ASSIGNMENT:9.5

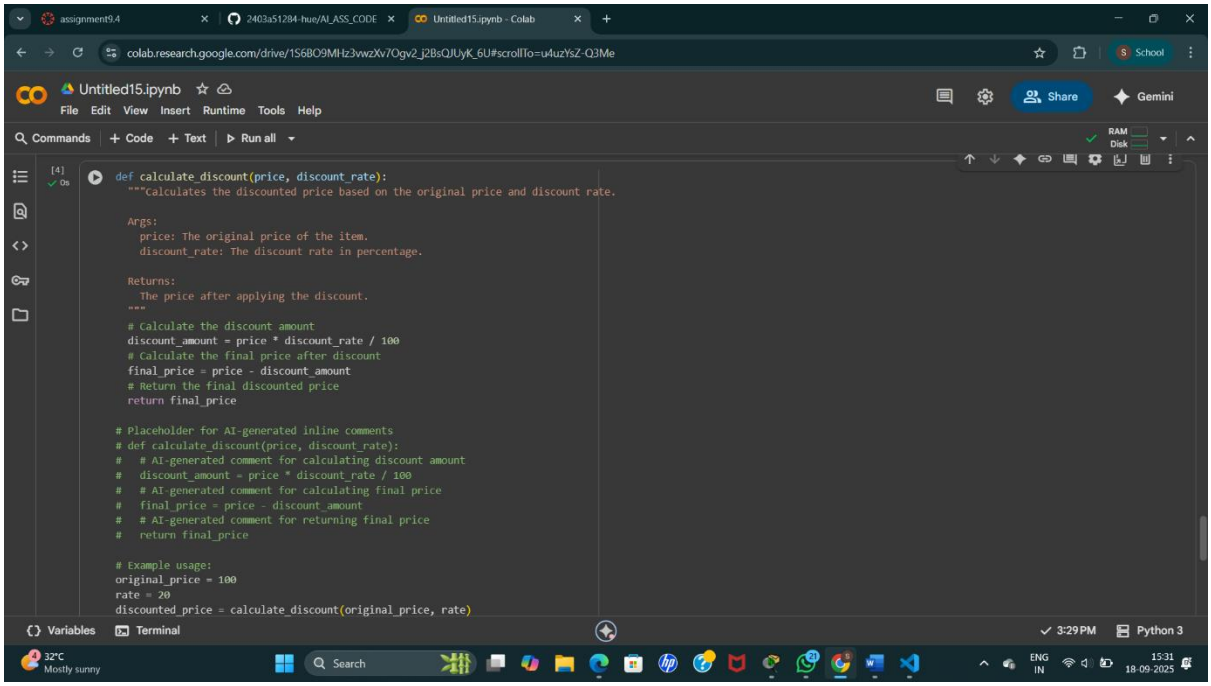
HTNO:2403A51284

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

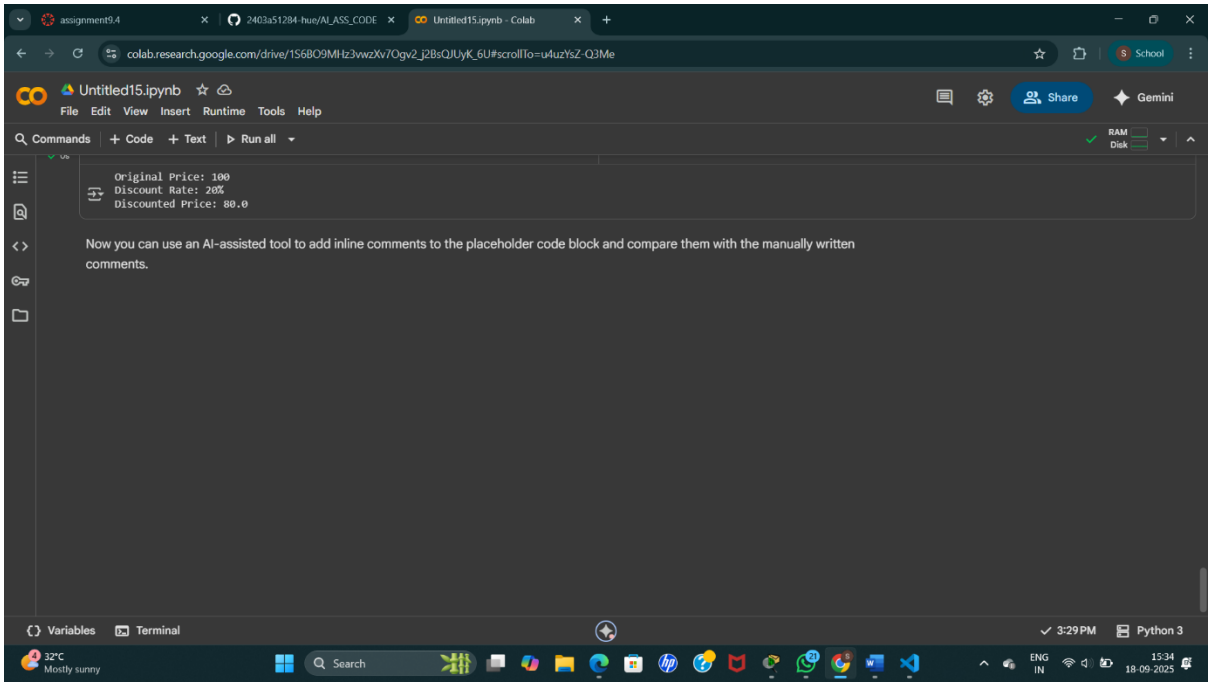
PROMPT:

Add line-by-line comments to the following Python function, explaining each step.
Then, modify the function to include a docstring in Google-style or NumPy-style format.
Compare the AI-generated comments with your manually written version for clarity and completeness.

CODE:



OUTPUT:



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:

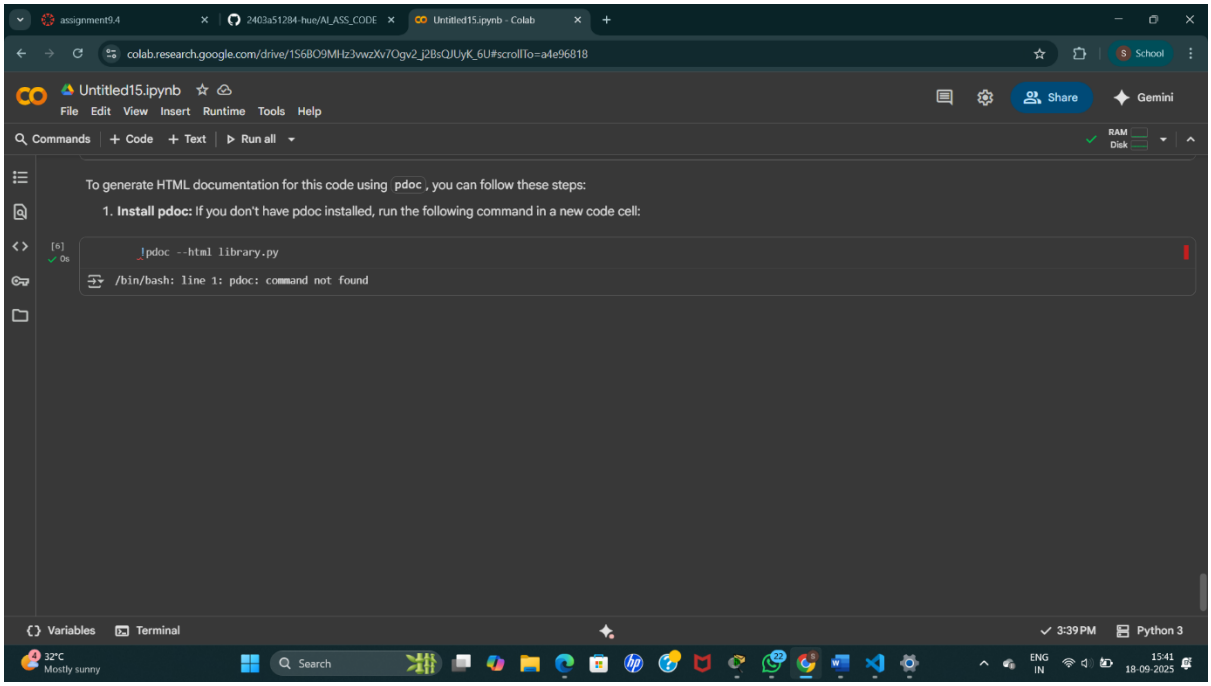
Add detailed docstrings to each function in the following Library Management System Python script.
Each docstring should include a description, input parameters, and output details.
Then, use a documentation generator tool (such as pdoc, Sphinx, or MkDocs) to automatically create HTML documentation from the code.
Submit both the annotated Python code and the generated HTML documentation as output.

CODE:

```python
def add_book(title, author, year):
  """Adds a new book to the library system.

  Args:
    title: The title of the book.
    author: The author of the book.
    year: The publication year of the book.

  Returns:
    A message confirming the book has been added.
  """
  # code to add book
  print(f"Book '{title}' by {author} ({year}) added.")

def issue_book(book_id, user_id):
  """Issues a book to a user.

  Args:
    book_id: The ID of the book to issue.
    user_id: The ID of the user borrowing the book.

  Returns:
    A message confirming the book has been issued to the user.
  """
  # code to issue book
  print(f"Book ID {book_id} issued to User ID {user_id}.")

# Example usage (optional, for testing the functions within the notebook)
# add_book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams", 1979)
```

```python
  """Adds a new book to the library system.

  Args:
    title: The title of the book.
    author: The author of the book.
    year: The publication year of the book.

  Returns:
    A message confirming the book has been added.
  """
  # code to add book
  print(f"Book '{title}' by {author} ({year}) added.")

def issue_book(book_id, user_id):
  """Issues a book to a user.

  Args:
    book_id: The ID of the book to issue.
    user_id: The ID of the user borrowing the book.

  Returns:
    A message confirming the book has been issued to the user.
  """
  # code to issue book
  print(f"Book ID {book_id} issued to User ID {user_id}.")

# Example usage (optional, for testing the functions within the notebook)
# add_book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams", 1979)
# issue_book(101, 505)
```

OUTPUT:



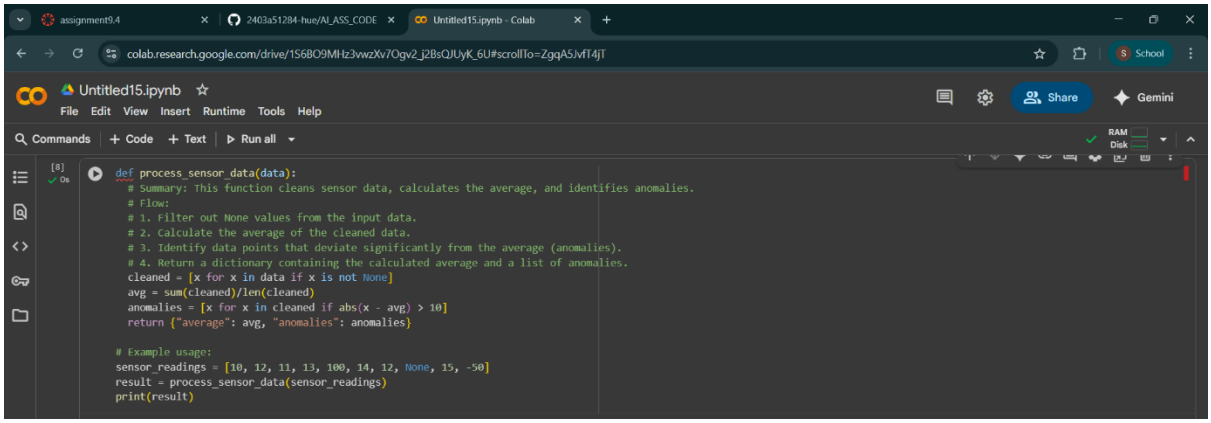
Task Description #3 (AI-Assisted Code Summarization)
Scenario: You are reviewing a colleague's codebase containing long functions. def process_sensor_data(data):
cleaned = [x for x in data if x is not None]
avg = sum(cleaned)/len(cleaned)
anomalies = [x for x in cleaned if abs(x - avg) > 10]
return {"average": 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:

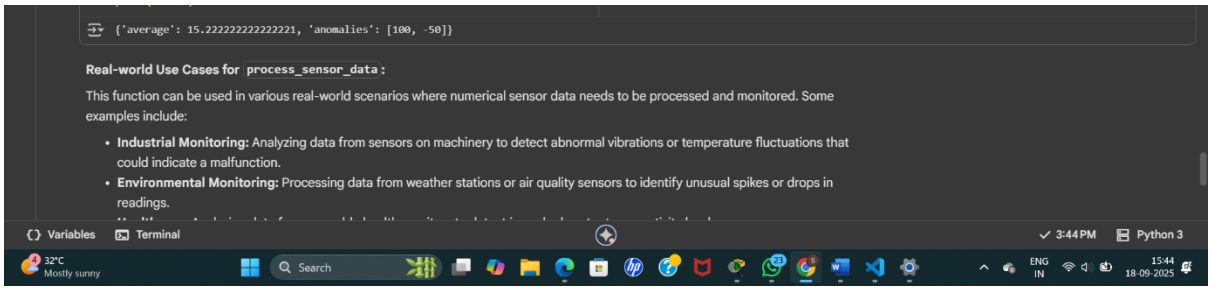
For the following Python function, generate a summary comment explaining its purpose in 2–3 lines.
Then, add a flow-style comment with a step-by-step explanation of the logic.
Finally, write a short paragraph of documentation describing possible real-world use cases for this function.

CODE:



```python
def process_sensor_data(data):
    # Summary: This function cleans sensor data, calculates the average, and identifies anomalies.
    # Flow:
    # 1. Filter out None values from the input data.
    # 2. Calculate the average of the cleaned data.
    # 3. Identify data points that deviate significantly from the average (anomalies).
    # 4. Return a dictionary containing the calculated average and a list of anomalies.
    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}

# Example usage:
sensor_readings = [10, 12, 11, 13, 100, 14, 12, None, 15, -50]
result = process_sensor_data(sensor_readings)
print(result)
```

OUTPUT:



```
{'average': 15.222222222222221, 'anomalies': [100, -50]}
```

Real-world Use Cases for `process_sensor_data`:

This function can be used in various real-world scenarios where numerical sensor data needs to be processed and monitored. Some examples include:

- **Industrial Monitoring:** Analyzing data from sensors on machinery to detect abnormal vibrations or temperature fluctuations that could indicate a malfunction.
- **Environmental Monitoring:** Processing data from weather stations or air quality sensors to identify unusual spikes or drops in readings.

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:

Write a README.md file for the following Chatbot Application, including project description, installation steps, usage instructions, and an example interaction.
Add inline comments to the main Python script, focusing on explaining the logic rather than trivial code.
Generate a plain English usage guide based on the code comments.
Reflect on how automated documentation supports real-time project maintainability compared to manual documentation.

CODE:

Reasoning: The subtask requires creating a Python script for a simple chatbot with specific functionalities like taking user input, responding based on conditions, having a default response, and looping until an exit command. A single code block can encompass all these steps.

```python
def simple_chatbot():
    """A simple chatbot that responds to user input."""
    print("Hello! I'm a simple chatbot. Type 'bye' to exit.")

    while True:
        user_input = input("You: ").lower()  # Get user input and convert to lowercase

        if "hello" in user_input or "hi" in user_input:
            print("Chatbot: Hi there!")
        elif "how are you" in user_input:
            print("Chatbot: I'm a chatbot, so I don't have feelings, but I'm ready to chat!")
        elif "what is your name" in user_input:
            print("Chatbot: I don't have a name. I'm just a simple chatbot.")
        elif "bye" in user_input or "quit" in user_input:
            print("Chatbot: Goodbye! Have a great day!")
            break  # Exit the loop
        else:
            print("Chatbot: I'm not sure how to respond to that. Can you please rephrase?")

# Start the chatbot
simple_chatbot()
```

OUTPUT:

```
Hello! I'm a simple chatbot. Type 'bye' to exit.
You: SAM
Chatbot: I'm not sure how to respond to that. Can you please rephrase?
You: bye
Chatbot: Goodbye! Have a great day!
```