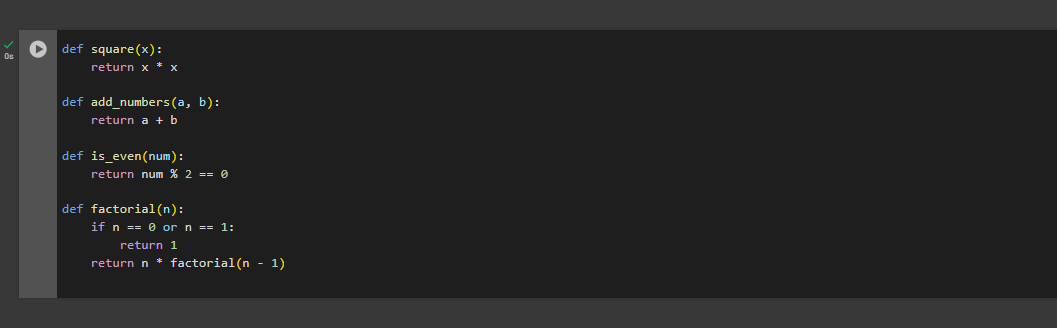
**LAB ASSIGNMENT 9.1**

**Name: Yarlagadda Mohith Sai Shankar**

**Hall ticket: 2403A51323**

**Task Description #1** (Documentation – Google-Style Docstrings for  
Python Functions)

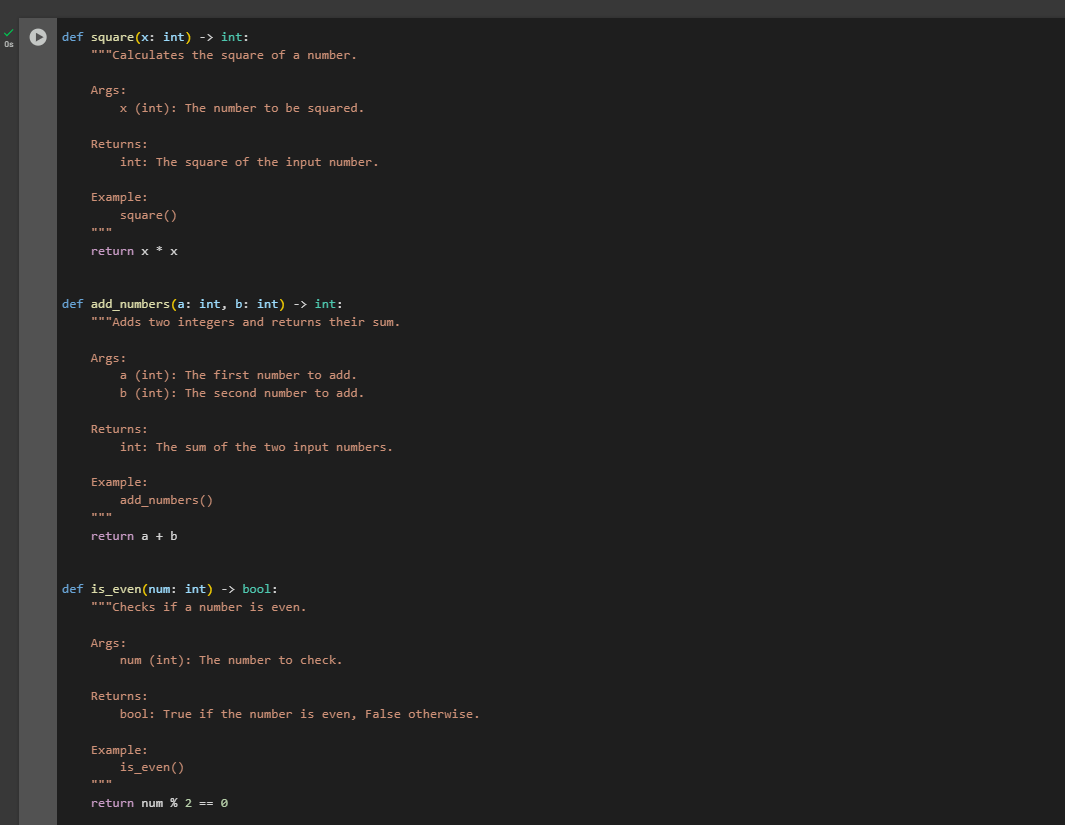
**Original Script (without docstrings):**

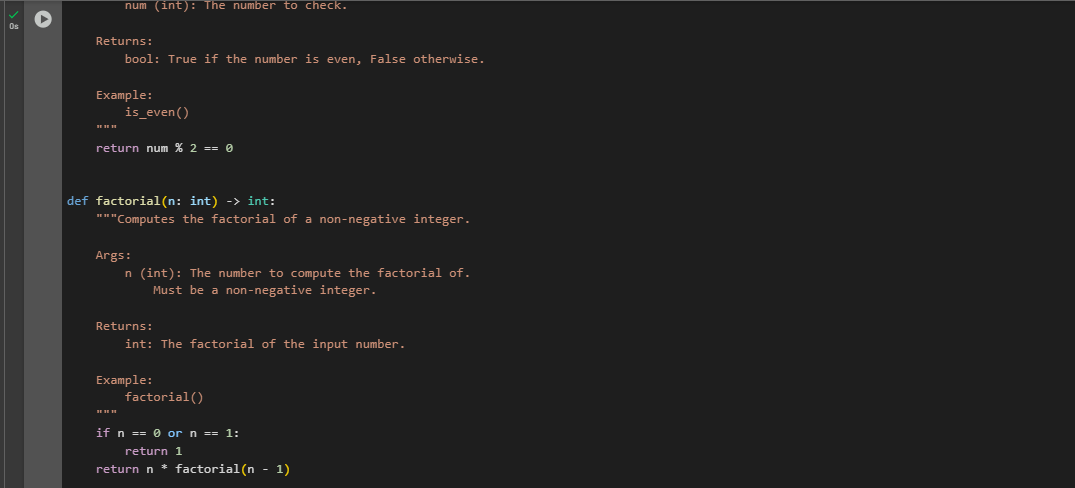
****

**Observation:**

The given Python script contains four functions: square, add\_numbers, is\_even, and factorial.  
Each function performs a basic mathematical operation and the logic is correct.  
However, the script does not include docstrings to explain what each function does, what type of inputs it expects, or what it returns.  
There are also no type hints, which makes it less clear whether the functions are intended for integers, floats, or other data types.  
Because of this, a new reader or user of the code must carefully read the function implementation to understand its purpose.  
While the code is small and understandable, in larger projects the lack of documentation would make the script harder to maintain, reuse, and share with others.

**Script After Adding Google-Style Docstrings:**

****

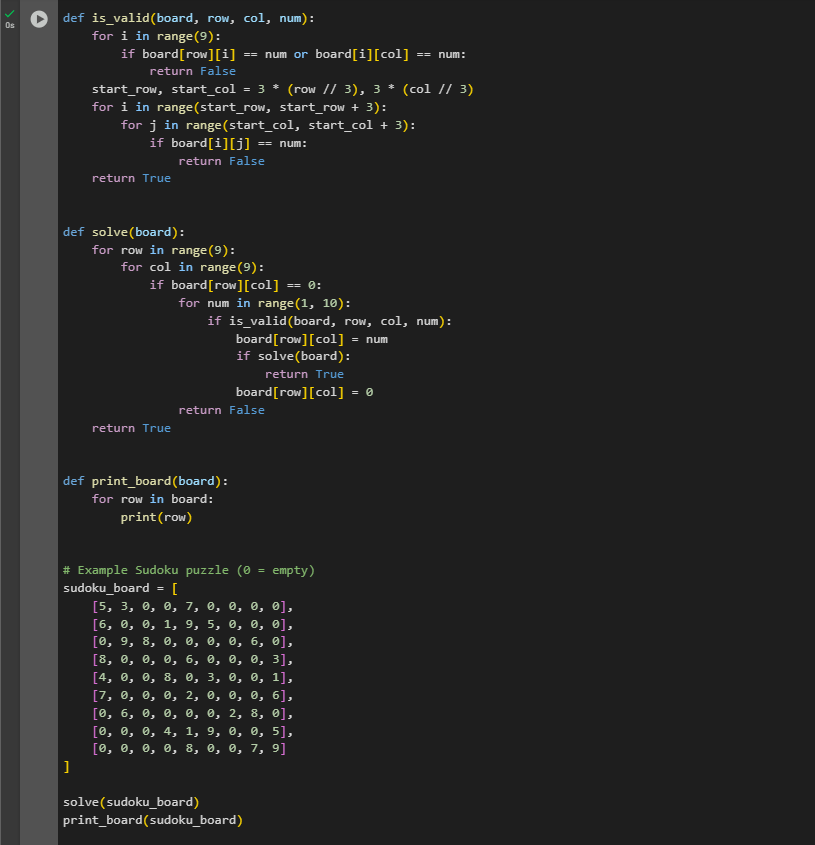
****

**Observation:**

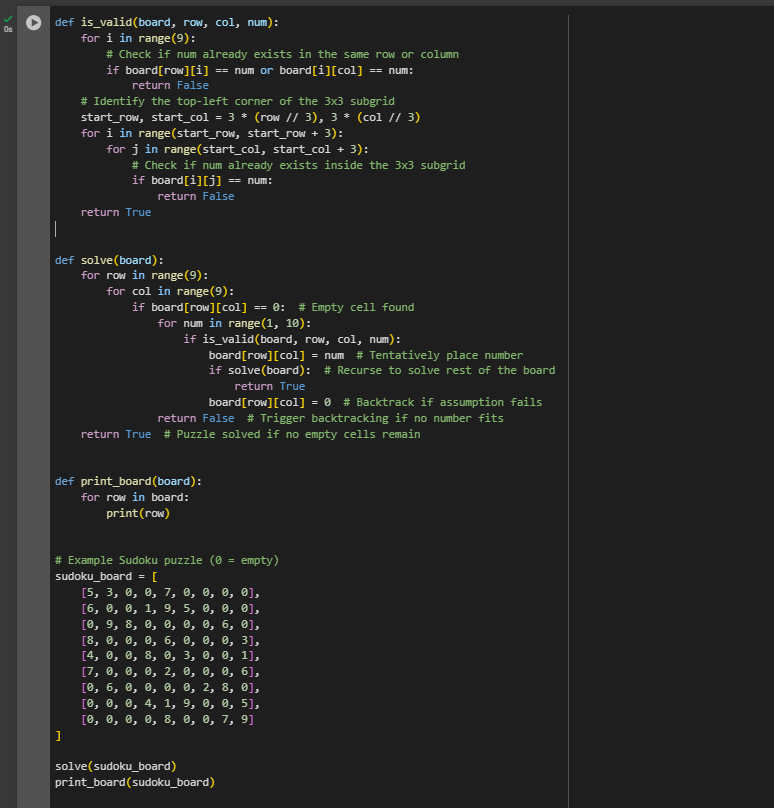
The updated script contains the same four functions: square, add\_numbers, is\_even, and factorial.  
This time, each function includes Google-style docstrings along with type hints.  
The docstrings clearly describe the purpose of the function, the parameters with their types, the return type, and also provide a simple example usage.  
This makes the code much easier to read and understand, even for someone who has never seen it before.  
Users can now call help(function\_name) in Python to quickly learn how to use each function without reading the internal code.  
Overall, the script is now more professional, user-friendly, and maintainable, and it follows good coding practices suitable for larger projects.

**Task Description #2** (Documentation – Inline Comments for Complex  
Logic)

**Python Script (Without Comments):**

****

**Python Script (With Comments for tricky or non-intuitive code sections):**

****

**Observation:**

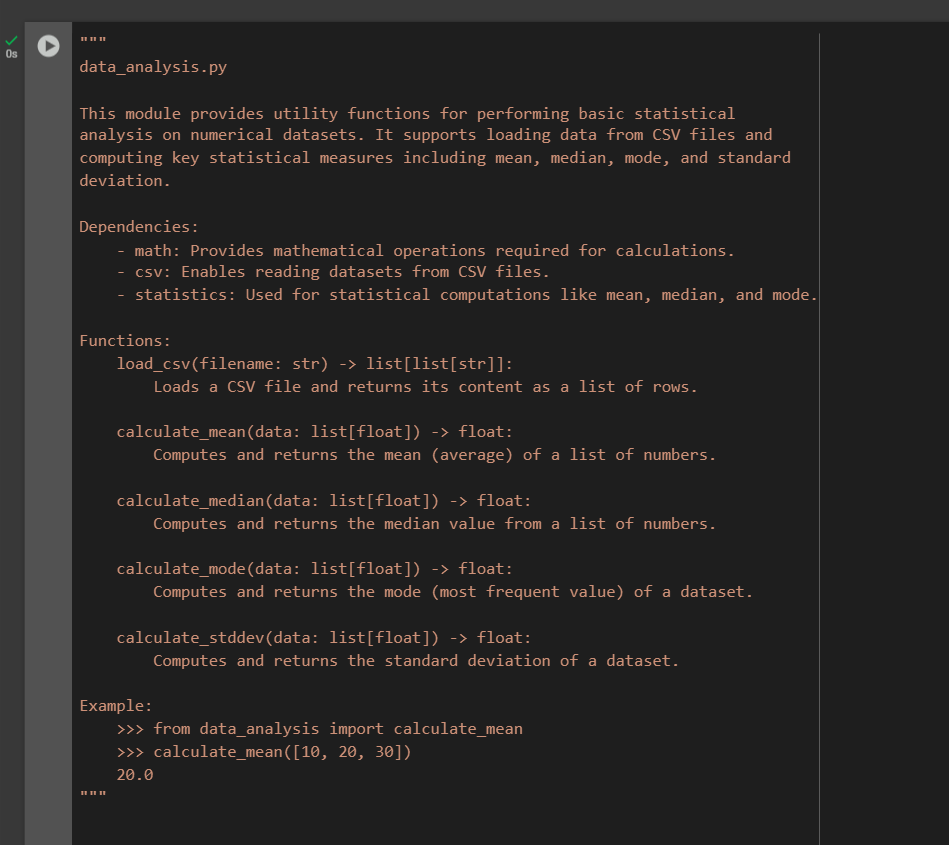
The Sudoku solver uses **backtracking** to fill empty cells with valid numbers while checking Sudoku rules.  
Inline comments have been added only to explain **non-intuitive logic**, such as:

* Validity checks across rows, columns, and 3×3 subgrids.
* The calculation of the subgrid’s starting index.
* The recursive backtracking process of placing, validating, and undoing numbers.
* The reason for returning False when no valid number can be placed.

The comments do not explain obvious syntax like loops or assignments, keeping the code clean while clarifying the tricky parts.  
This improves **readability and maintainability**, making it easier for future developers to follow the algorithm’s flow without cluttering the script with unnecessary explanations.

**Task Description #3 (Documentation – Module-Level Documentation):**

**Python file:**

****

**Python file link:**

<https://drive.google.com/file/d/1Ved7gy-Jdgf7f4qrnsD89k8bg4WMNiF6/view?usp=drive_link>

**Observation:**

we used AI to generate a **module-level docstring** for a given Python file. The AI analyzed the entire code and summarized its **purpose, dependencies, functions, and usage** in a clear and structured manner. The documentation was written in **Google-style format**, making it easy to understand and maintain.

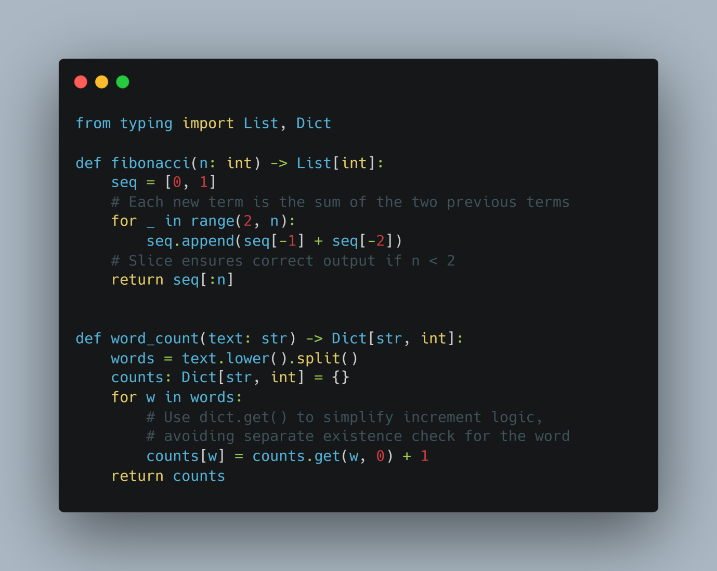
The docstring provided:

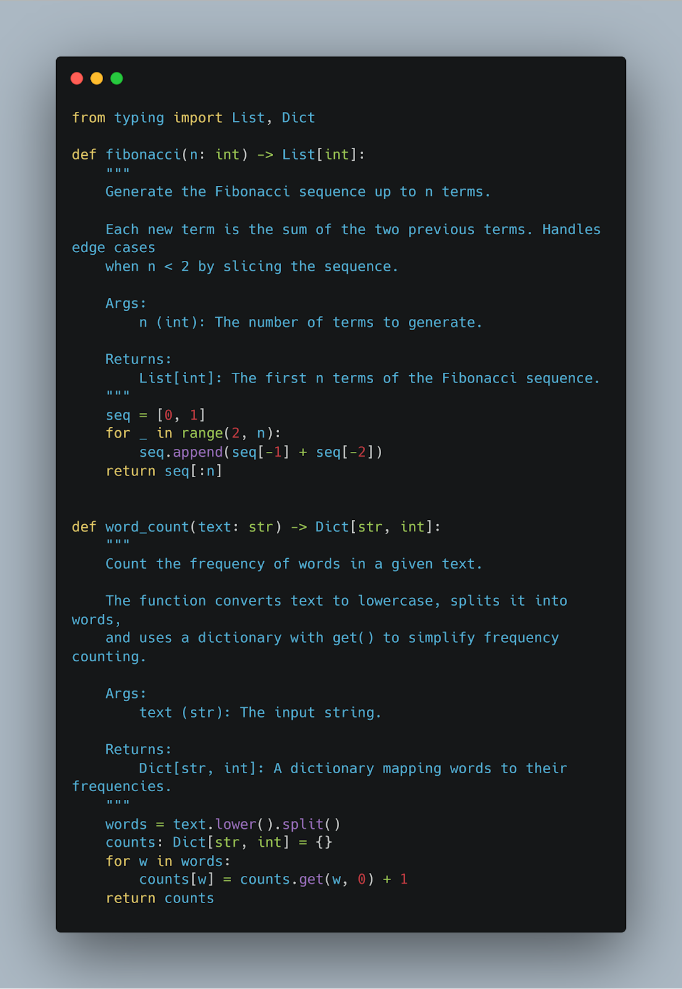
* A **brief description** of the module’s purpose.
* A list of **dependencies (imports)** required by the file.
* A summary of **all main functions and classes** defined in the code.
* An **example usage snippet** showing how the module can be used in practice.

This ensures the Python file is **self-documented**, improving readability, maintainability, and helping future developers quickly understand the code.

**Task Description #4** (Documentation – Convert Comments to Structured Docstrings):

**Original Code:**



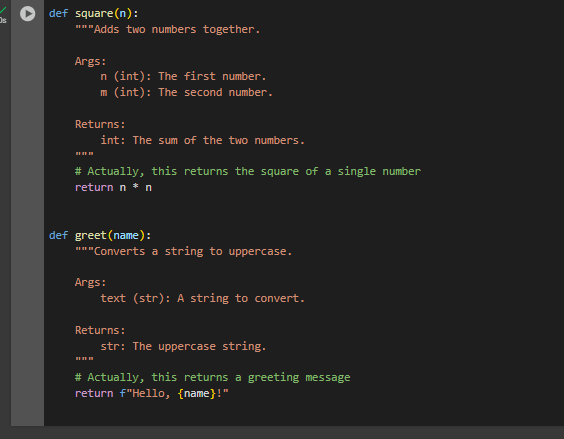


**Observation:**

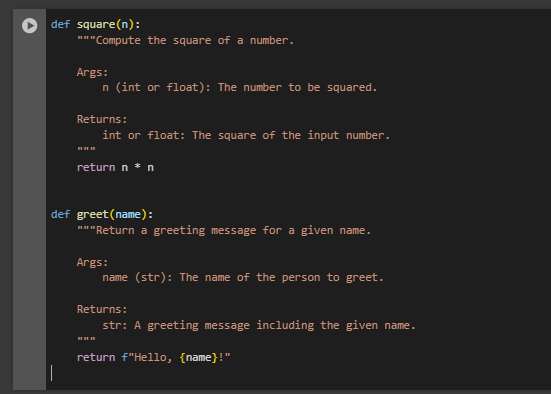
Turning the comments into docstrings made the code is cleaner and easier to read. Now the purpose and details of each function are right at the top, so you don’t have to dig through the code to understand it. It feels more beginner-friendly and organized.

**Task Description #5** (Documentation – Review and Correct Docstrings):

**Input code (with inline comments):**



**Output code (with Google-style docstring):**



**Observation:**

The original docstrings were **incorrect or misleading** compared to the actual function behavior.

Functions were analyzed to understand their true purpose before rewriting the docstrings.

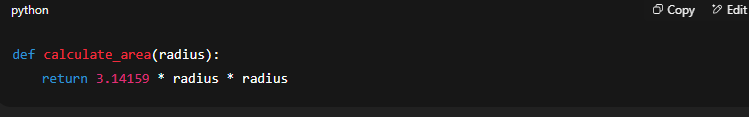
Google-style docstrings were added with:

* **Brief description** of the function.
* **Args** section specifying parameters and types.
* **Returns** section specifying the return value and type.

The updated docstrings are **accurate, structured, and readable**, improving code documentation and maintainability.

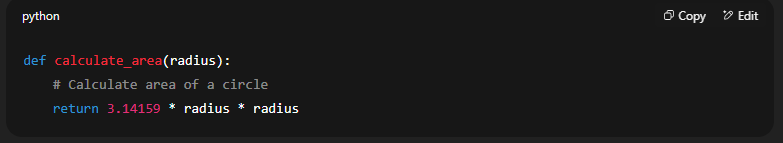
**Task Description #6** (Documentation – Prompt Comparison Experiment)

Python function:



**Vague Prompt:**

Prompt:  
“Add comments to this function.”

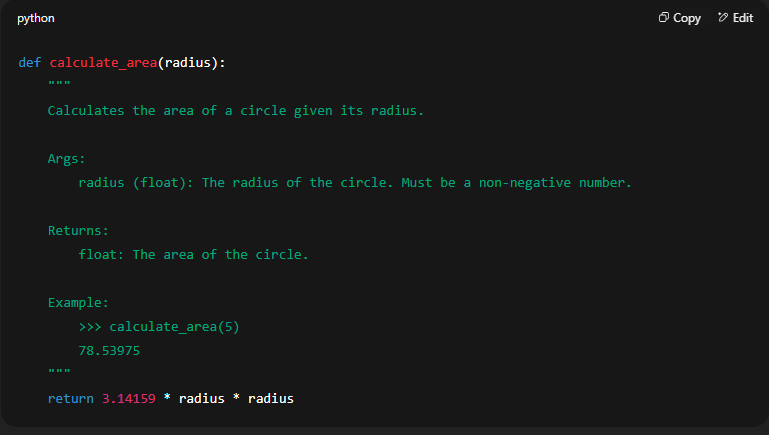


**Observations for Vague Prompt:**

* **Quality:** Very basic; only a single inline comment.
* **Accuracy:** Correctly states what the function does, but very minimal.
* **Completeness:** Lacks detail about parameters, return type, units, or examples. It may not help someone unfamiliar with Python or math concepts.

**Detailed Prompt:**

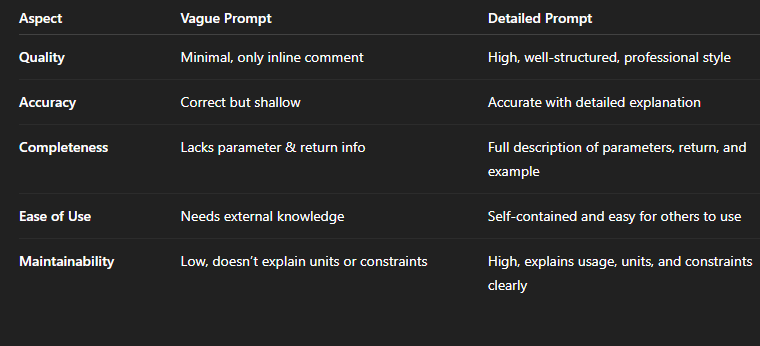
**Prompt:**  
“Add Google-style docstrings with parameters, return types, and examples.”



**Observations for Detailed Prompt:**

* **Quality:** High-quality documentation; structured, readable, and professional.
* **Accuracy:** Accurately describes the purpose, parameter type, return type, and provides an example.
* **Completeness:** Fully self-contained; someone unfamiliar with the function can understand how to use it. Includes units implicitly (area), parameter constraints, and example output.

**Comparison Table:**



**Key Takeaways:**

1. Vague prompts produce minimal and sometimes insufficient documentation.
2. Detailed prompts guide AI to create professional, complete, and usable documentation.
3. For real projects, detailed prompts save time and reduce errors for future users or developers.