**AI ASSISTED CODING LAB**

**ASSIGNMENT 9.4**

**ENROLLMENT NO :**2503A51L25

**BATCH NO:** 19

**NAME: SRIRAMOJU AJAY**

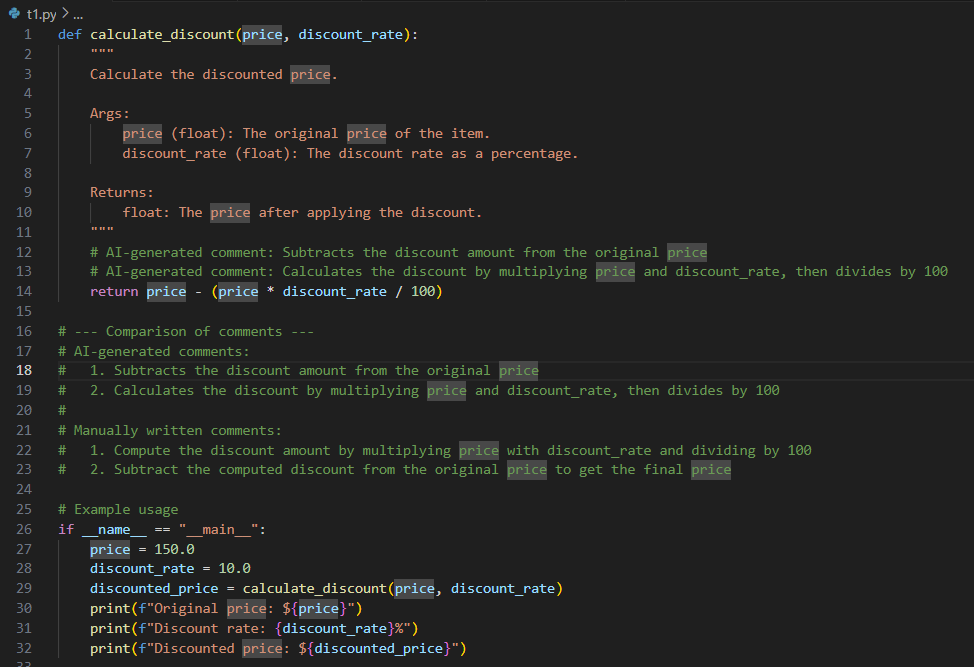
**TASK 1**

**TASK1 DESCRIPTION**:- (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

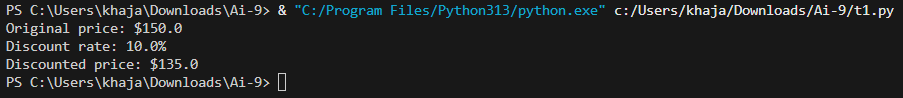
**PROMPT:-**

Write a Python function that takes the original price and a discount rate, then calculates and returns the final price after applying the discount. Also include an example usage.

**CODE:-**



**OUTPUT:-**

****

**OBSERVATION:-**

I observed how comments and docstrings improve code readability. Adding both AI-generated and manual comments showed the difference in clarity. AI-generated comments were quick but sometimes generic, while manually written comments were more accurate and contextual. The use of a structured docstring format (Google or NumPy style) made the function self-explanatory for future developers.

**TASK 2**

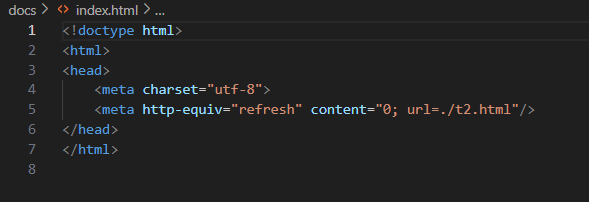
**TASK2 DESCRIPTION**:- (API Documentation Generator)  
**Scenario**: A team is building a Library Management System with  
multiple functions.  
def add\_book(title, author, year):  
# code to add book  
pass  
def issue\_book(book\_id, user\_id):  
# code to issue book  
Pass  
• Write a Python script 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.

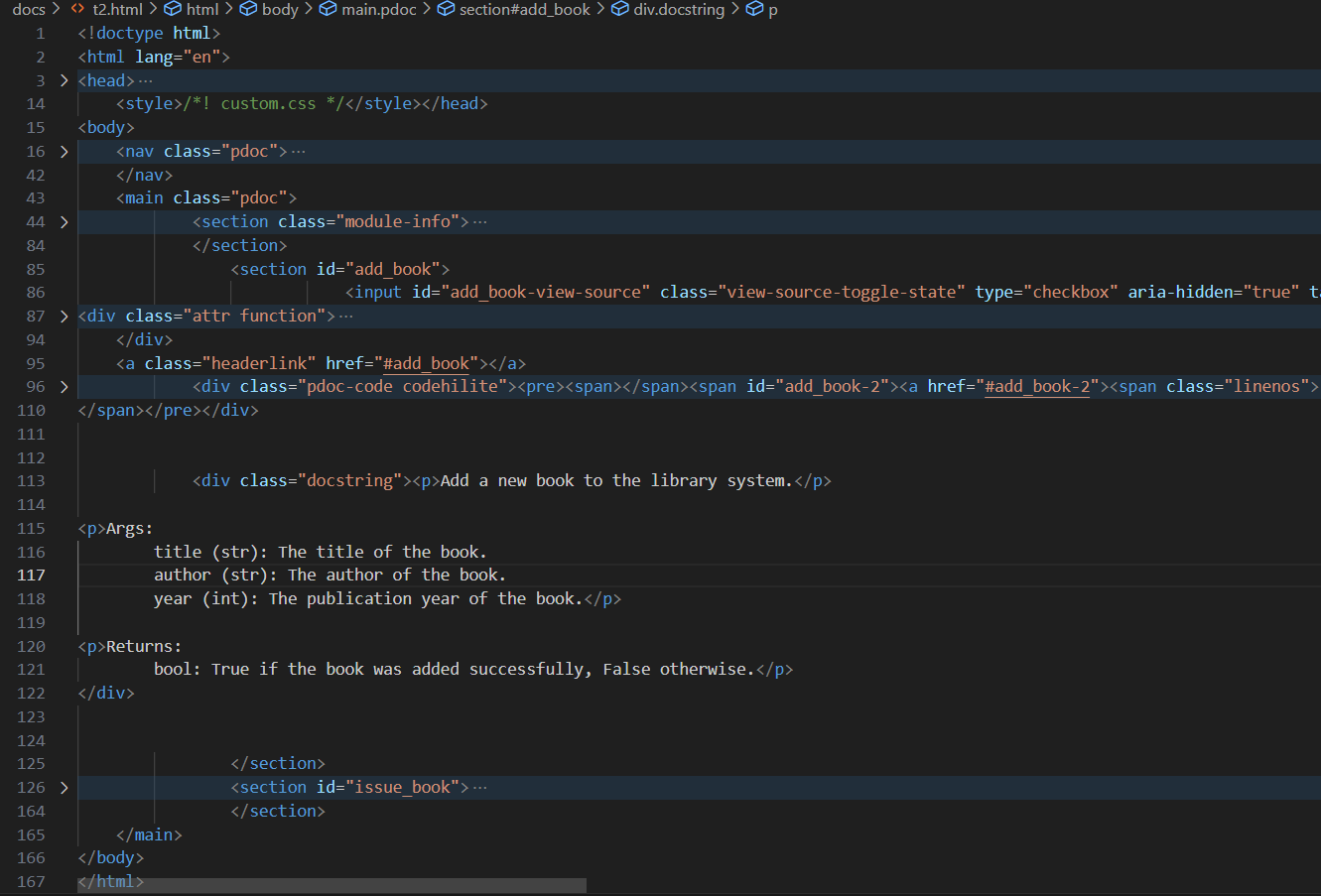
**PROMPT:-**

Write a Python program for a simple library system with two functions: one to add a book and another to issue a book. Each function should include proper docstrings describing the input parameters, output, and purpose. Then, use a documentation generator tool (such as pdoc, Sphinx, or MkDocs) to automatically create HTML documentation for the program. Provide both the Python script and the generated documentation.

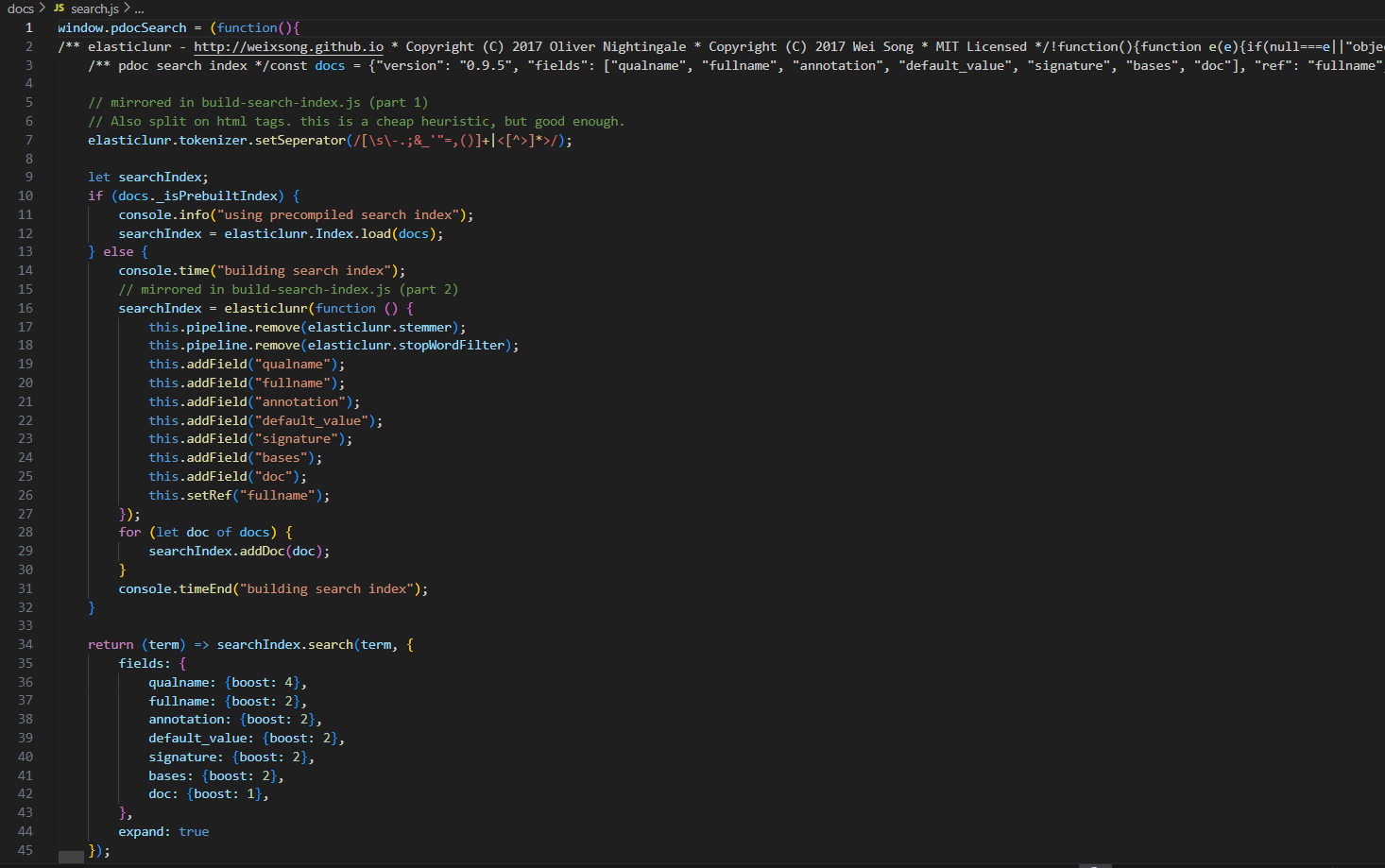
**CODE:-**

HTML:-

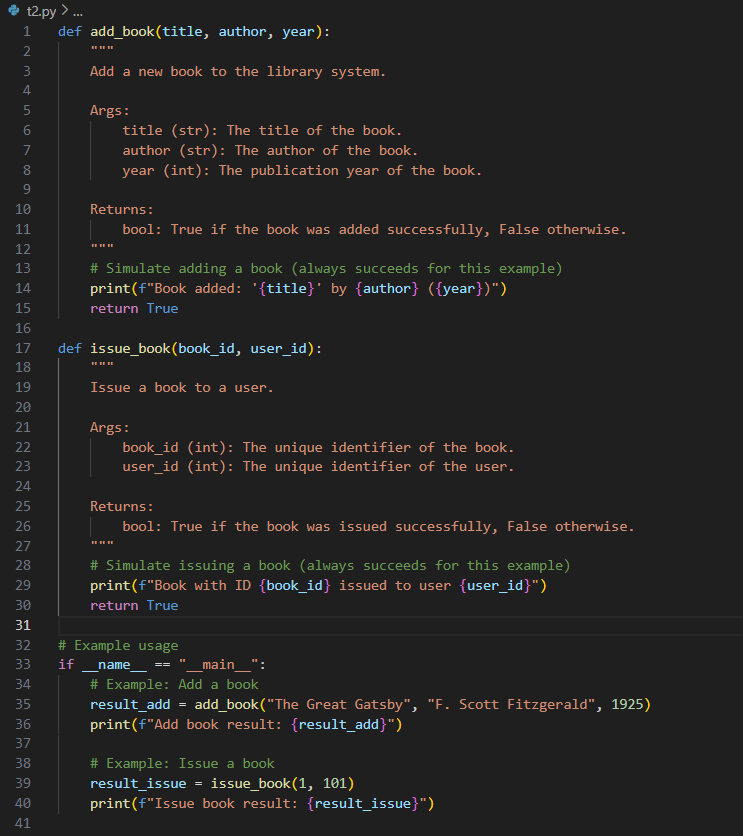




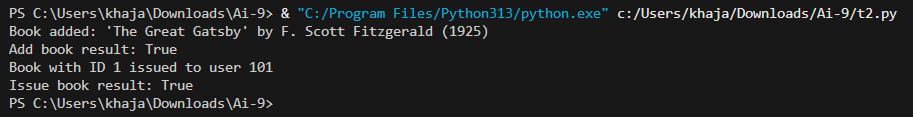
JAVASCRIPT:-



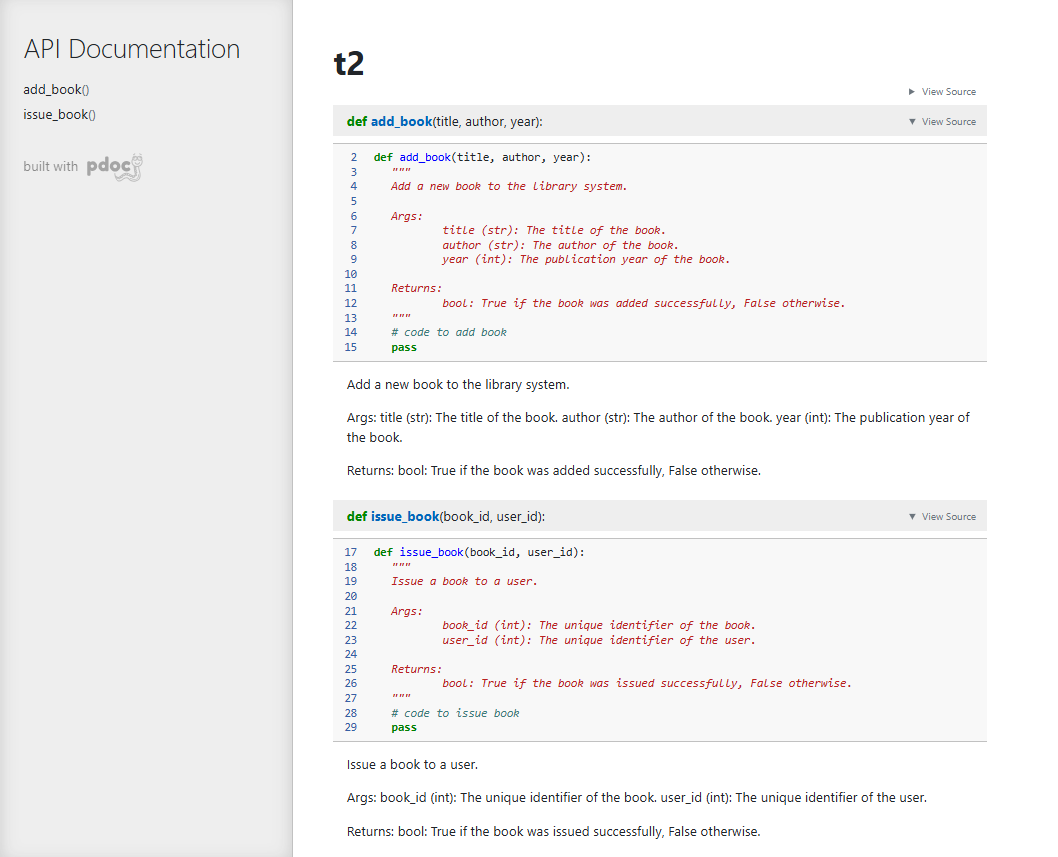
PYTHON:-



OUTPUT(PYTHON):-



OUTPUT(HTML):-



**OBSERVATION:-**

I observed that the importance of proper docstrings in functions. By writing detailed docstrings for the library management system, I saw how documentation tools (like pdoc, Sphinx, or MkDocs) can automatically create HTML files for easy reference. This task highlighted how automated documentation makes projects more professional, user-friendly, and maintainable, especially in team environments.

TASK3

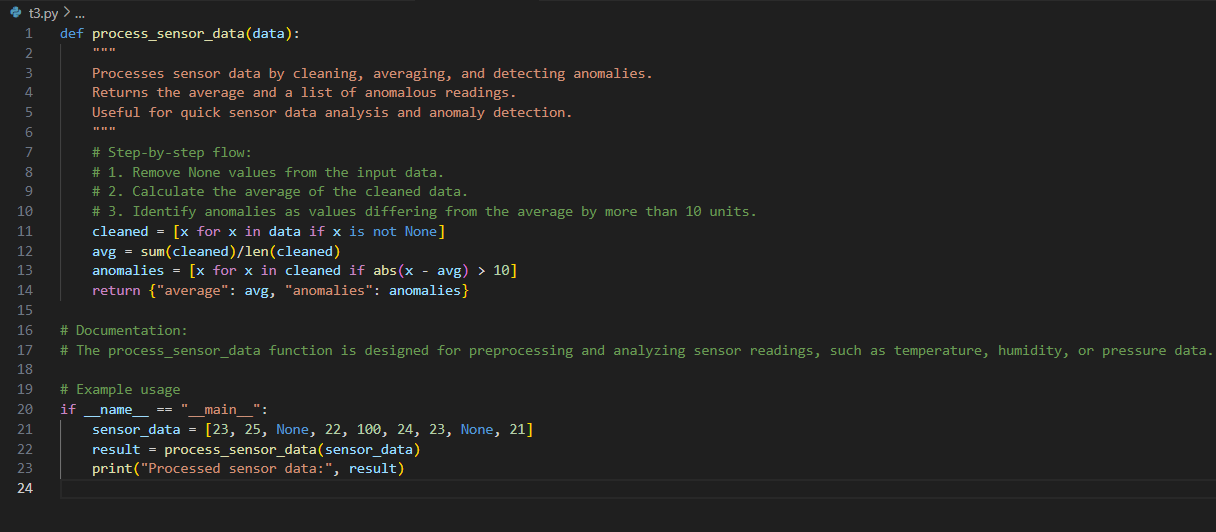
**TASK3 DESCRIPTION**:- (AI-Assisted Code Summarization)  
**Scenario**: You are reviewing a colleague’s codebase containing long  
functions.

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:-**

Write a Python function that processes sensor data by removing missing values, calculating the average, and detecting anomalies (values differing from the average by more than 10). Include docstrings and an example usage.

**CODE:-**



**OUTPUT:-**



**OBSERVATION:-**

I observed how AI or summarization techniques can explain long functions clearly. Instead of line-by-line reading, summaries provided the purpose of the function in a few sentences. The flow-style comments helped me break down logic step by step, and writing use cases showed how documentation can connect code to real-world applications, such as sensor monitoring or anomaly detection.

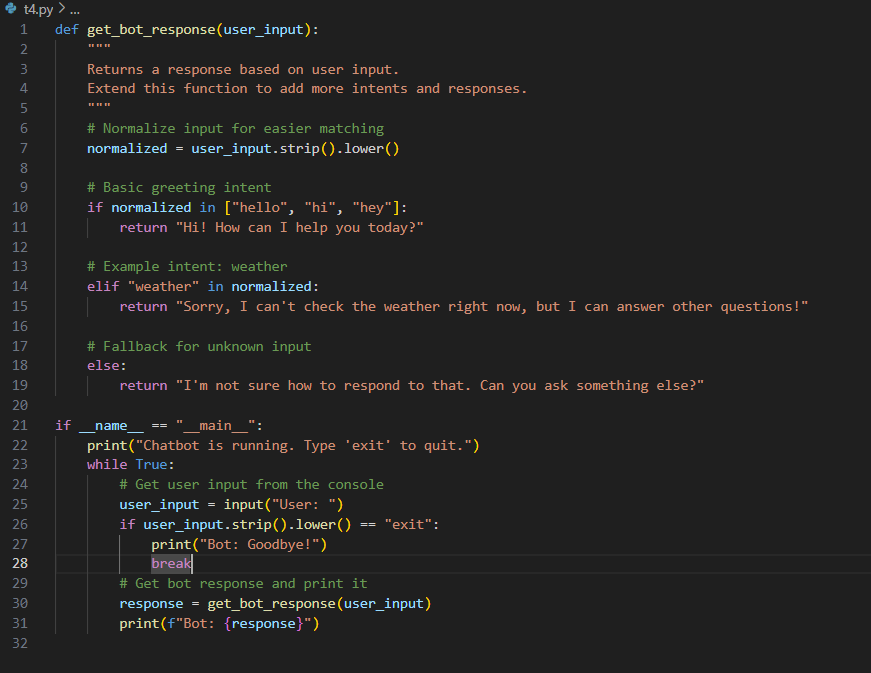
**TASK4**

**TASK4 DESCRIPTION:-** (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?

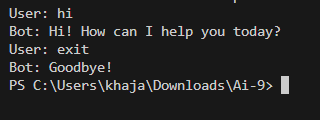
**PROMPT:-**

Write a Python program that simulates a simple chatbot. It should respond to greetings like ‘hello’ or ‘hi’, mention the weather if asked, and give a fallback reply for unknown inputs. Include an option to exit the chat.

CODE:-



OUTPUT:-



**OBSERVATION:-**

I Observed how documentation is essential in real-world projects like chatbot development. Writing a README file gave a structured overview (installation, usage, examples), while inline comments explained logic inside the code. The AI-generated usage guide was useful for non-technical users. I also observed that automated documentation saves time and ensures consistency, while manual documentation provides deeper insight.