Lab Assignment-9.3

Goli Harini Reddy

2303A51544

Batch:08

Task 1: Basic Docstring Generation

Scenario

You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.

Requirements

• Write a Python function to return the sum of even numbers and sum of odd numbers in a given list

• Manually add a Google Style docstring to the function

• Use an AI-assisted tool (Copilot / Cursor AI) to generate a function-level docstring

• Compare the AI-generated docstring with the manually written docstring

• Analyze clarity, correctness, and completeness

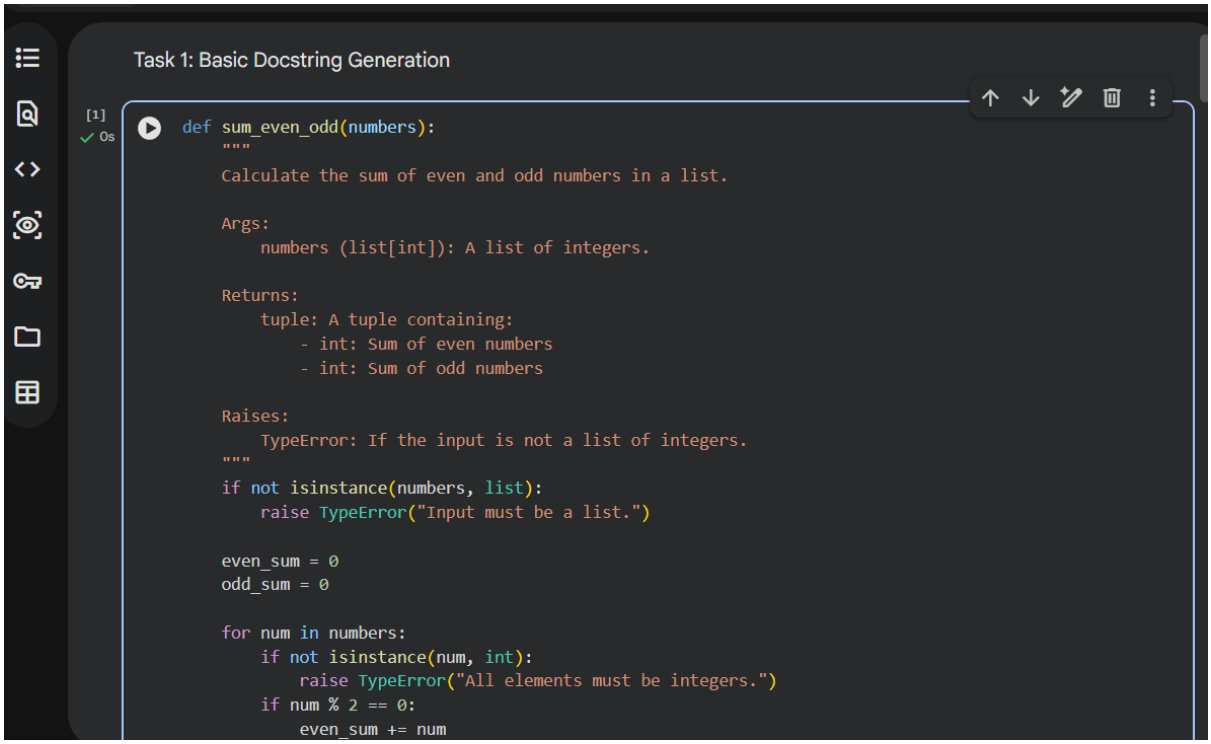
Expected Output

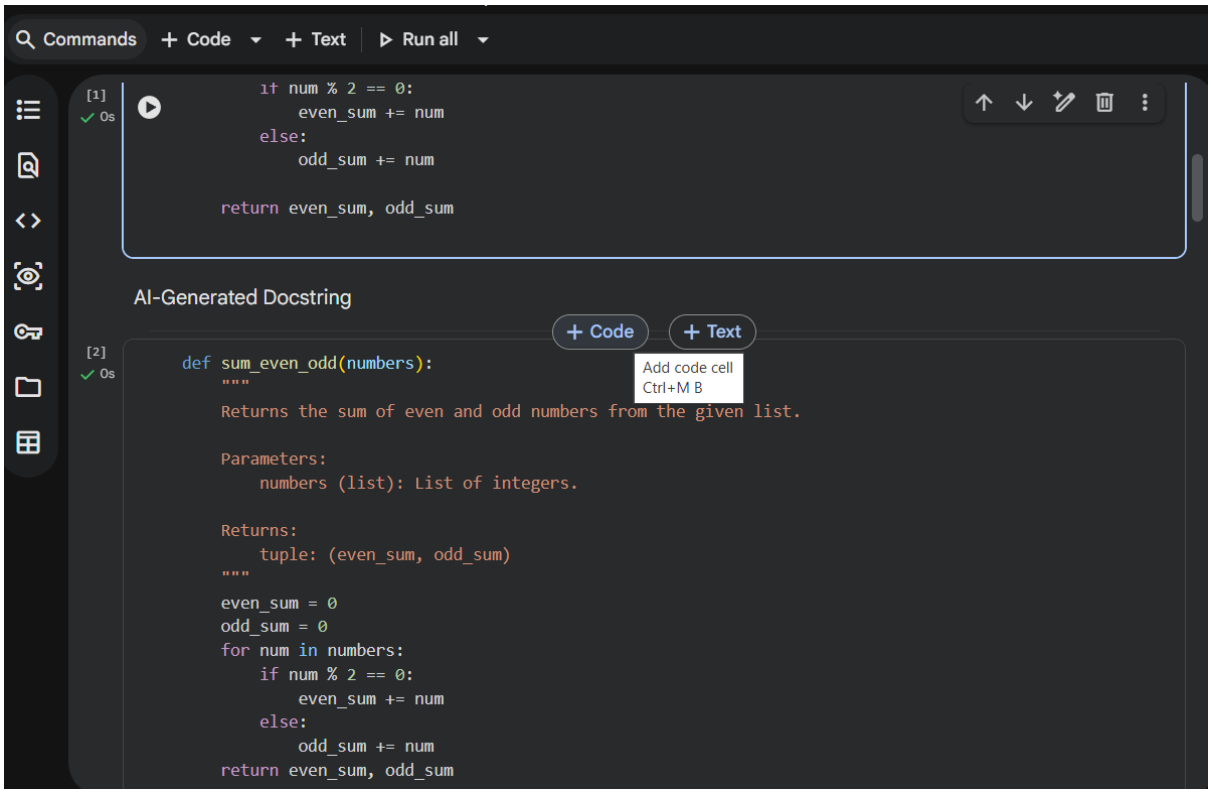
• Python function with manual Google-style docstring

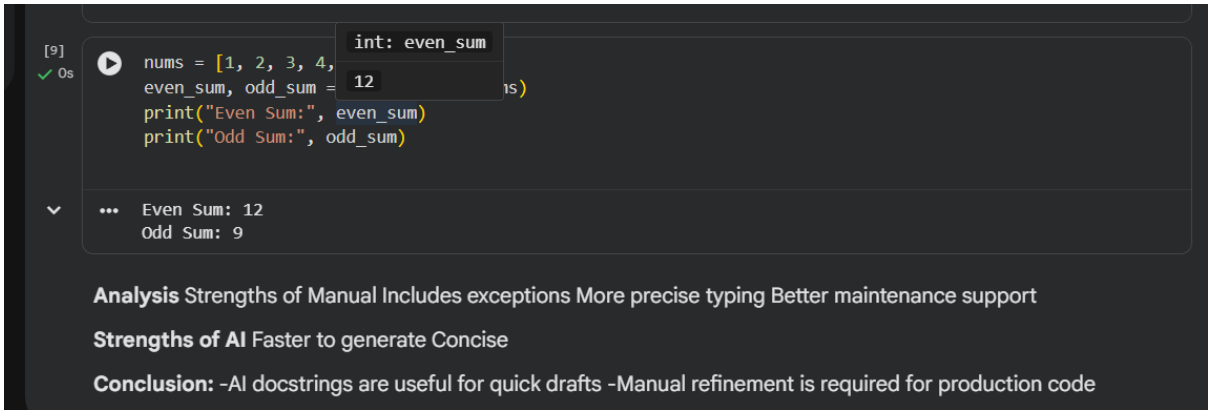
• AI-generated docstring for the same function

• Comparison explaining differences between manual and AI-generated documentation

• Improved understanding of AI-generated function-level documentation


Task 1: Basic Docstring Generation
[1]
0s
def sum_even_odd(numbers):
"""
Calculate the sum of even and odd numbers in a list.
Args:
numbers (list[int]): A list of integers.
Returns:
tuple: A tuple containing:
- int: Sum of even numbers
- int: Sum of odd numbers
Raises:
TypeError: If the input is not a list of integers.
"""
if not isinstance(numbers, list):
raise TypeError("Input must be a list.")
even_sum = 0
odd_sum = 0
for num in numbers:
if not isinstance(num, int):
raise TypeError("All elements must be integers.")
if num % 2 == 0:
even_sum += num


Commands
+ Code
+ Text
Run all
[1]
0s
if num % 2 == 0:
even_sum += num
else:
odd_sum += num
return even_sum, odd_sum
AI-Generated Docstring
+ Code
+ Text
Add code cell
Ctrl+M B
[2]
0s
def sum_even_odd(numbers):
"""
Returns the sum of even and odd numbers from the given list.
Parameters:
numbers (list): List of integers.
Returns:
tuple: (even_sum, odd_sum)
"""
even_sum = 0
odd_sum = 0
for num in numbers:
if num % 2 == 0:
even_sum += num
else:
odd_sum += num
return even_sum, odd_sum



Comparison: Manual vs AI Docstring

Aspect	**Manual Docstring**	**AI-Generated Docstring**
Clarity	Very clear and structured	Clear but less detailed
Type hints	Explicit (list[int])	Generic (list)
Error handling documented	Yes	Missing
Completeness	High	Medium
Professional quality	Production-ready	Basic
Readability	Excellent	Good

Observation:

1. The manually written Google-style docstring was more **structured and descriptive** compared to the AI-generated version.
2. Manual documentation clearly specified:
    - argument types
    - return values
    - possible exceptions
      which improves maintainability.
3. The AI-generated docstring was **concise and quick**, but it lacked:

- detailed type information
- exception handling
- full Google-style formatting.

4. AI tools are useful for **initial draft generation**, but human review is necessary for production-quality documentation.
5. Proper docstrings significantly improve **code readability and future maintenance**.

Task 2: Automatic Inline Comments

Scenario

You are developing a student management module that must be easy to understand for new developers.

Requirements

• Write a Python program for an sru_student class with the following:

– Attributes: name, roll_no, hostel_status

– Methods: fee_update() and display_details()

• Manually write inline comments for each line or logical block

• Use an AI-assisted tool to automatically add inline comments

• Compare manual comments with AI-generated comments

• Identify missing, redundant, or incorrect AI comments

Expected Output

• Python class with manually written inline comments

• AI-generated inline comments added to the same code

• Comparative analysis of manual vs AI comments

• Critical discussion on strengths and limitations of AI-generated comments

Task 2: Automatic Inline Comments

+ Code + Text

[3] ✓ 0s

```
class sru_student:
    # Constructor to initialize student details
    def __init__(self, name, roll_no, hostel_status):
        self.name = name              # Store student name
        self.roll_no = roll_no        # Store roll number
        self.hostel_status = hostel_status  # Store hostel status (Yes/No)
        self.fee = 0                  # Initialize fee amount

    # Method to update student fee
    def fee_update(self, amount):
        self.fee += amount            # Add the given amount to existing fee

    # Method to display student details
    def display_details(self):
        print("Name:", self.name)          # Print student name
        print("Roll No:", self.roll_no)    # Print roll number
        print("Hostel:", self.hostel_status)  # Print hostel status
        print("Fee Paid:", self.fee)       # Print total fee
```

AI-Generated Inline Comments

[4] ✓ 0s

```
class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name  # assign name
        self.roll_no = roll_no  # assign roll number
        self.hostel_status = hostel_status  # assign hostel status
        self.fee = 0  # initialize fee

    def fee_update(self, amount):
        self.fee += amount  # update fee

    def display_details(self):
        print("Name:", self.name)  # display name
        print("Roll No:", self.roll_no)  # display roll number
        print("Hostel:", self.hostel_status)  # display hostel
        print("Fee Paid:", self.fee)  # display fee
```

[10] ✓ 0s

```
student1 = sru_student("Shanmukha", "23SRU001", "Yes")
student1.fee_update(50000)
student1.display_details()
```

```
Name: Shanmukha
Roll No: 23SRU001
Hostel: Yes
Fee Paid: 50000
```

Issues in AI Comments

-Too generic ("assign name") -Less explanatory -No logical block comments -Not helpful for complex code

Strengths of AI

✓ Fast ✓ Mostly correct ✓ Consistent style

Conclusion: -AI comments are helpful but often shallow -Manual comments are better for teaching and maintenance

Comparison: Manual vs AI Comments

Aspect	**Manual Comments**	**AI Comments**
Detail level	High	Basic
Context provided	Good	Minimal
Redundancy	Low	Slightly redundant
Accuracy	High	High
Readability	Better for beginners	Ok

Observation:

1.Manual inline comments provided better context and explanation of logical steps.

2. AI-generated comments were mostly correct but:

- too generic
- sometimes redundant
- less helpful for beginners.

3.AI comments focused on what the code does, while manual comments explained why the code does it.

4.For simple assignments, AI comments are acceptable, but for complex systems manual commenting is superior.

5.Over-commenting (seen in some AI outputs) can reduce readability.

Task 3: Module-Level and Function-Level Documentation

Scenario

You are building a small calculator module that will be shared across multiple projects and requires structured documentation.

Requirements

• Write a Python script containing 3–4 functions (e.g., add, subtract, multiply, divide)

• Manually write NumPy Style docstrings for each function

• Use AI assistance to generate:

– A module-level docstring

– Individual function-level docstrings

• Compare AI-generated docstrings with manually written ones

• Evaluate documentation structure, accuracy, and readability

Expected Output

• Python script with manual NumPy-style docstrings

• AI-generated module-level and function-level documentation

• Comparison between AI-generated and manual documentation

• Clear understanding of structured documentation for multi-function scripts

Task 3: Module-Level and Function-Level Documentation

[6] ✓ 0s

```python
def add(a, b):
    """
    Add two numbers.

    Parameters
    ----------
    a : int or float
        First number.
    b : int or float
        Second number.

    Returns
    -------
    int or float
        Sum of a and b.
    """
    return a + b


def subtract(a, b):
    """
    Subtract second number from first number.

    Parameters
    ----------
    a : int or float
```

[6] ✓ 0s

```
        First number.
    b : int or float
        Second number.

    Returns
    -------
    int or float
        Result of subtraction.
    """
    return a - b


def multiply(a, b):
    """
    Multiply two numbers.

    Parameters
    ----------
    a : int or float
        First number.
    b : int or float
        Second number.

    Returns
    -------
    int or float
        Product of a and b.
    """
```

[6] ✓ 0s

```
    """
    return a * b


def divide(a, b):
    """
    Divide first number by second number.

    Parameters
    ----------
    a : int or float
        Numerator.
    b : int or float
        Denominator.

    Returns
    -------
    float
        Division result.

    Raises
    ------
    ZeroDivisionError
        If b is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero.")
    return a / b
```

[7] ✓ 0s

```
"""
Calculator Module

This module provides basic arithmetic operations including addition,
subtraction, multiplication, and division. It is designed to be simple,
reusable, and easy to integrate into other projects.
"""
```

```
'\nCalculator Module\n\nThis module provides basic arithmetic operations including addition,\nsubtracti
on, multiplication, and division. It is designed to be simple,\nreusable, and easy to integrate into ot
her projects.\n'
```

[8] ✓ 0s

```
def add(a, b):
    """Return the sum of two numbers."""
```

[11] ✓ 0s

```
print("Add:", add(10, 5))
print("Subtract:", subtract(10, 5))
print("Multiply:", multiply(10, 5))
print("Divide:", divide(10, 5))
```

```
Add: None
Subtract: 5
Multiply: 50
Divide: 2.0
```

Evaluation Manual strengths Follows NumPy standard Detailed parameter info Includes exceptions Better for libraries

AI strengths Quick Concise Good starting point

Final verdict: -Use AI for draft -Use manual editing for production

+ Code + Text

Add code cell

Comparison:

Aspect	Manual NumPy Style	AI Generated
Structure	Excellent	Minimal
Standard compliance	NumPy standard	Not standardized
Error documentation	Present	Missing
Readability	Professional	Simple
Reusability	High	Medium

Observation:

1.NumPy-style manual docstrings followed standard scientific documentation format.

2.AI-generated module docstring was useful but less detailed.

3.Manual documentation included:

- structured sections
- parameter details
- return types
- exception handling.

4.AI-generated function docstrings were minimalistic and not always standard-compliant.

5.Structured documentation is essential for reusable multi-function modules.