# Assignment 9.4 – Lab 9: Documentation Generation

* Name – G.Sanjansah ,
* Date –
* Subject – AI Assisted Coding
* Hall Ticket Number – 2503a52l20
* Student Mail Id – 2503a52l20@sru.edu.in

## Task 1: Automatic Code Commenting

Prompt given to AI: Generate line-by-line comments and add a Google-style docstring for this function.

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

### AI-Generated Version (with comments)

def calculate\_discount(price, discount\_rate):  
 # Subtracts the discount amount from the original price  
 # discount is calculated as (price \* discount\_rate / 100)  
 return price - (price \* discount\_rate / 100)

With Google-style Docstring:

def calculate\_discount(price, discount\_rate):  
 """  
 Calculate the final price after applying discount.  
  
 Args:  
 price (float): Original price of the product.  
 discount\_rate (float): Discount percentage.  
  
 Returns:  
 float: Final price after discount deduction.  
 """  
 return price - (price \* discount\_rate / 100)

### My Manual Version

def calculate\_discount(price, discount\_rate):  
 # price → actual price of item  
 # discount\_rate → discount percentage (e.g., 10 for 10%)  
 # final price = price - discount  
 return price - (price \* discount\_rate / 100)

### Comparison

- AI Comments: Concise but slightly generic.  
- Manual Comments: More descriptive about parameters and formula.  
- Docstring: Provides professional-level structured documentation.

## Task 2: API Documentation Generator

Prompt: Add docstrings to Library Management functions and generate HTML documentation using Sphinx/pdoc.

def add\_book(title, author, year):  
 """  
 Add a new book to the library.  
  
 Args:  
 title (str): Title of the book.  
 author (str): Author name.  
 year (int): Year of publication.  
  
 Returns:  
 dict: Confirmation with book details.  
 """  
 # code to add book  
 pass  
  
  
def issue\_book(book\_id, user\_id):  
 """  
 Issue a book to a library user.  
  
 Args:  
 book\_id (int): Unique identifier of the book.  
 user\_id (int): Unique identifier of the user.  
  
 Returns:  
 str: Success or failure message.  
 """  
 # code to issue book  
 pass

### Output (Generated Documentation Example)

Functions:  
 add\_book(title, author, year)  
 Add a new book to the library.  
 Parameters: title, author, year  
 Returns: dict  
  
 issue\_book(book\_id, user\_id)  
 Issue a book to a library user.  
 Parameters: book\_id, user\_id  
 Returns: str

## Task 3: AI-Assisted Code Summarization

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}

### Summary Comment (2–3 lines)

This function processes raw sensor data by removing invalid values, calculating the average, and identifying anomalies that deviate significantly from the average. It returns both the average and the anomalies.

### Flow-Style Comment

def process\_sensor\_data(data):  
 # 1. Remove None values from data  
 cleaned = [x for x in data if x is not None]  
  
 # 2. Calculate average of cleaned values  
 avg = sum(cleaned) / len(cleaned)  
  
 # 3. Detect anomalies (values differing by >10 from average)  
 anomalies = [x for x in cleaned if abs(x - avg) > 10]  
  
 # 4. Return dictionary with average and anomalies  
 return {"average": avg, "anomalies": anomalies}

### Use Cases

- Monitoring IoT devices where faulty sensor readings need filtering.  
- Industrial automation systems to detect abnormal readings.  
- Health data monitoring (e.g., detecting sudden spikes in heart rate).

## Task 4: Real-Time Project Documentation

### README.md (for Chatbot Project)

# Chatbot Application  
  
## Description  
A conversational chatbot that responds to user queries using AI/NLP techniques.  
  
## Installation  
1. Clone repository  
2. Install dependencies: `pip install -r requirements.txt`  
3. Run: `python main.py`  
  
## Usage  
- Start chatbot using terminal  
- Type your queries and receive responses  
  
## Example  
User: Hello   
Bot: Hi! How can I help you today?

### Inline Comments Example

def chatbot\_response(user\_input):  
 # Step 1: Preprocess user input (remove special chars, lowercase)  
 processed = preprocess(user\_input)  
  
 # Step 2: Match input with trained model/patterns  
 response = model.get\_response(processed)  
  
 # Step 3: Return chatbot’s response  
 return response

### AI-Assisted Usage Guide (Generated)

The chatbot takes user input, processes it into a clean format, matches it with stored responses, and returns a conversational reply. Run the program, type messages, and interact as you would in a normal chat.

### Reflection

- Manual Documentation: More detailed, but time-consuming and inconsistent across team members.  
- Automated Documentation: Faster, standardized, easily updated from code.  
- Best Practice: Use a hybrid approach → Auto-generate base docs, then refine manually.