AI ASSISTED CODING

ASSIGNMENT – 9.5

MD ZIAUDDIN

2403A51271

BATCH:12

Task-1: (Automatic Code Commenting)

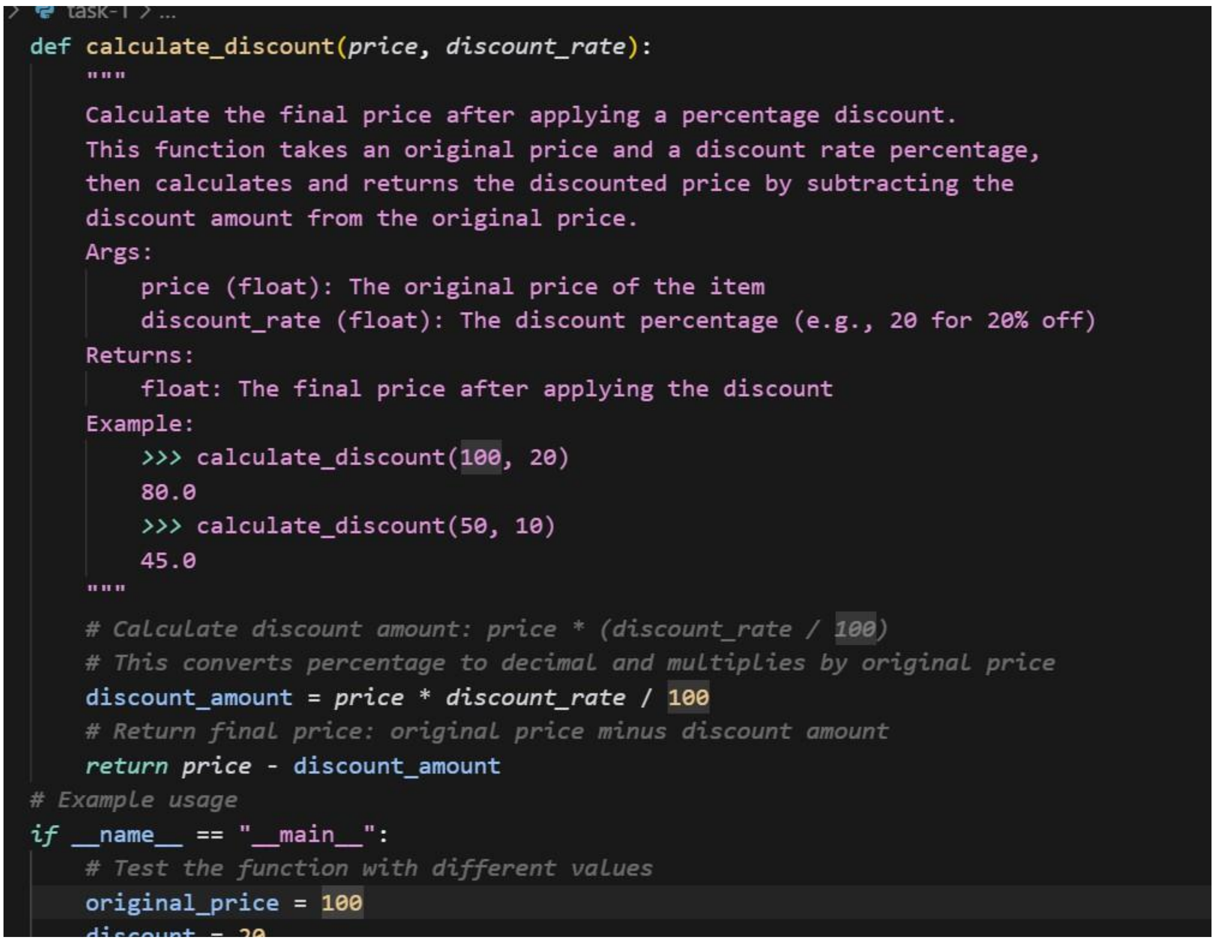
Scenario: You have been given a Python function without comments.
def calculate_discount(price, discount_rate):
return price - (price * discount_rate / 100)
• Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.
• Modify the function so that it includes a docstring in Google-style or NumPy-style format.
• Compare the auto-generated comments with your manually written version

Prompt: Given this Python function:

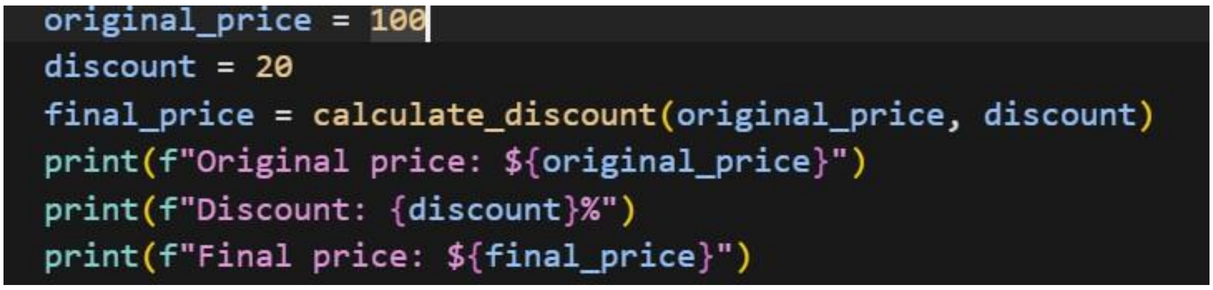
```
def calculate_discount (price, discount_rate):

    return price - (price * discount_rate / 100)
```

1. Add **line-by-line comments** explaining each line.
2. Rewrite the function with a **Google-style docstring** (include description, parameters, return, example).
3. Compare AI-generated comments with your manually written comments.

Code:

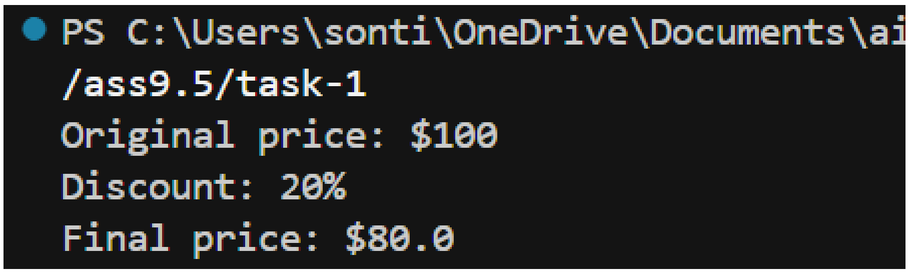


```
> task-1 > ...
def calculate_discount(price, discount_rate):
    """
    Calculate the final price after applying a percentage discount.
    This function takes an original price and a discount rate percentage,
    then calculates and returns the discounted price by subtracting the
    discount amount from the original price.
    Args:
        price (float): The original price of the item
        discount_rate (float): The discount percentage (e.g., 20 for 20% off)
    Returns:
        float: The final price after applying the discount
    Example:
        >>> calculate_discount(100, 20)
        80.0
        >>> calculate_discount(50, 10)
        45.0
    """
    # Calculate discount amount: price * (discount_rate / 100)
    # This converts percentage to decimal and multiplies by original price
    discount_amount = price * discount_rate / 100
    # Return final price: original price minus discount amount
    return price - discount_amount
# Example usage
if __name__ == "__main__":
    # Test the function with different values
    original_price = 100
    discount = 20
```



```
    original_price = 100
    discount = 20
    final_price = calculate_discount(original_price, discount)
    print(f"Original price: ${original_price}")
    print(f"Discount: {discount}%")
    print(f"Final price: ${final_price}")
```

Output:



```
PS C:\Users\sonti\OneDrive\Documents\ai
/ass9.5/task-1
Original price: $100
Discount: 20%
Final price: $80.0
```

Task-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
```

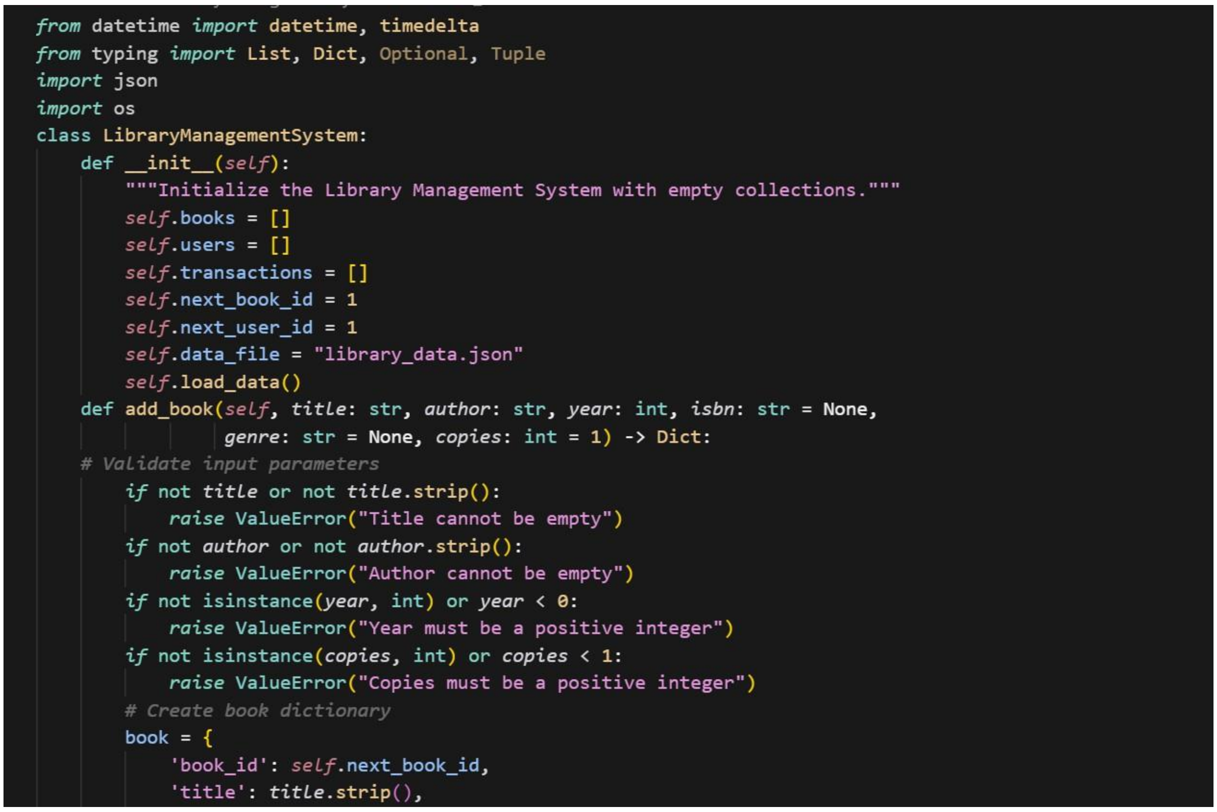
• Write a Python script that uses docstrings for each function (with input, output, and description).
• Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.
• Submit both the code and the generated documentation as output.

Prompt: Given these functions:

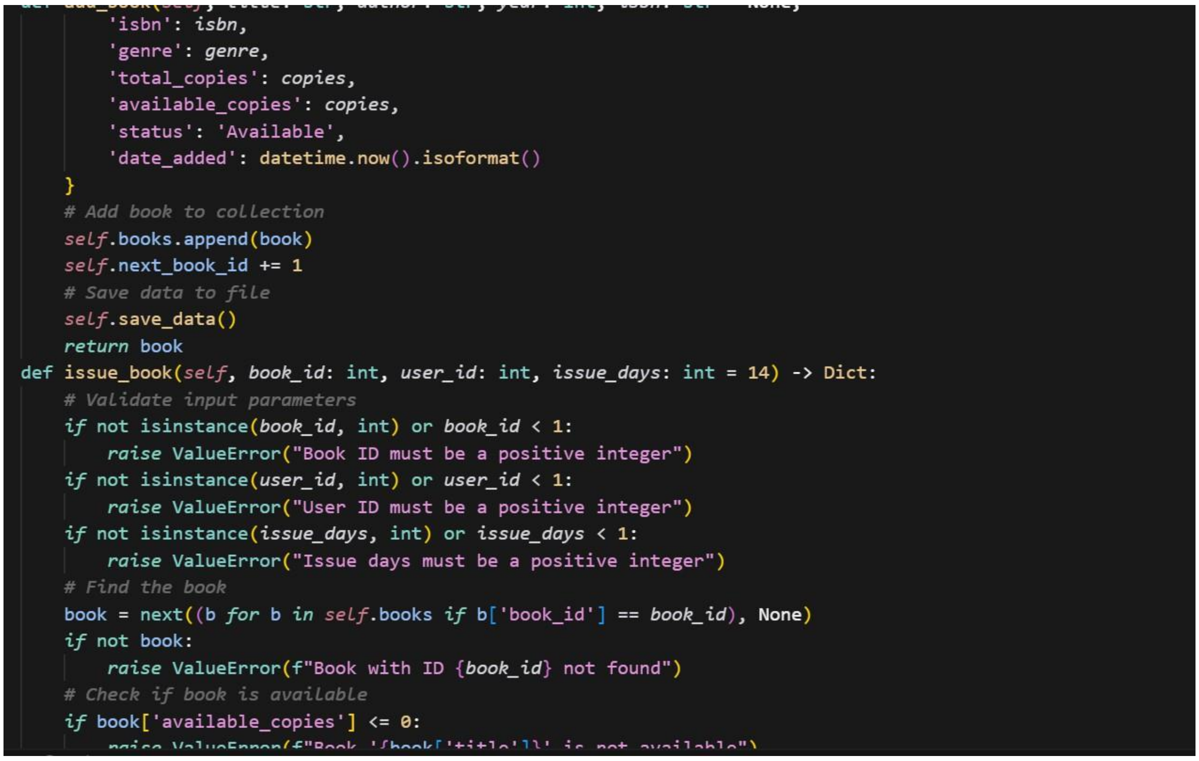
```
def add_book (title, author, year):
    pass
def issue_book (book_id, user_id):
    pass
```

1. Add **docstrings** with description, inputs, outputs, and example.
2. Generate **HTML documentation** using pdoc, Sphinx, or MkDocs.
3. Submit the Python script and generated docs.

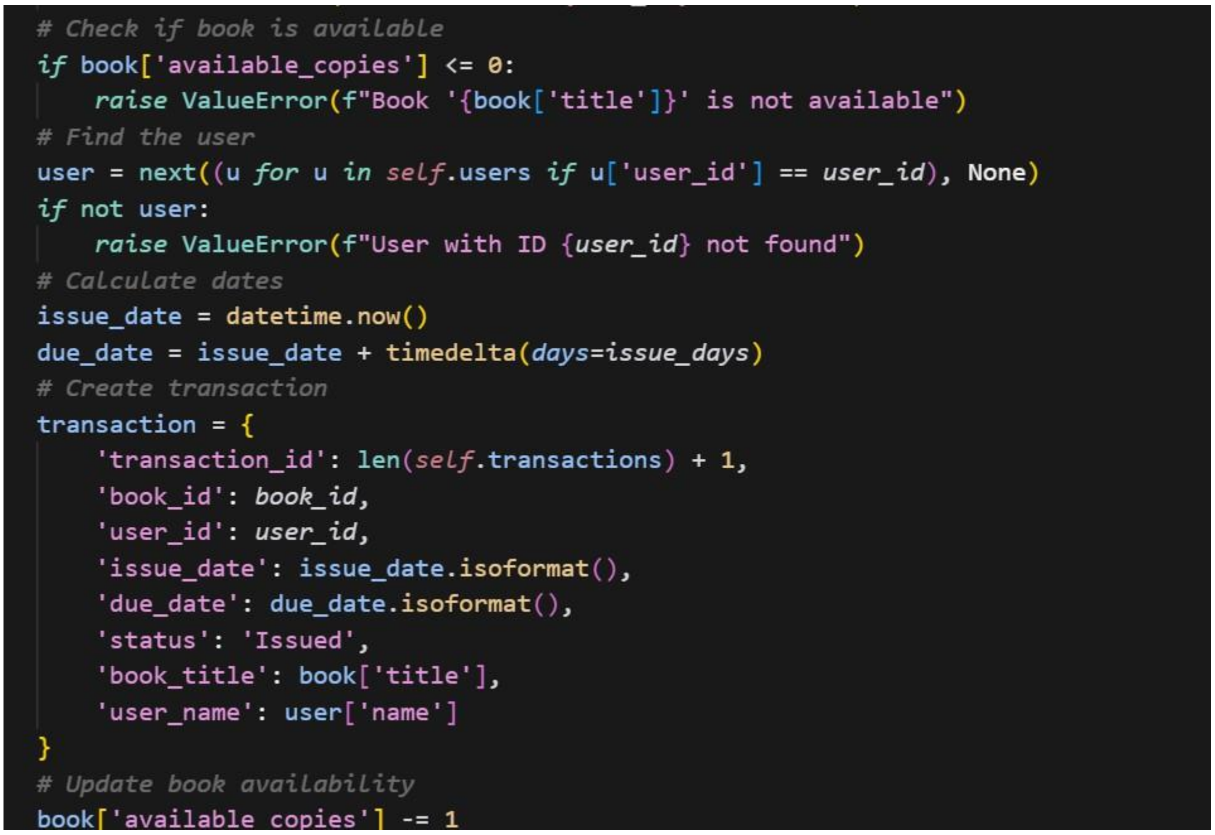
Code:



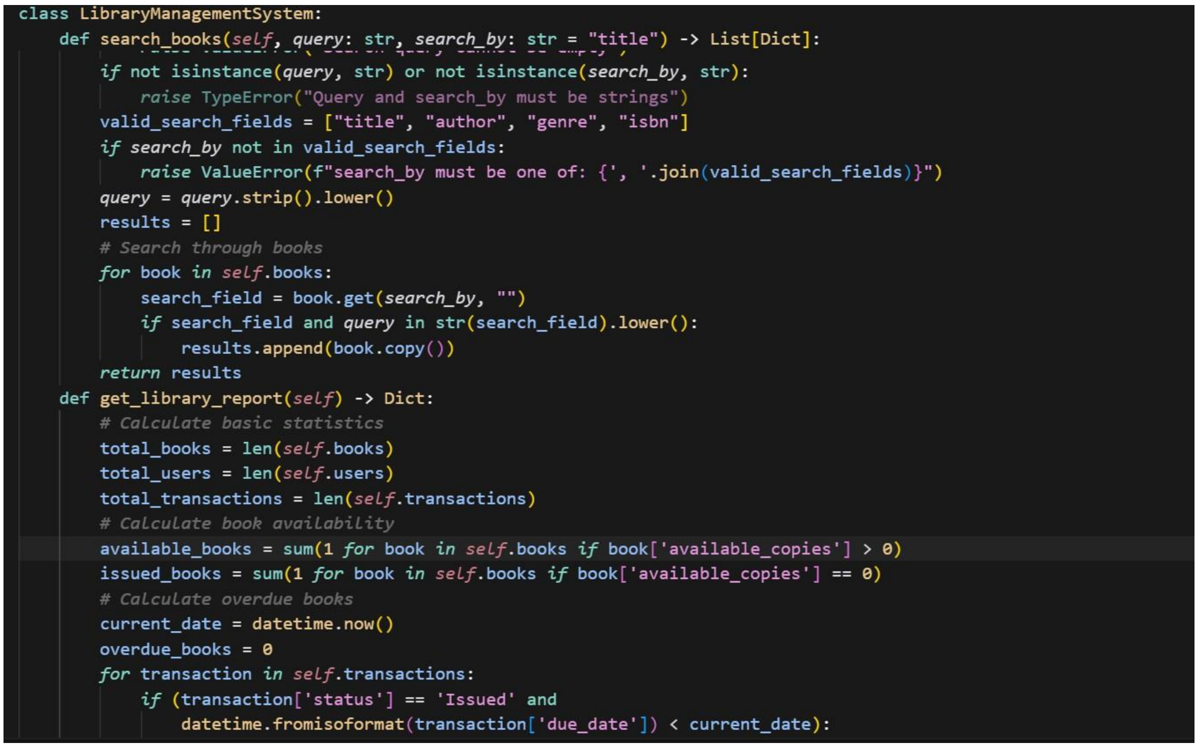
```python
from datetime import datetime, timedelta
from typing import List, Dict, Optional, Tuple
import json
import os
class LibraryManagementSystem:
    def __init__(self):
        """Initialize the Library Management System with empty collections."""
        self.books = []
        self.users = []
        self.transactions = []
        self.next_book_id = 1
        self.next_user_id = 1
        self.data_file = "library_data.json"
        self.load_data()
    def add_book(self, title: str, author: str, year: int, isbn: str = None,
                 genre: str = None, copies: int = 1) -> Dict:
    # Validate input parameters
        if not title or not title.strip():
            raise ValueError("Title cannot be empty")
        if not author or not author.strip():
            raise ValueError("Author cannot be empty")
        if not isinstance(year, int) or year < 0:
            raise ValueError("Year must be a positive integer")
        if not isinstance(copies, int) or copies < 1:
            raise ValueError("Copies must be a positive integer")
        # Create book dictionary
        book = {
            'book_id': self.next_book_id,
            'title': title.strip(),
```



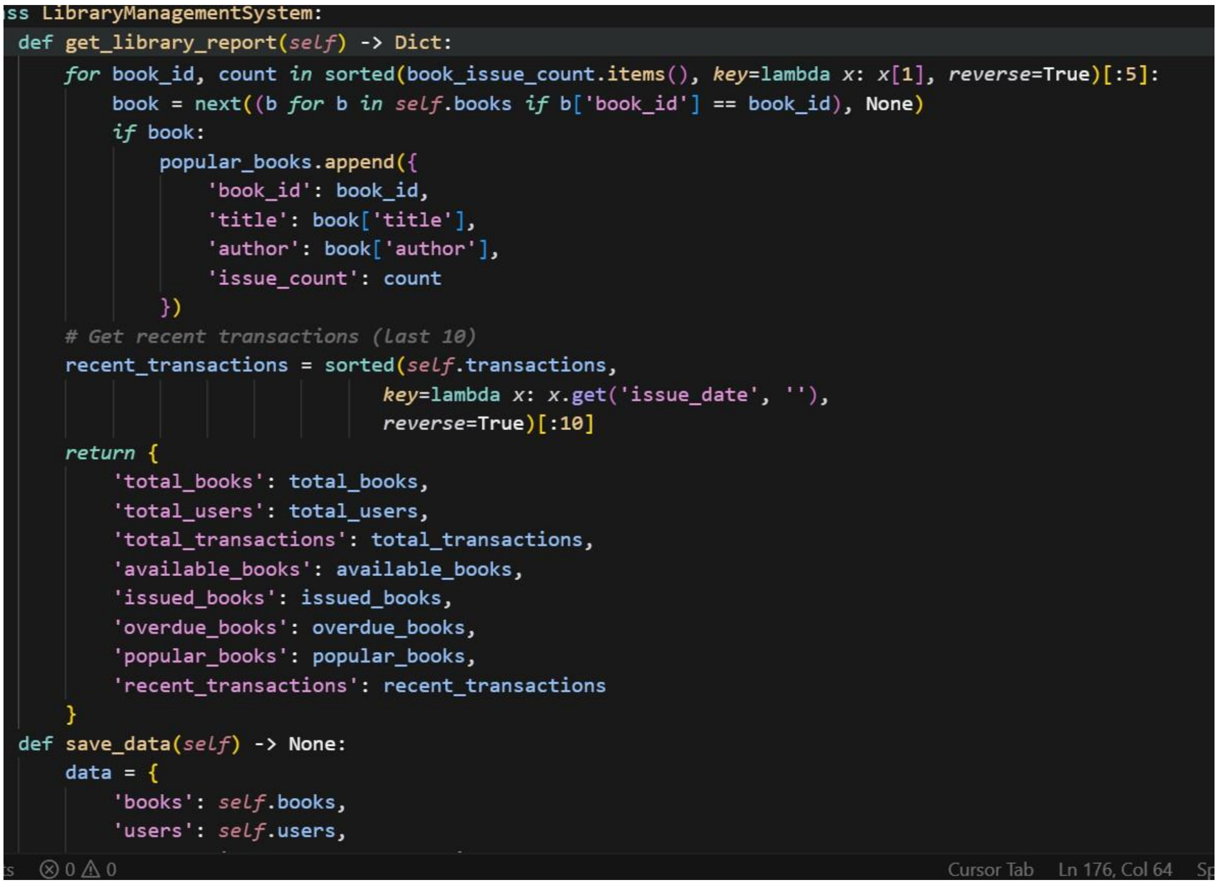
```python
            'isbn': isbn,
            'genre': genre,
            'total_copies': copies,
            'available_copies': copies,
            'status': 'Available',
            'date_added': datetime.now().isoformat()
        }
        # Add book to collection
        self.books.append(book)
        self.next_book_id += 1
        # Save data to file
        self.save_data()
        return book
    def issue_book(self, book_id: int, user_id: int, issue_days: int = 14) -> Dict:
        # Validate input parameters
        if not isinstance(book_id, int) or book_id < 1:
            raise ValueError("Book ID must be a positive integer")
        if not isinstance(user_id, int) or user_id < 1:
            raise ValueError("User ID must be a positive integer")
        if not isinstance(issue_days, int) or issue_days < 1:
            raise ValueError("Issue days must be a positive integer")
        # Find the book
        book = next((b for b in self.books if b['book_id'] == book_id), None)
        if not book:
            raise ValueError(f"Book with ID {book_id} not found")
        # Check if book is available
        if book['available_copies'] <= 0:
            raise ValueError(f"Book '{book['title']}' is not available")
```



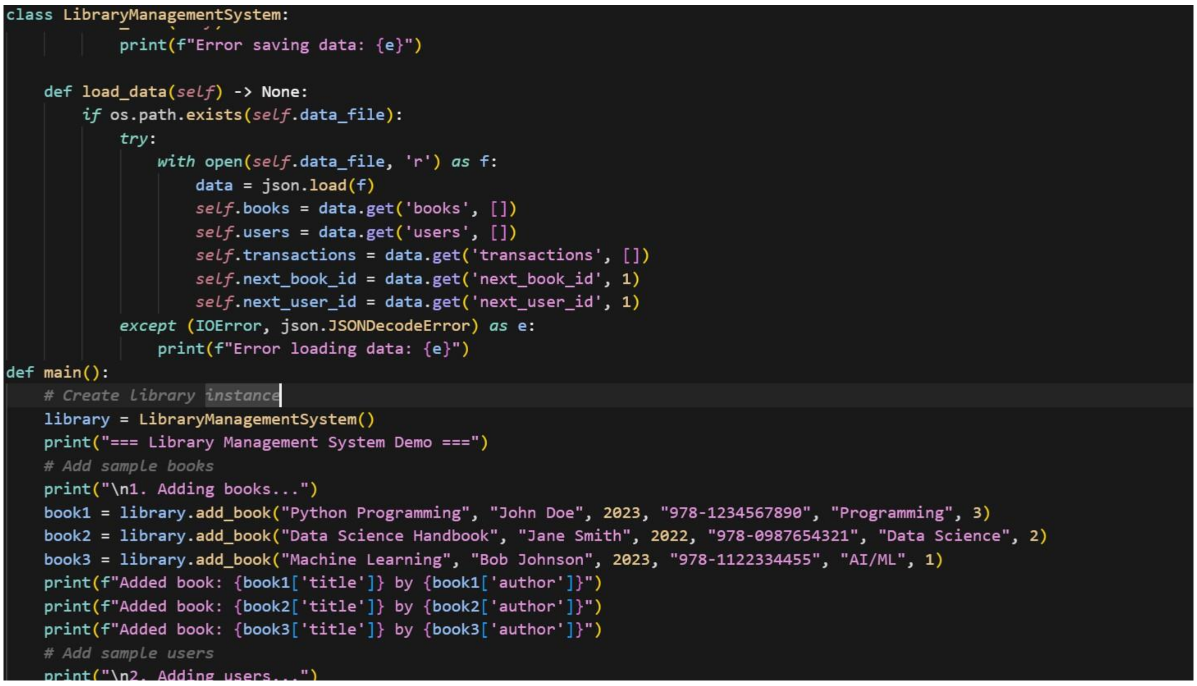
```python
# Check if book is available
if book['available_copies'] <= 0:
    raise ValueError(f"Book '{book['title']}' is not available")
# Find the user
user = next((u for u in self.users if u['user_id'] == user_id), None)
if not user:
    raise ValueError(f"User with ID {user_id} not found")
# Calculate dates
issue_date = datetime.now()
due_date = issue_date + timedelta(days=issue_days)
# Create transaction
transaction = {
    'transaction_id': len(self.transactions) + 1,
    'book_id': book_id,
    'user_id': user_id,
    'issue_date': issue_date.isoformat(),
    'due_date': due_date.isoformat(),
    'status': 'Issued',
    'book_title': book['title'],
    'user_name': user['name']
}
# Update book availability
book['available_copies'] -= 1
```



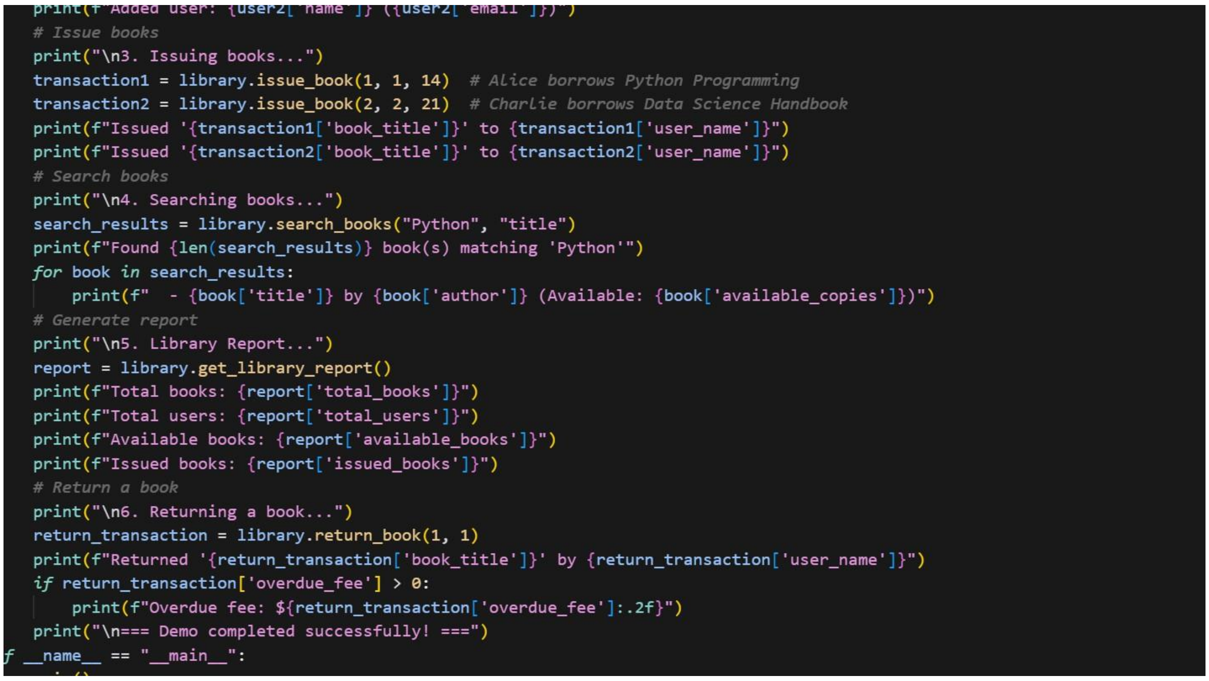
```python
class LibraryManagementSystem:
    def search_books(self, query: str, search_by: str = "title") -> List[Dict]:
        if not isinstance(query, str) or not isinstance(search_by, str):
            raise TypeError("Query and search_by must be strings")
        valid_search_fields = ["title", "author", "genre", "isbn"]
        if search_by not in valid_search_fields:
            raise ValueError(f"search_by must be one of: {', '.join(valid_search_fields)}")
        query = query.strip().lower()
        results = []
        # Search through books
        for book in self.books:
            search_field = book.get(search_by, "")
            if search_field and query in str(search_field).lower():
                results.append(book.copy())
        return results
    def get_library_report(self) -> Dict:
        # Calculate basic statistics
        total_books = len(self.books)
        total_users = len(self.users)
        total_transactions = len(self.transactions)
        # Calculate book availability
        available_books = sum(1 for book in self.books if book['available_copies'] > 0)
        issued_books = sum(1 for book in self.books if book['available_copies'] == 0)
        # Calculate overdue books
        current_date = datetime.now()
        overdue_books = 0
        for transaction in self.transactions:
            if (transaction['status'] == 'Issued' and
                datetime.fromisoformat(transaction['due_date']) < current_date):
```



```python
ss LibraryManagementSystem:
    def get_library_report(self) -> Dict:
        for book_id, count in sorted(book_issue_count.items(), key=lambda x: x[1], reverse=True)[:5]:
            book = next((b for b in self.books if b['book_id'] == book_id), None)
            if book:
                popular_books.append({
                    'book_id': book_id,
                    'title': book['title'],
                    'author': book['author'],
                    'issue_count': count
                })
        # Get recent transactions (last 10)
        recent_transactions = sorted(self.transactions,
                                   key=lambda x: x.get('issue_date', ''),
                                   reverse=True)[:10]
        return {
            'total_books': total_books,
            'total_users': total_users,
            'total_transactions': total_transactions,
            'available_books': available_books,
            'issued_books': issued_books,
            'overdue_books': overdue_books,
            'popular_books': popular_books,
            'recent_transactions': recent_transactions
        }
    def save_data(self) -> None:
        data = {
            'books': self.books,
            'users': self.users,
```

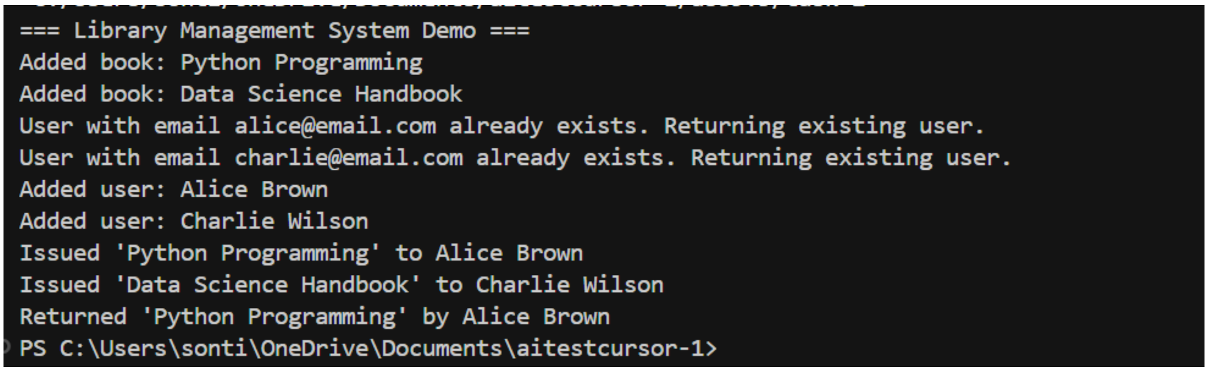


```python
class LibraryManagementSystem:
                print(f"Error saving data: {e}")

    def load_data(self) -> None:
        if os.path.exists(self.data_file):
            try:
                with open(self.data_file, 'r') as f:
                    data = json.load(f)
                    self.books = data.get('books', [])
                    self.users = data.get('users', [])
                    self.transactions = data.get('transactions', [])
                    self.next_book_id = data.get('next_book_id', 1)
                    self.next_user_id = data.get('next_user_id', 1)
            except (IOError, json.JSONDecodeError) as e:
                print(f"Error loading data: {e}")
def main():
    # Create library instance
    library = LibraryManagementSystem()
    print("=== Library Management System Demo ===")
    # Add sample books
    print("\n1. Adding books...")
    book1 = library.add_book("Python Programming", "John Doe", 2023, "978-1234567890", "Programming", 3)
    book2 = library.add_book("Data Science Handbook", "Jane Smith", 2022, "978-0987654321", "Data Science", 2)
    book3 = library.add_book("Machine Learning", "Bob Johnson", 2023, "978-1122334455", "AI/ML", 1)
    print(f"Added book: {book1['title']} by {book1['author']}")
    print(f"Added book: {book2['title']} by {book2['author']}")
    print(f"Added book: {book3['title']} by {book3['author']}")
    # Add sample users
    print("\n2. Adding users...")
```



```python
    print(f"Added user: {user2['name']} ({user2['email']})")
    # Issue books
    print("\n3. Issuing books...")
    transaction1 = library.issue_book(1, 1, 14)  # Alice borrows Python Programming
    transaction2 = library.issue_book(2, 2, 21)  # Charlie borrows Data Science Handbook
    print(f"Issued '{transaction1['book_title']}' to {transaction1['user_name']}")
    print(f"Issued '{transaction2['book_title']}' to {transaction2['user_name']}")
    # Search books
    print("\n4. Searching books...")
    search_results = library.search_books("Python", "title")
    print(f"Found {len(search_results)} book(s) matching 'Python'")
    for book in search_results:
        print(f"  - {book['title']} by {book['author']} (Available: {book['available_copies']})")
    # Generate report
    print("\n5. Library Report...")
    report = library.get_library_report()
    print(f"Total books: {report['total_books']}")
    print(f"Total users: {report['total_users']}")
    print(f"Available books: {report['available_books']}")
    print(f"Issued books: {report['issued_books']}")
    # Return a book
    print("\n6. Returning a book...")
    return_transaction = library.return_book(1, 1)
    print(f"Returned '{return_transaction['book_title']}' by {return_transaction['user_name']}")
    if return_transaction['overdue_fee'] > 0:
        print(f"Overdue fee: ${return_transaction['overdue_fee']:.2f}")
    print("\n=== Demo completed successfully! ===")
f __name__ == "__main__":
```

Output:



```
=== Library Management System Demo ===
Added book: Python Programming
Added book: Data Science Handbook
User with email alice@email.com already exists. Returning existing user.
User with email charlie@email.com already exists. Returning existing user.
Added user: Alice Brown
Added user: Charlie Wilson
Issued 'Python Programming' to Alice Brown
Issued 'Data Science Handbook' to Charlie Wilson
Returned 'Python Programming' by Alice Brown
PS C:\Users\sonti\OneDrive\Documents\aitestcursor-1>
```

Task-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: Given this function:

```
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}
```

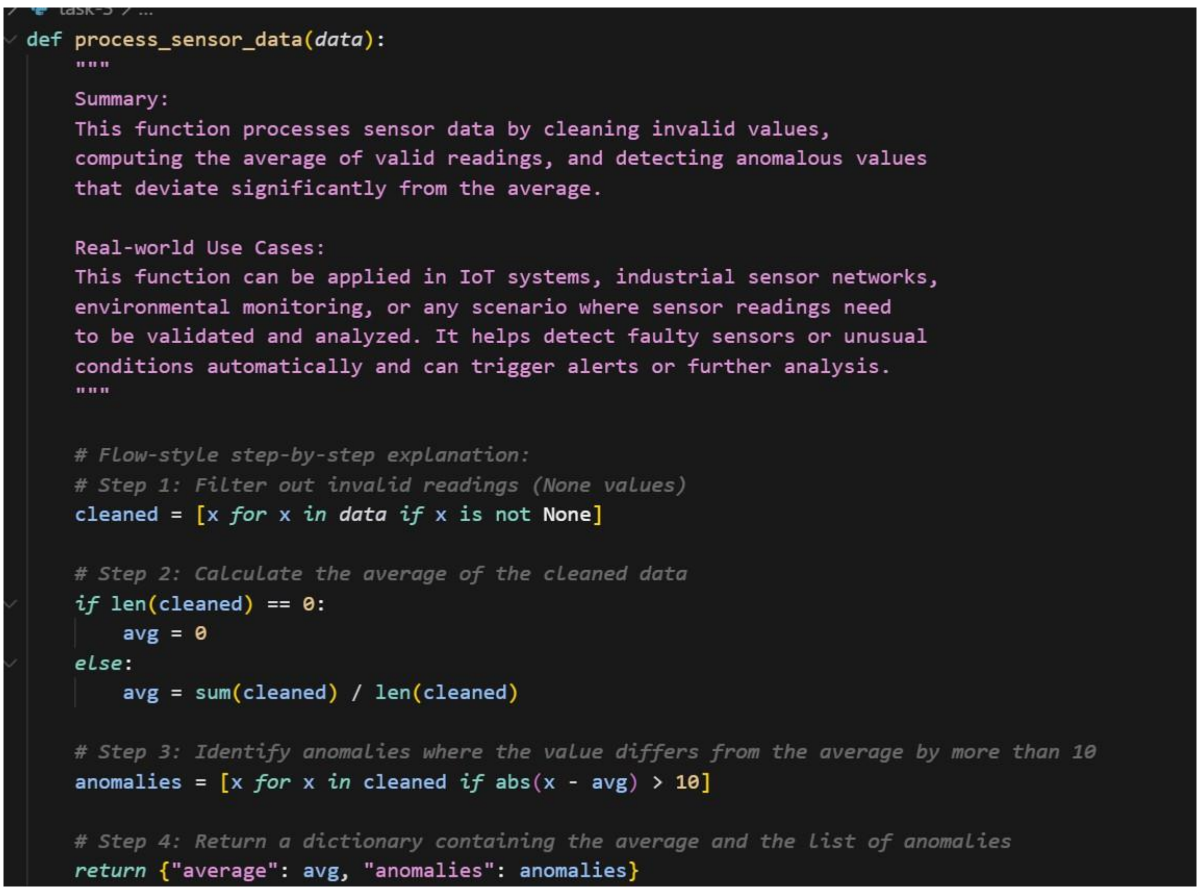
Write:

1. A 2–3-line summary comment.

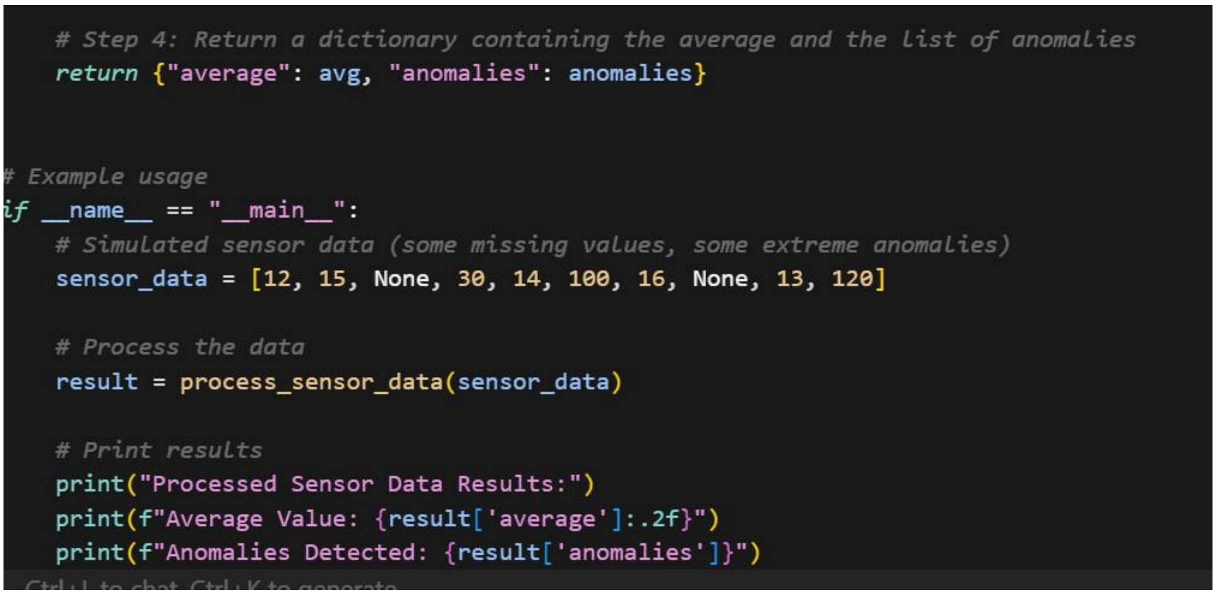
2. Step-by-step flow-style comments.

3. A short paragraph on real-world use cases.

Code:

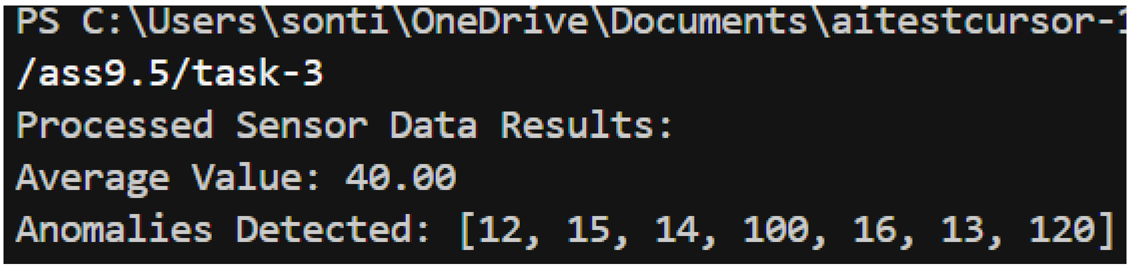


```python
def process_sensor_data(data):
    """
    Summary:
    This function processes sensor data by cleaning invalid values,
    computing the average of valid readings, and detecting anomalous values
    that deviate significantly from the average.

    Real-world Use Cases:
    This function can be applied in IoT systems, industrial sensor networks,
    environmental monitoring, or any scenario where sensor readings need
    to be validated and analyzed. It helps detect faulty sensors or unusual
    conditions automatically and can trigger alerts or further analysis.
    """

    # Flow-style step-by-step explanation:
    # Step 1: Filter out invalid readings (None values)
    cleaned = [x for x in data if x is not None]

    # Step 2: Calculate the average of the cleaned data
    if len(cleaned) == 0:
        avg = 0
    else:
        avg = sum(cleaned) / len(cleaned)

    # Step 3: Identify anomalies where the value differs from the average by more than 10
    anomalies = [x for x in cleaned if abs(x - avg) > 10]

    # Step 4: Return a dictionary containing the average and the list of anomalies
    return {"average": avg, "anomalies": anomalies}
```



```python
    # Step 4: Return a dictionary containing the average and the list of anomalies
    return {"average": avg, "anomalies": anomalies}


# Example usage
if __name__ == "__main__":
    # Simulated sensor data (some missing values, some extreme anomalies)
    sensor_data = [12, 15, None, 30, 14, 100, 16, None, 13, 120]

    # Process the data
    result = process_sensor_data(sensor_data)

    # Print results
    print("Processed Sensor Data Results:")
    print(f"Average Value: {result['average']:.2f}")
    print(f"Anomalies Detected: {result['anomalies']}")
```

Output:



```
PS C:\Users\sonti\OneDrive\Documents\aitestcursor-
/ass9.5/task-3
Processed Sensor Data Results:
Average Value: 40.00
Anomalies Detected: [12, 15, 14, 100, 16, 13, 120]
```

Task-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: You are an AI Python documentation assistant.

1. Add Google-style docstrings to all functions in this Python script.

2. Add inline comments explaining the logic.

3. Generate a simple plain-English usage guide.

Python Script:

<PASTE YOUR PYTHON CODE HERE>

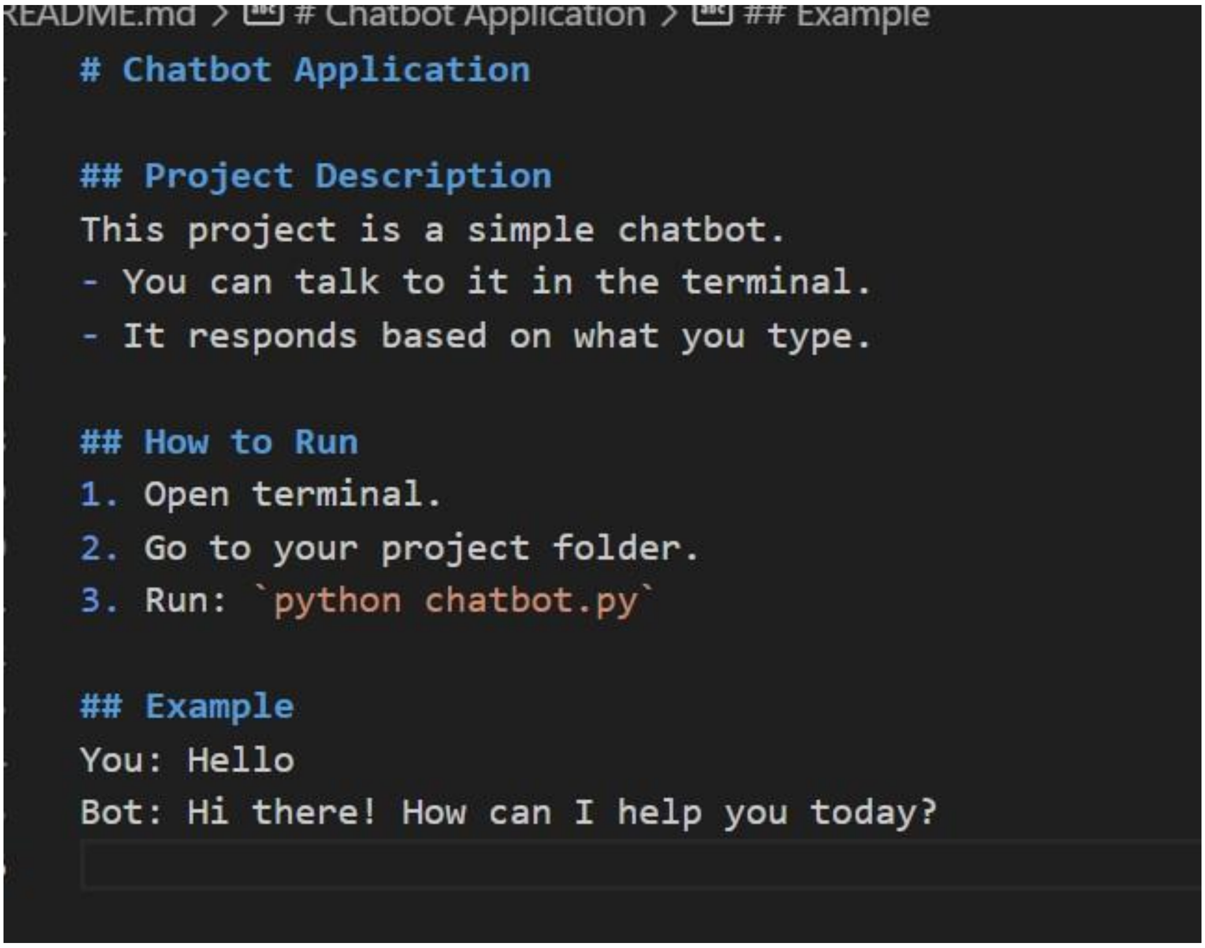
Output:

- Python code with docstrings and comments.

- Plain-English usage guide.

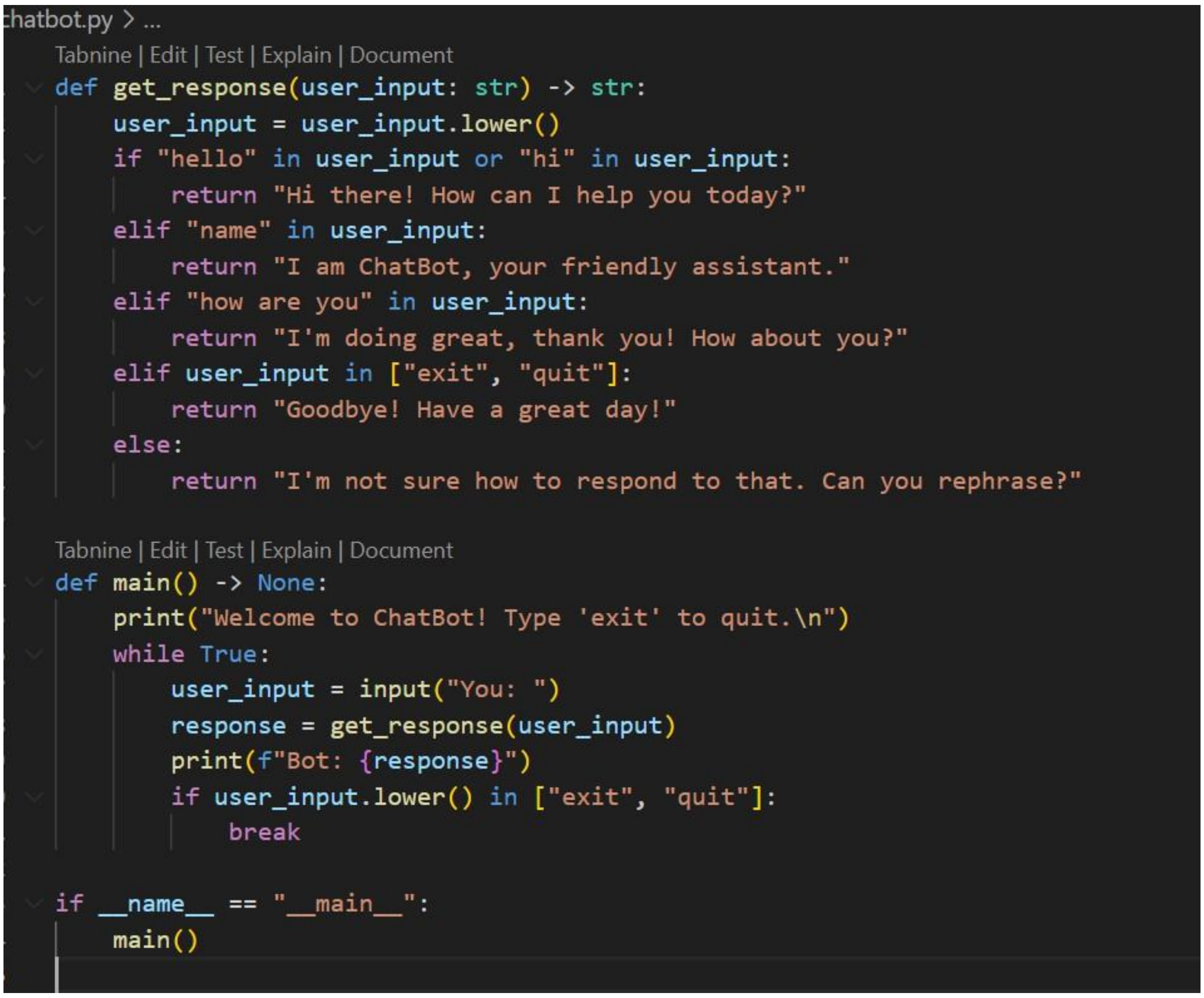
Code:

Readme code:



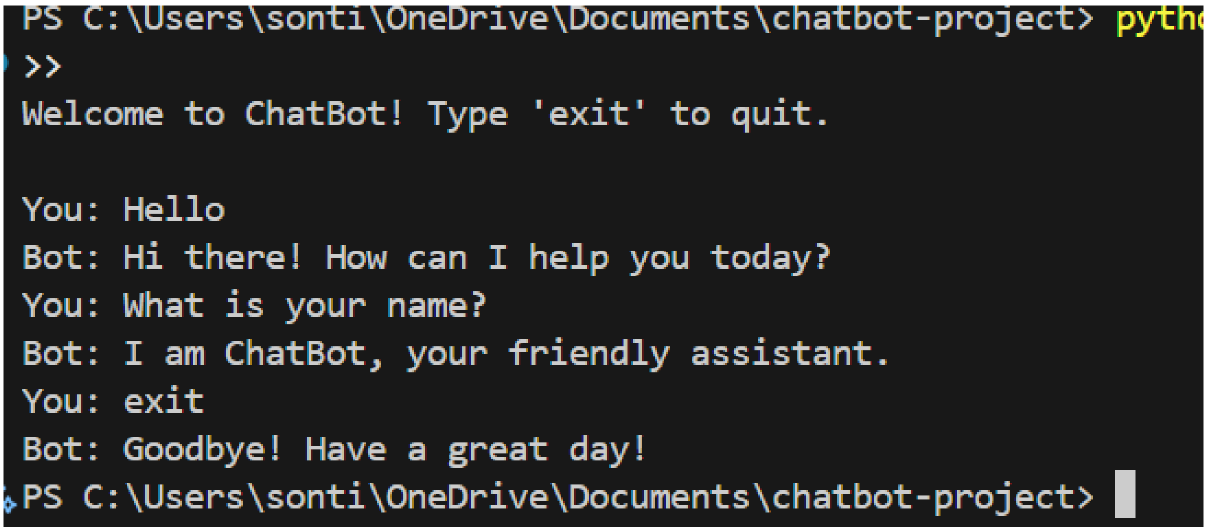
```
# Chatbot Application

## Project Description
This project is a simple chatbot.
- You can talk to it in the terminal.
- It responds based on what you type.

## How to Run
1. Open terminal.
2. Go to your project folder.
3. Run: `python chatbot.py`

## Example
You: Hello
Bot: Hi there! How can I help you today?
```

Chatbot code:



```
hatbot.py > ...
    Tabnine | Edit | Test | Explain | Document
    def get_response(user_input: str) -> str:
        user_input = user_input.lower()
        if "hello" in user_input or "hi" in user_input:
            return "Hi there! How can I help you today?"
        elif "name" in user_input:
            return "I am ChatBot, your friendly assistant."
        elif "how are you" in user_input:
            return "I'm doing great, thank you! How about you?"
        elif user_input in ["exit", "quit"]:
            return "Goodbye! Have a great day!"
        else:
            return "I'm not sure how to respond to that. Can you rephrase?"

    Tabnine | Edit | Test | Explain | Document
    def main() -> None:
        print("Welcome to ChatBot! Type 'exit' to quit.\n")
        while True:
            user_input = input("You: ")
            response = get_response(user_input)
            print(f"Bot: {response}")
            if user_input.lower() in ["exit", "quit"]:
                break

    if __name__ == "__main__":
        main()
```

Output:



```
PS C:\Users\sonti\OneDrive\Documents\chatbot-project> pytho
>>
Welcome to ChatBot! Type 'exit' to quit.

You: Hello
Bot: Hi there! How can I help you today?
You: What is your name?
Bot: I am ChatBot, your friendly assistant.
You: exit
Bot: Goodbye! Have a great day!
PS C:\Users\sonti\OneDrive\Documents\chatbot-project>
```