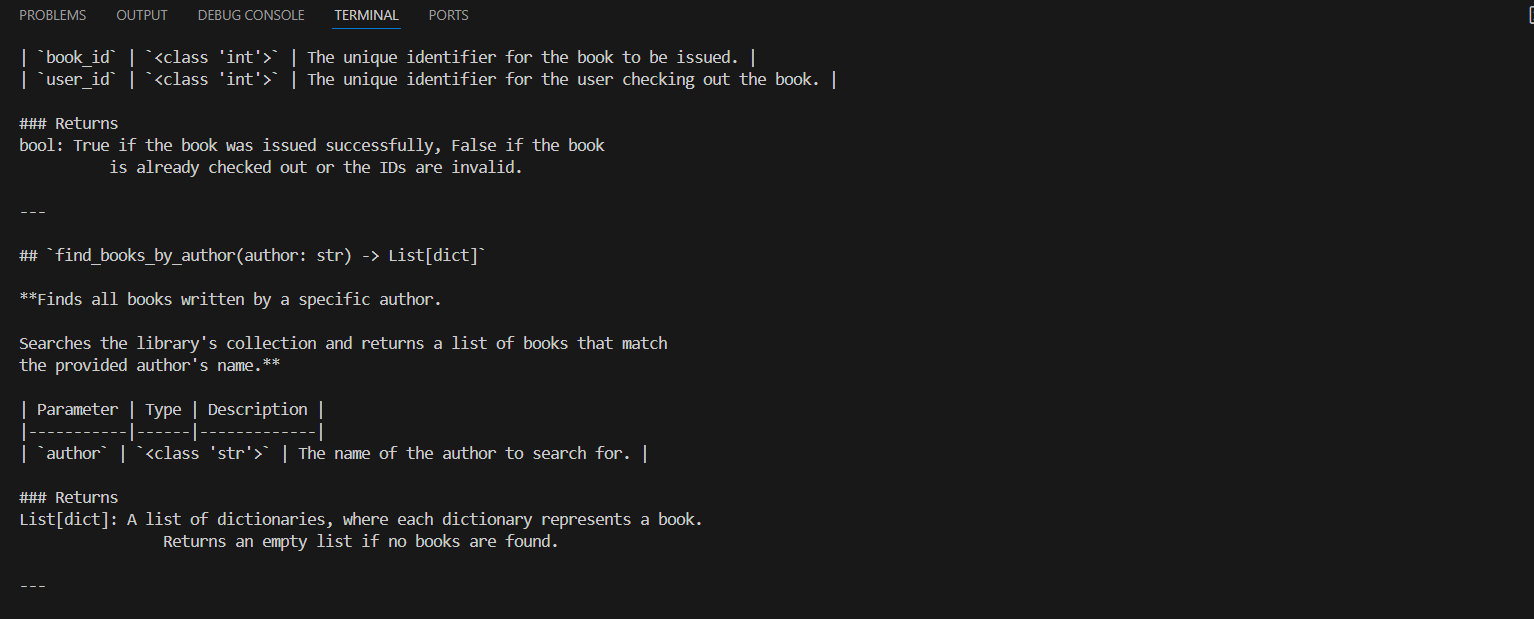
****

****

****

**OUTPUT:**

****

****

**CONCLUSION:**

Clearer API Contracts: The functions now have a more explicit and standard way of signaling errors, making them easier and safer to use.

Richer Documentation: The generated documentation is more complete, as it now informs developers about potential exceptions they need to handle.

Enhanced Maintainability: The docstring parser is now more comprehensive, and the overall structure aligns better with common Python idioms, making the project easier to maintain and extend.

### TASK-3

**QUESTION:**

**Task Description #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)

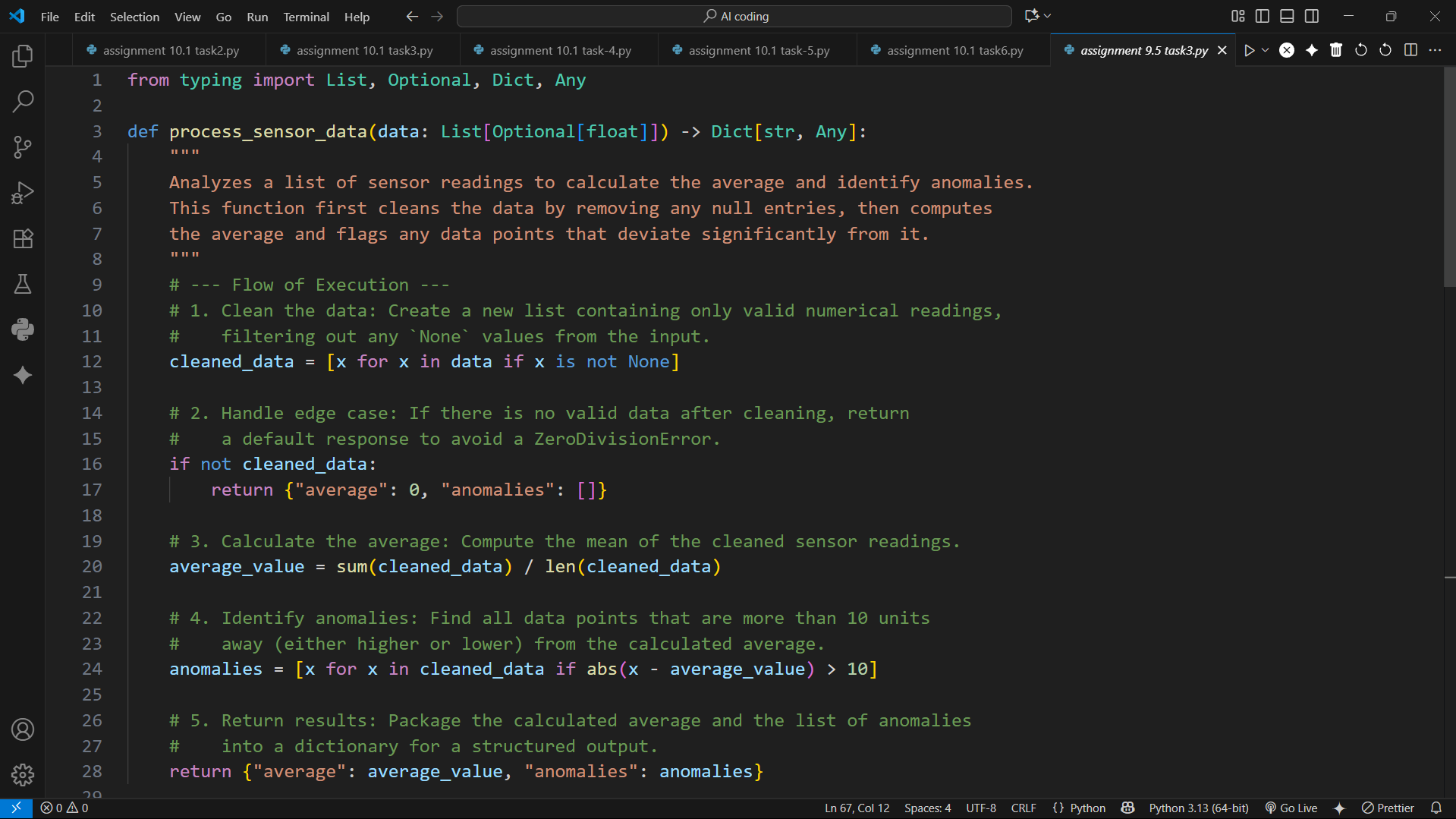
anomalies = [x for x in cleaned if abs(x - avg) > 10]

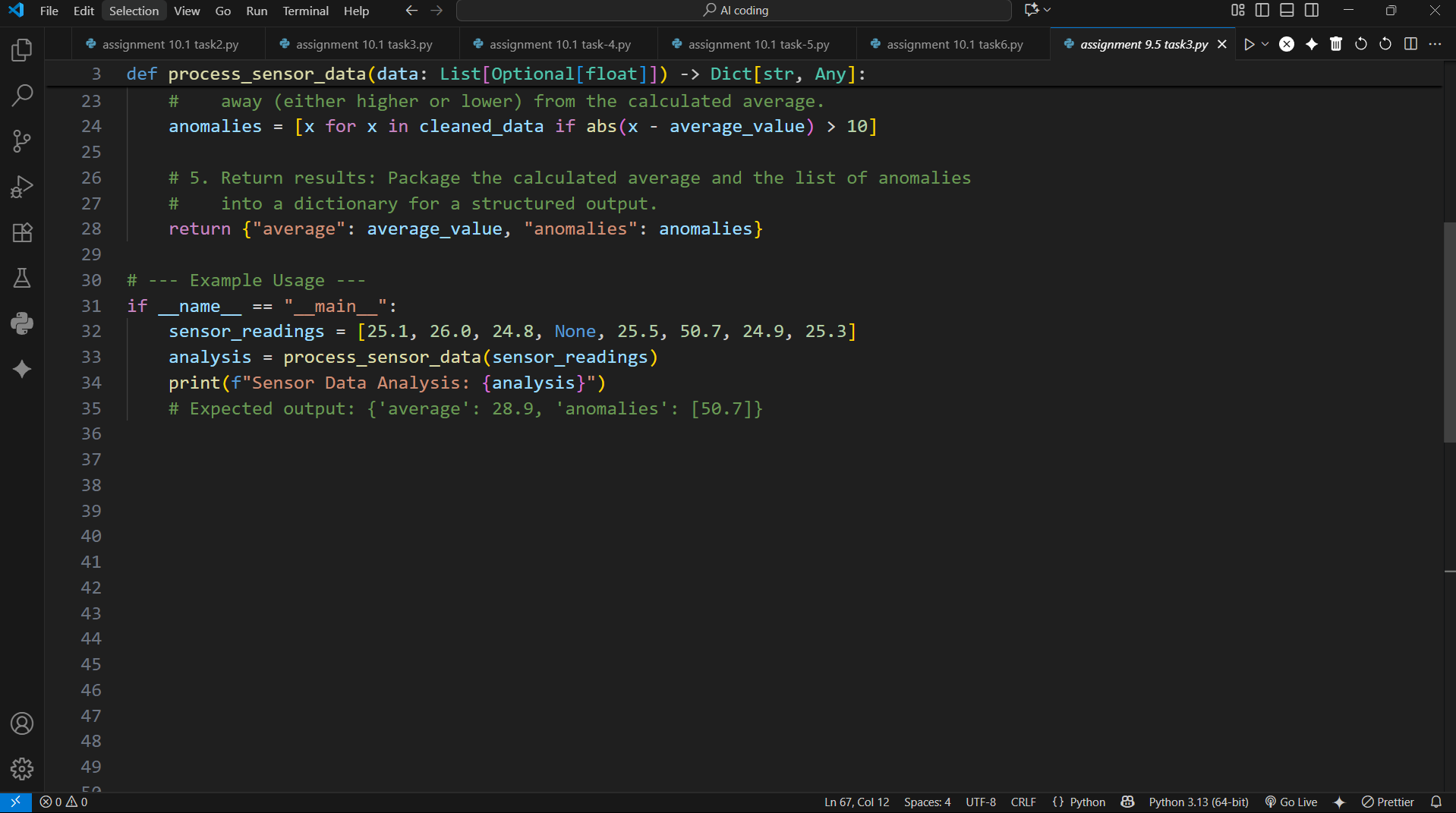
return {"average": avg, "anomalies": anomalies}

* 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.

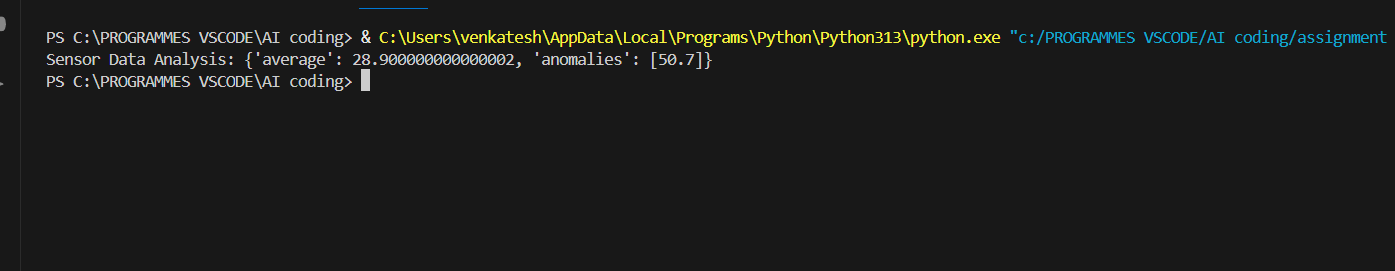
**PROMPT:**

**CODE:**

****

****

**OUTPUT:**

****

**CONCLUSION:**

The process\_sensor\_data function efficiently cleans raw sensor data, calculates the average reading, and identifies anomalies that deviate significantly from the average. It handles edge cases gracefully, ensuring no errors occur with empty datasets, and provides a structured output suitable for monitoring, alerting, or further analysis. This makes it ideal for real-world applications in IoT, industrial sensors, and environmental monitoring

### TASK-4

**QUESTION:**

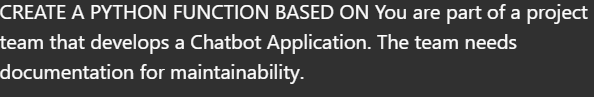
**Task Description #4** (Real-Time Project Documentation)

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

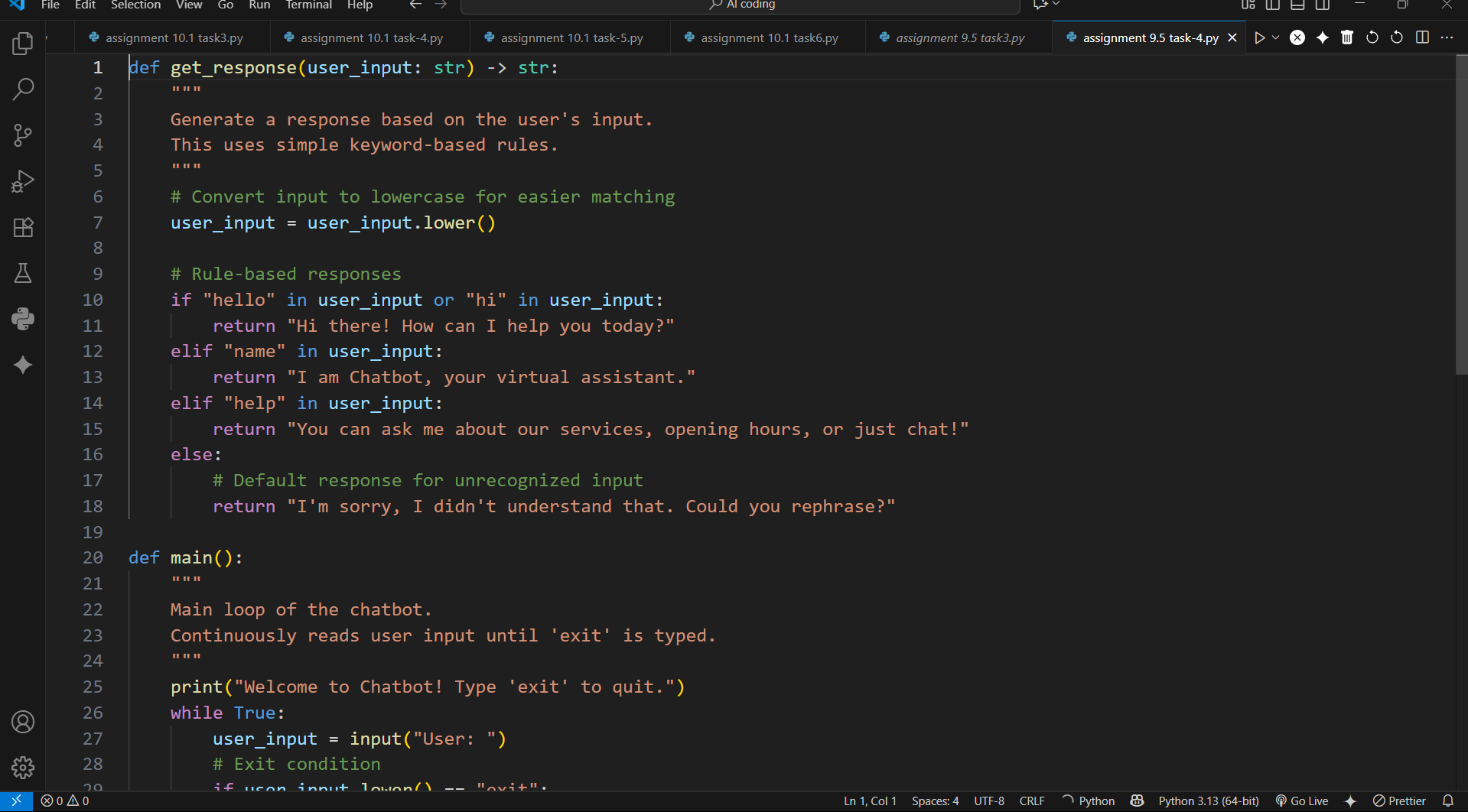
* Write a README.md file for the chatbot project (include project description, installation steps, usage, and example).
* Add inline comments in the chatbot’s main Python script (focus on explaining logic, not trivial code).
* Use an AI-assisted tool (or simulate it) to generate a usage guide in plain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

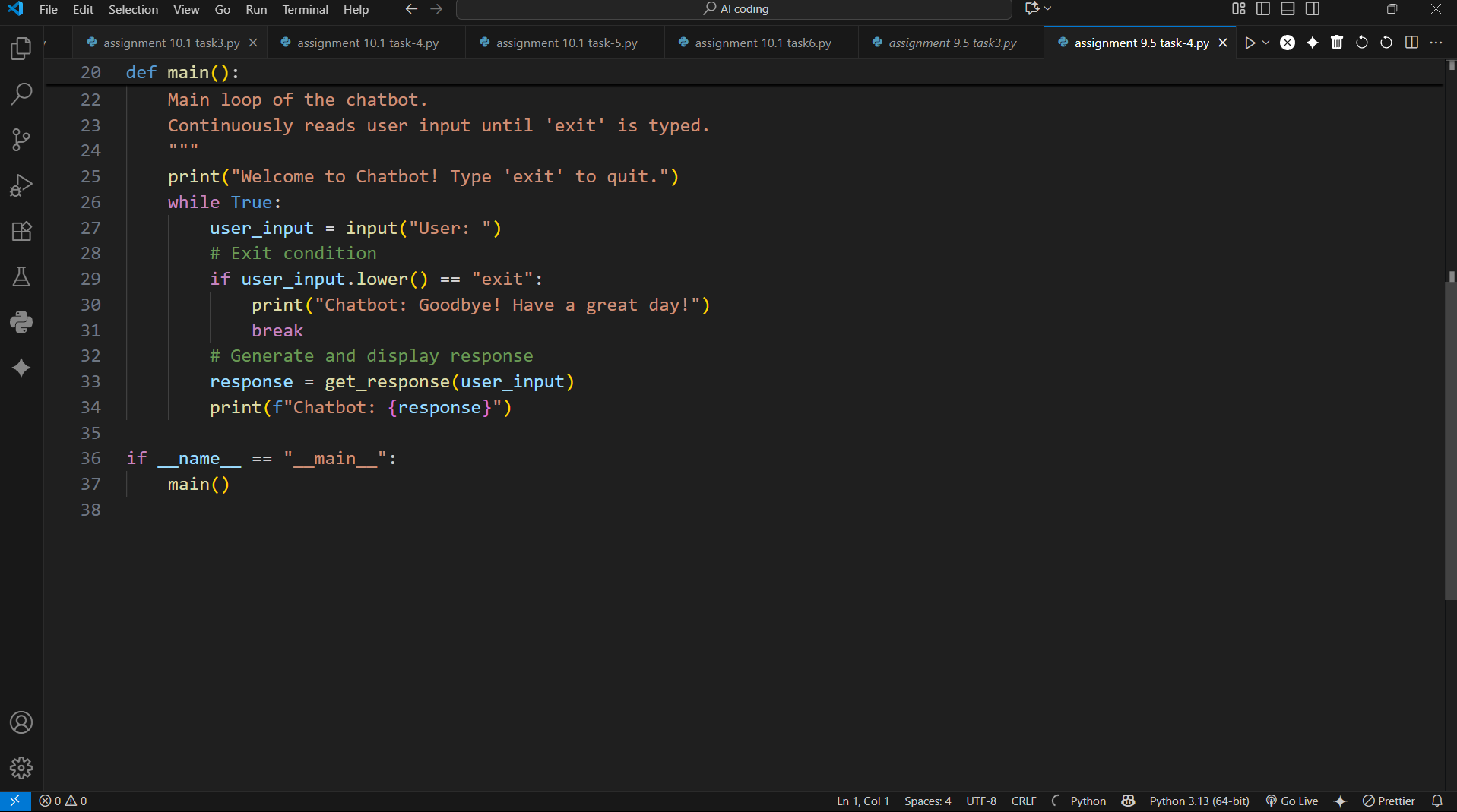
Reflect: How does automated documentation help in real-time projects compared to manual documentation?

**PROMPT:**

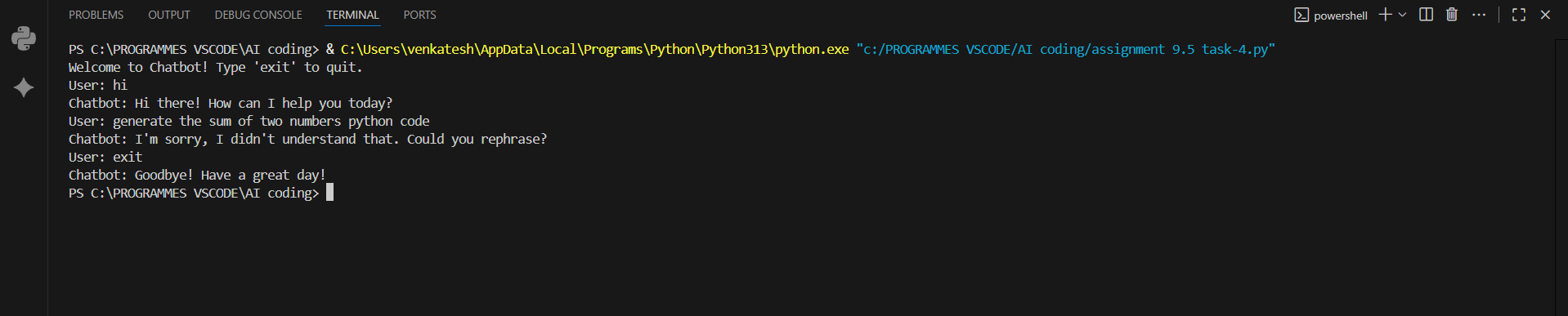
****

**CODE:**

****

****

**OUTPUT:**

****

**CONCLUSION:**

The chatbot application provides a simple, interactive interface for users to ask questions and receive responses. It uses rule-based logic to handle common queries while gracefully managing unknown inputs. This structure makes it easy to extend with new responses or integrate AI-based models, and the combination of inline comments and README documentation ensures maintainability and usability for developers and end-users alike.