**2403a51238**

**T.Chandana**

**Batch:11**

**AI-ASSISTED-CODING**

**Assignment-9.4**

**Task Description #1** (Automatic Code Commenting)  
Scenario: You have been given a Python function without comments.

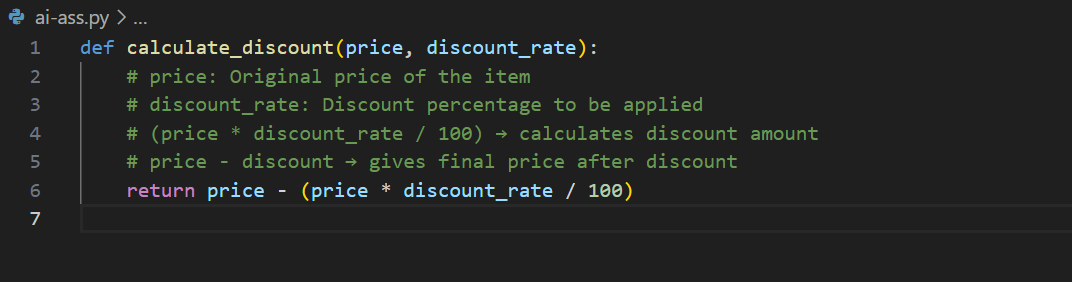
def calculate\_discount(price, discount\_rate):  
return price - (price \* discount\_rate / 100)

• 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

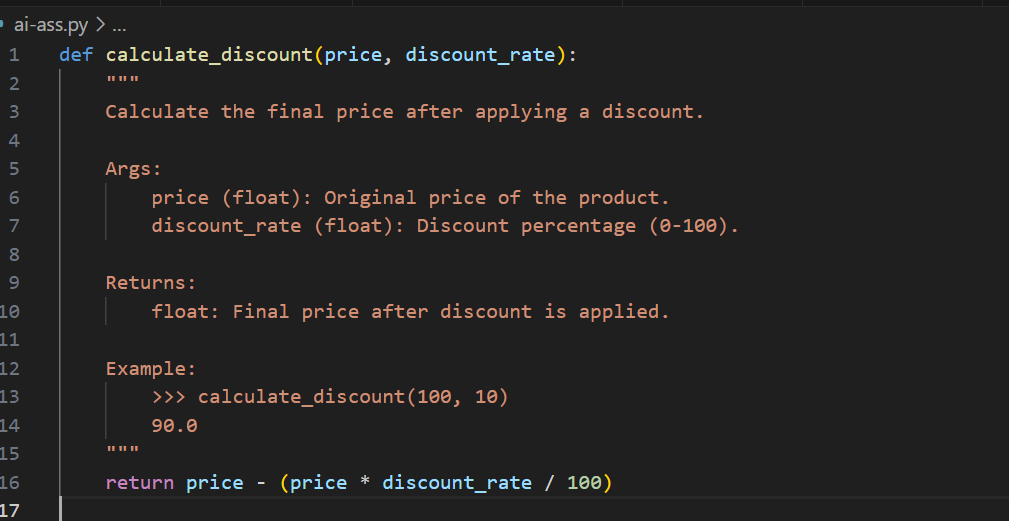
**Prompt:**

*“def calculate\_discount(price, discount\_rate):  
return price - (price \* discount\_rate / 100)*

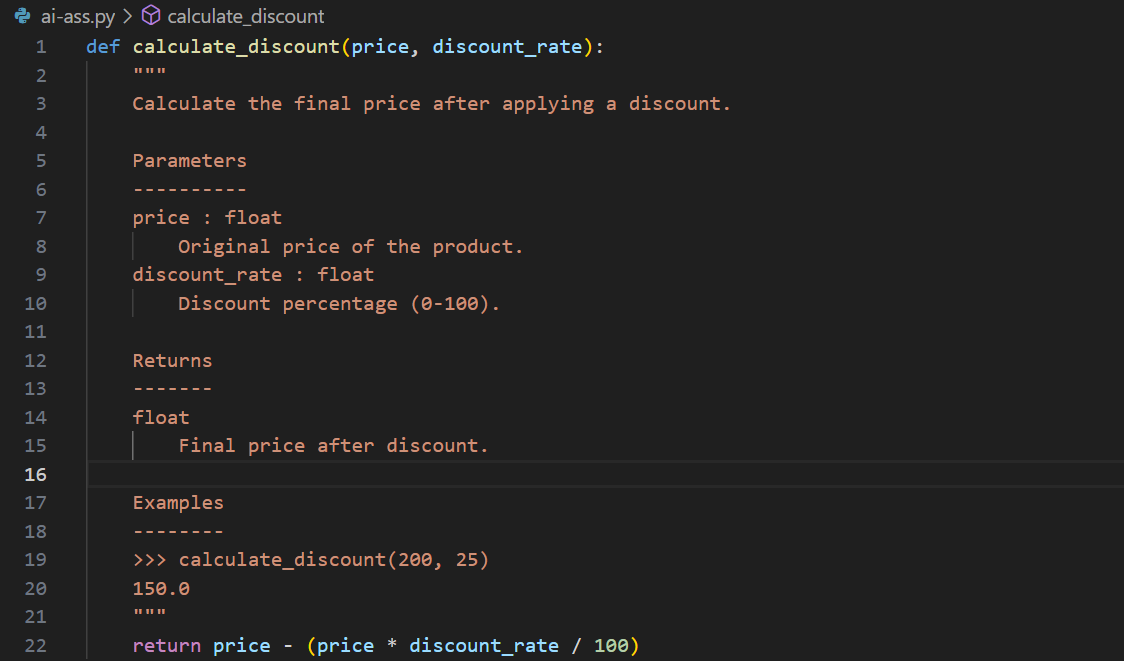
*to generate line-by-line comments for the function.”*



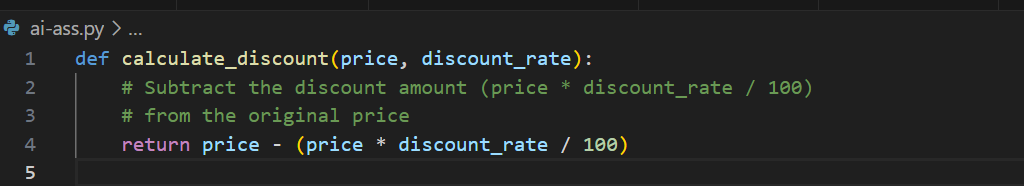
**With Docstring (Google Style):**



**With Docstring (NumPy Style):**



**AI-Generated Commenting (Simulation):**



**Comparison (Manual vs AI)**

* **AI version** → short, single-line, explains formula only.
* **Manual version** → detailed, explains each parameter, step, and purpose.
* **Docstring version** → standardized, good for auto-documentation tools (e.g., Sphinx, pdoc).

**Task Description #2** (API Documentation Generator)  
Scenario: A team is building a Library Management System with multiple functions.

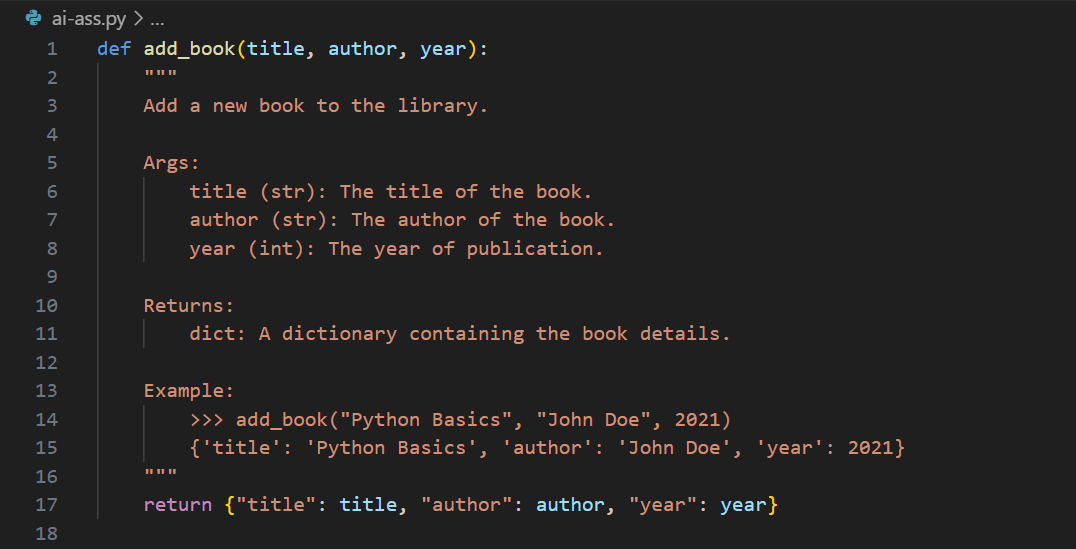
def add\_book(title, author, year):  
# code to add book pass  
def issue\_book(book\_id, user\_id):  
# code to issue book Pass

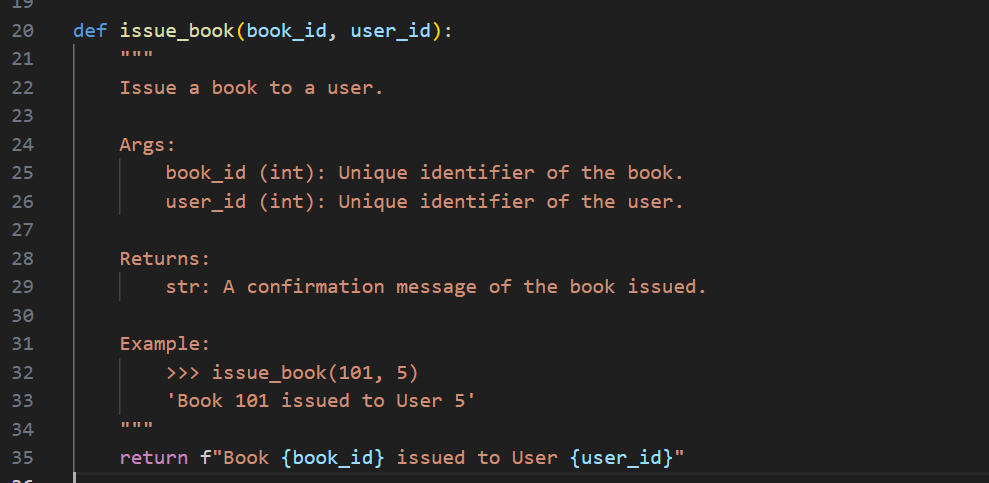
• Write a Python script that uses docstrings for each function (with  
input, output, and description).  
• Use a documentation generator tool (like pdoc, Sphinx, or  
MkDocs) to automatically create HTML documentation.  
• Submit both the code and the generated documentation as output.

**Prompt:**

*“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”*





**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}

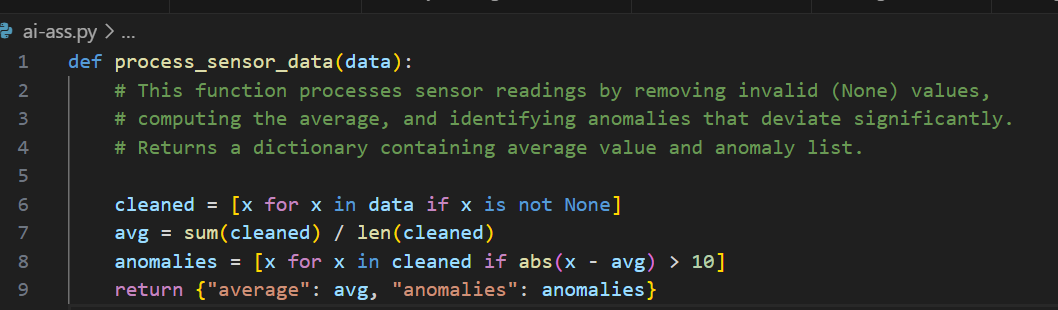
• Generate a summary comment explaining the purpose of the  
function in 2–3 lines.  
• Create a flow-style comment (step-by-step explanation).  
• Write a short paragraph of documentation describing possible use  
cases of this function in real-world scenarios.

**Prompt:**

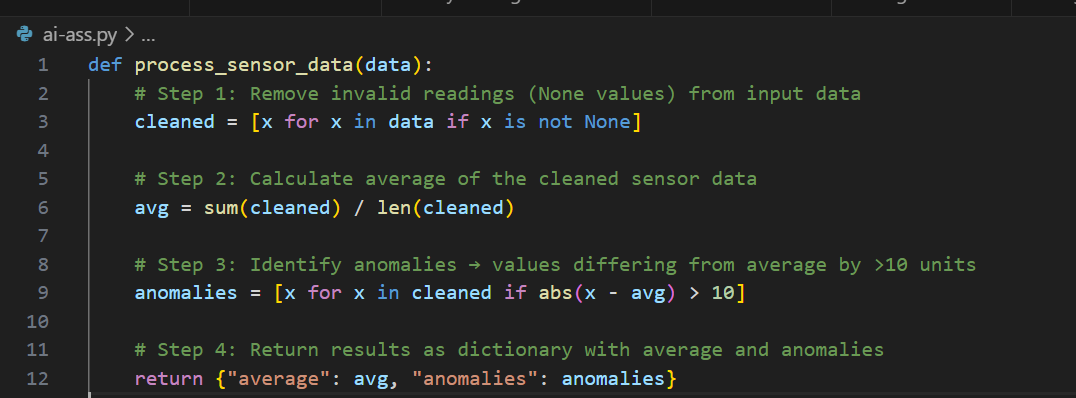
*“def process\_sensor\_data(data):  
cleaned = [x for x in data if x is not None]  
avg = sum(cleaned)/len(cleaned)  
anomalies = [x for x in cleaned if abs(x - avg) > 10]  
return {"average": avg, "anomalies": anomalies}*

*Generate a summary comment explaining the purpose of the  
function in 2–3 lines.  
Create a flow-style comment (step-by-step explanation).”*

**Function with Summary Comment (2–3 lines)**

****

**Flow-Style Comment (Step-by-Step)**

****

**Documentation :**

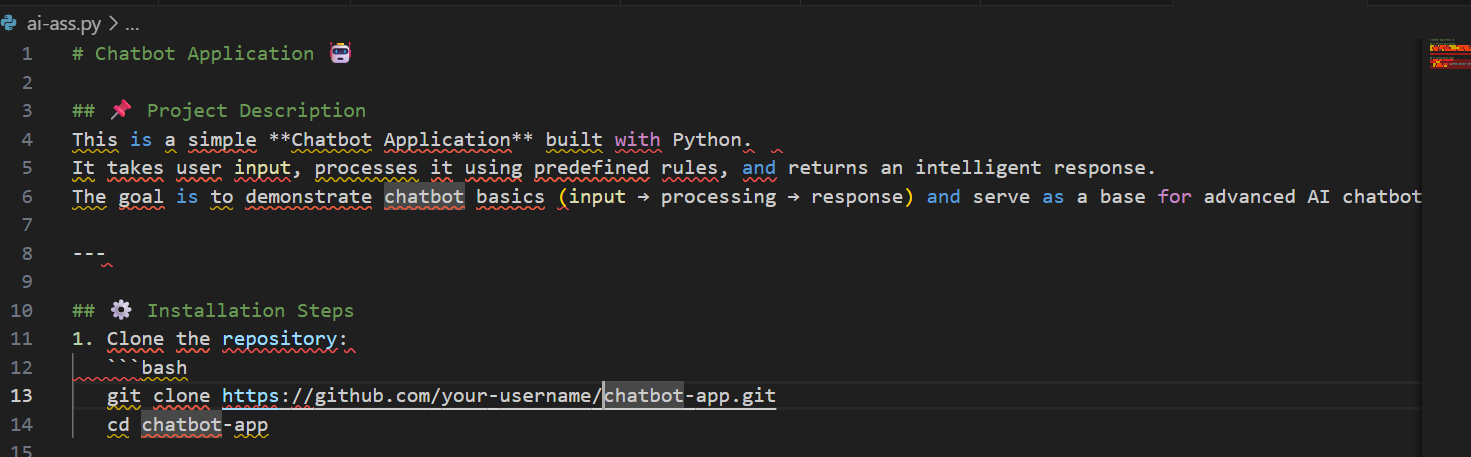
This function can be used in **IoT and sensor-based systems** to ensure data quality and detect unusual readings. For example, in a **smart home**, it could process temperature sensor data, filter out faulty readings, and detect sudden spikes indicating a malfunctioning sensor. In **industrial monitoring**, it could help identify anomalies in machine vibration or pressure readings, enabling predictive maintenance and reducing downtime. Similarly, in **healthcare devices**, it could clean and analyze patient vital signs (e.g., heart rate data) to flag abnormal conditions.

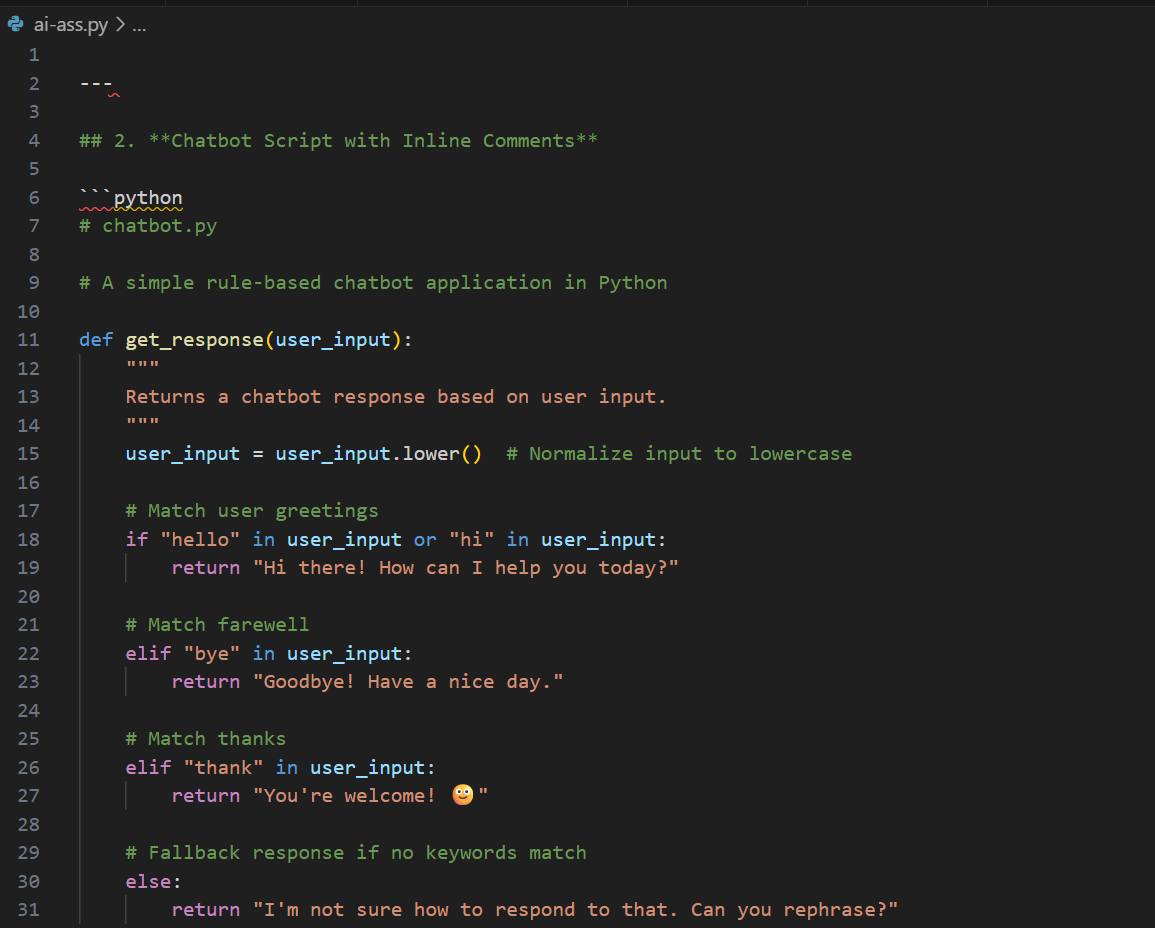
**Task Description #4** (Real-Time Project Documentation)  
Scenario: You are part of a project team that develops a Chatbot  
Application. The team needs documentation for maintainability.

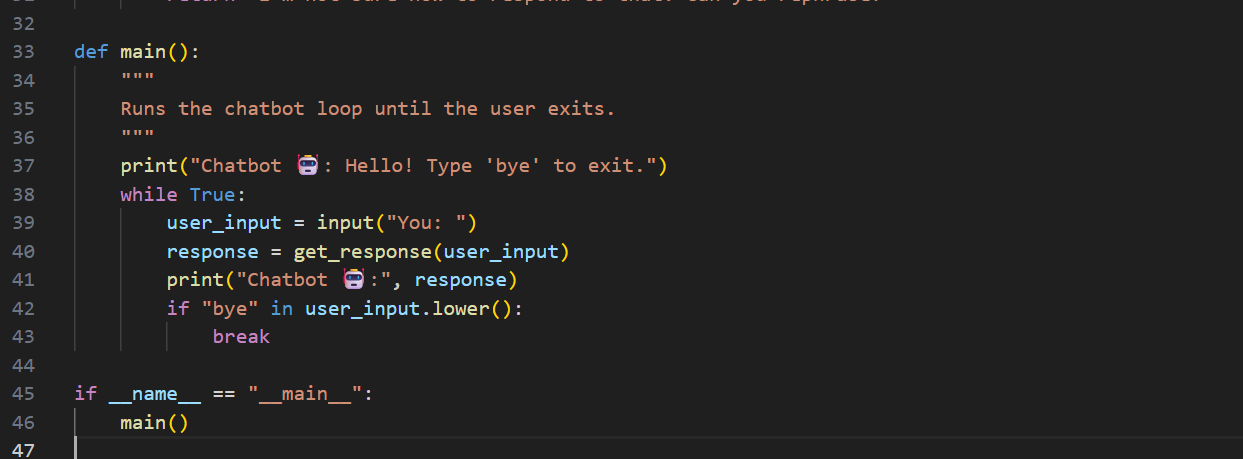
• Write a README.md file for the chatbot project (include project  
description, installation steps, usage, and example).  
• Add inline comments in the chatbot’s main Python script (focus  
on explaining logic, not trivial code).  
• Use an AI-assisted tool (or simulate it) to generate a usage guide  
in plain English from your code comments.  
• Reflect: How does automated documentation help in real-time  
projects compared to manual documentation?

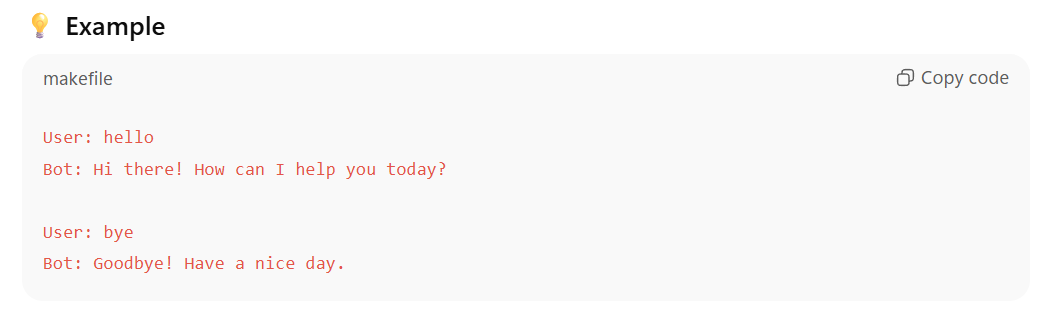
**Prompt:**

*“generate a usage guide in plain English from your code comments”*

******

******

******

******

**AI-Assisted Usage Guide (Generated from Comments)**

Plain English summary:

* Start the chatbot by running python chatbot.py.
* The chatbot greets the user and waits for input.
* If you type greetings like “hi” or “hello”, it replies with a friendly greeting.
* If you say “bye”, the chatbot ends the conversation.
* If you thank the bot, it replies politely.
* If your message doesn’t match any of the above, it asks you to rephrase.

**Reflection:**

Automated documentation tools help in **real-time projects** by:

* Keeping documentation **consistent with the code** (if code changes, docs can be auto-updated).
* Saving **developer time** (no need to write long manuals manually).
* Making projects easier to **maintain and onboard** new developers.
* Reducing human errors (e.g., outdated instructions in README).

Compared to **manual documentation**, automation ensures that:

* Docs evolve **along with the codebase**.
* Teams can generate **API references, usage guides, and examples** quickly.
* Developers can focus more on building features rather than maintaining documentation.