**Lab 2: Exploring Additional AI Coding Tools – Gemini (Collab) and Cursor AI**

**ASSIGNMENT: 2.3**

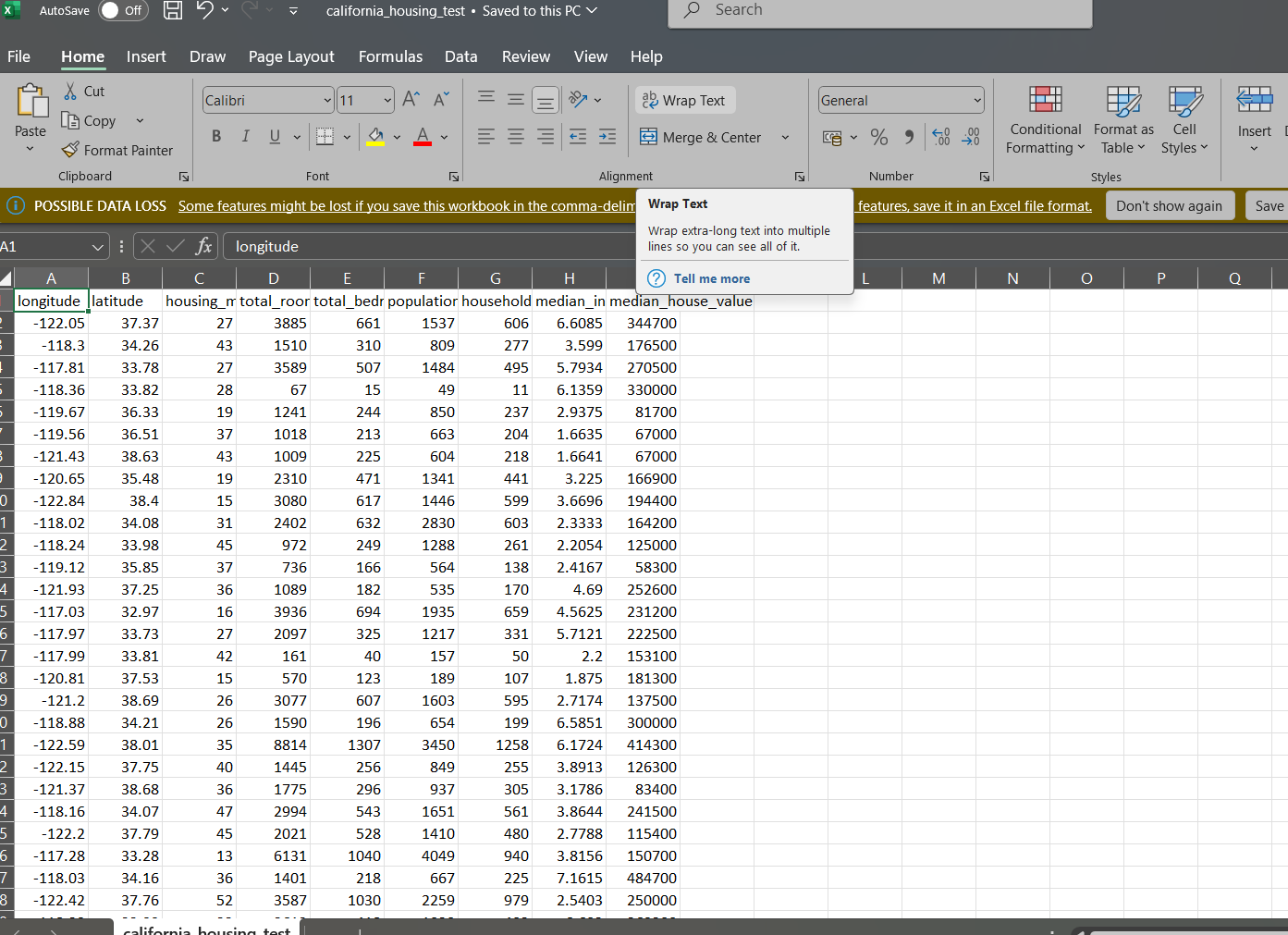
**HTNO:2403a51344**

**Name: Bollam Sathvika**

**Batch no:24BTCAICSB14**

**TASK 1:**

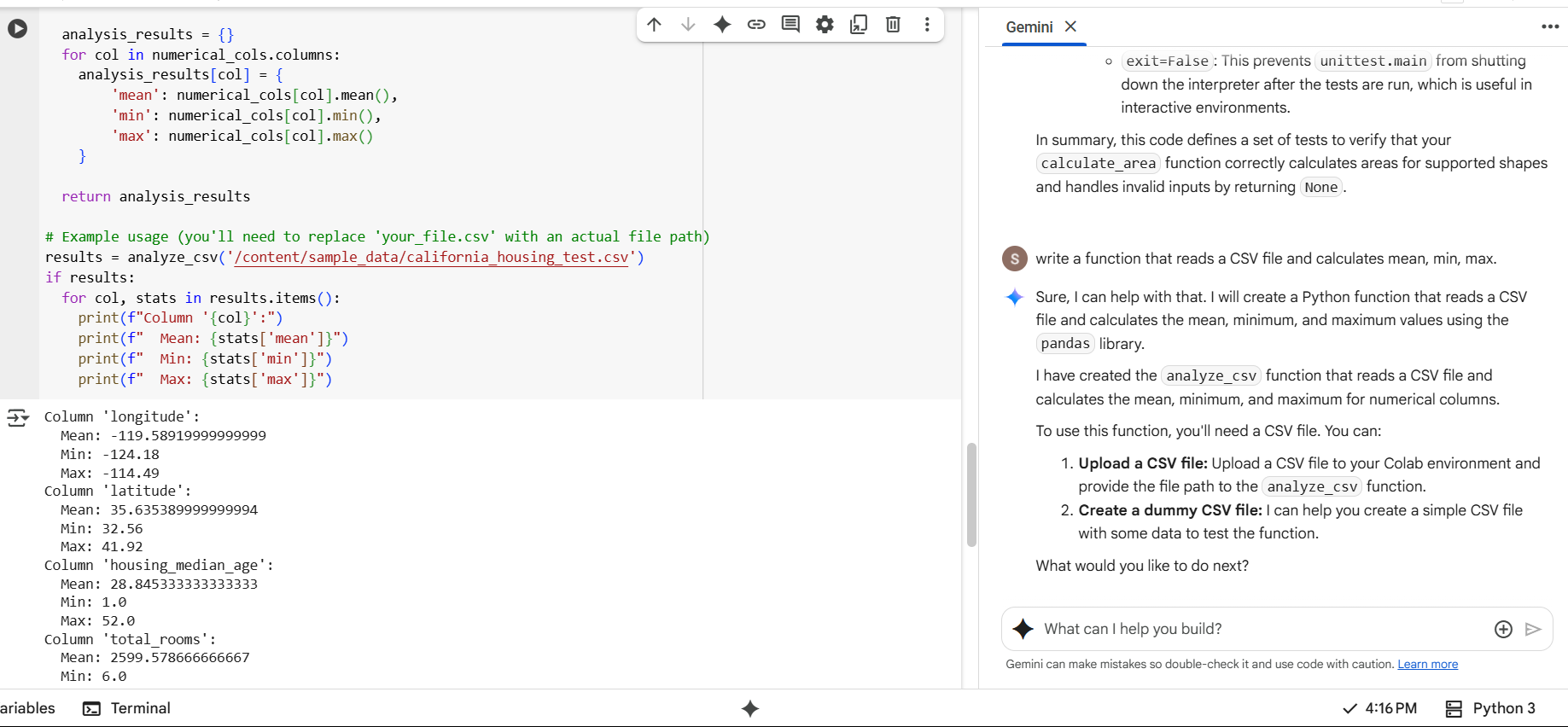
**Step1:** Create a file with .csv extension

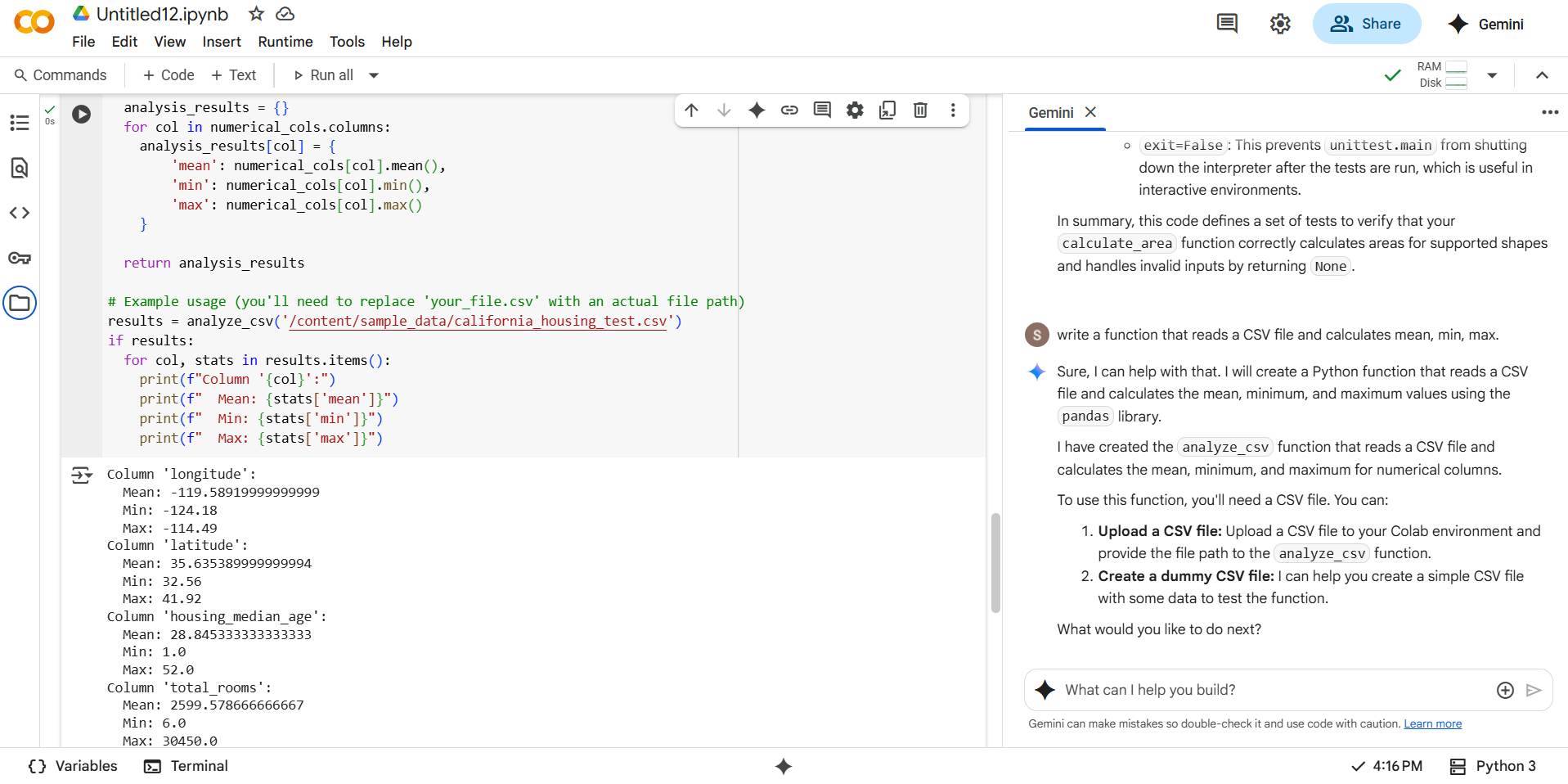


**Step 2:**

Calculates mean, max, min

**Code and Output:**

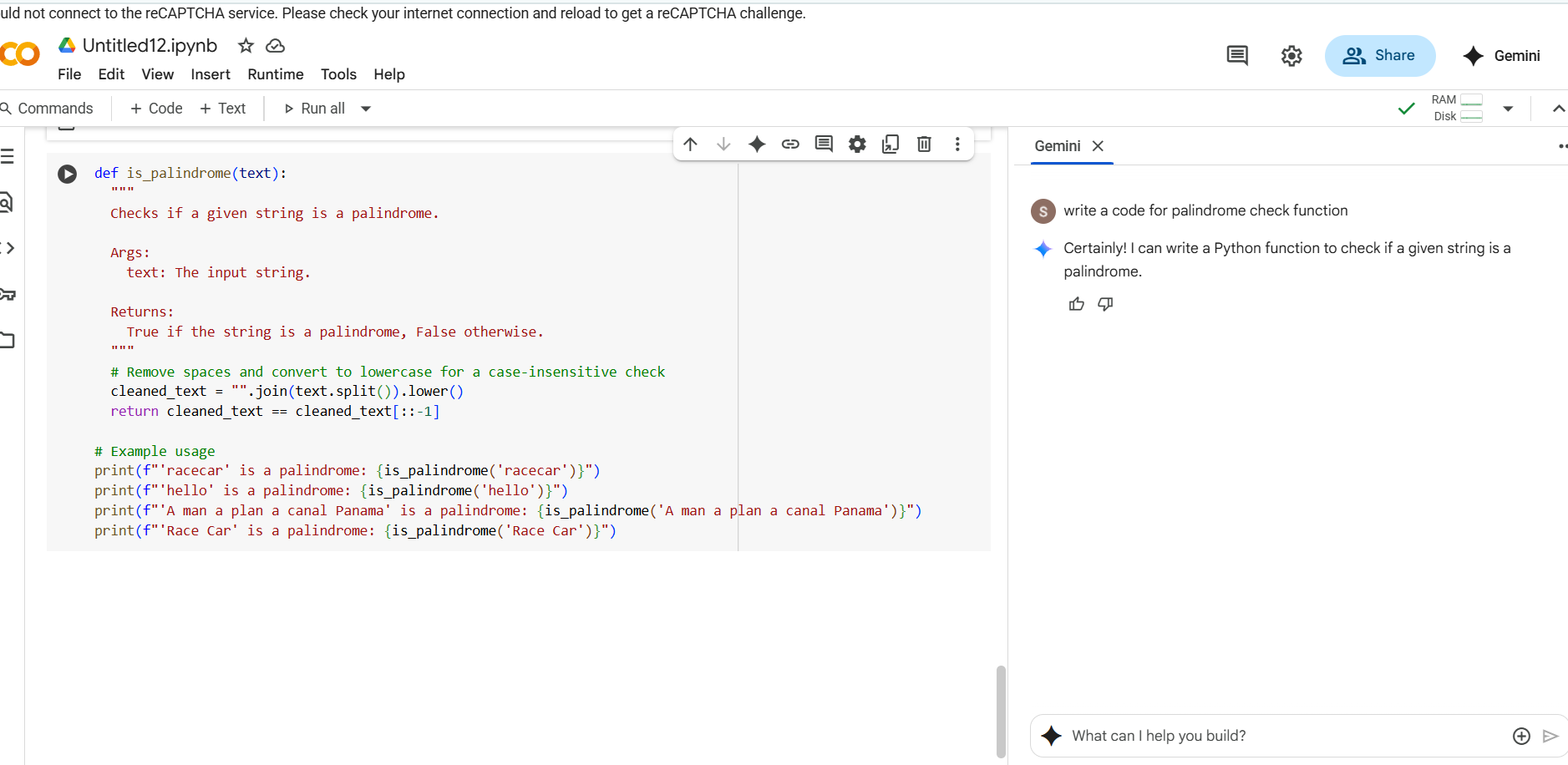
****

****

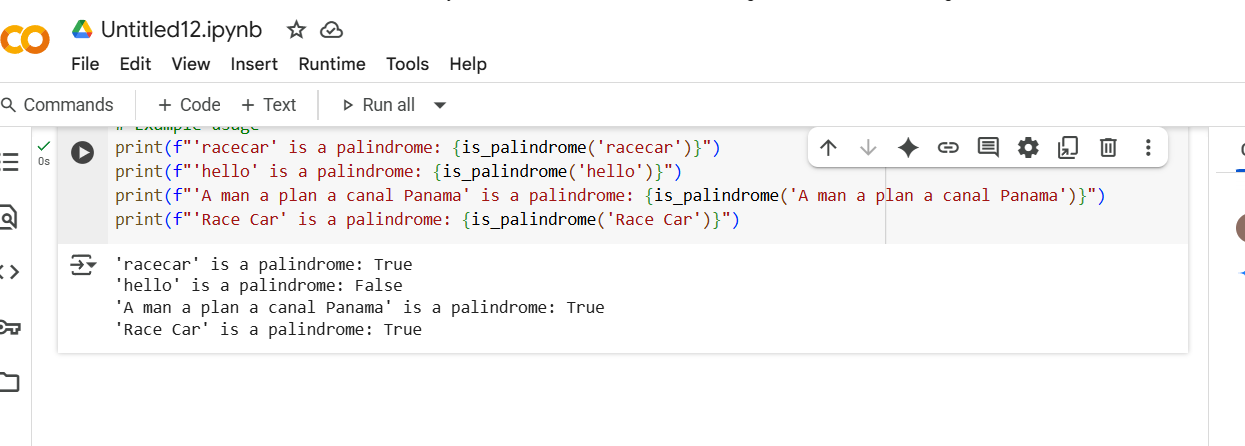
**Task 2:**

**Gemini-google colab version:**

**Code:**

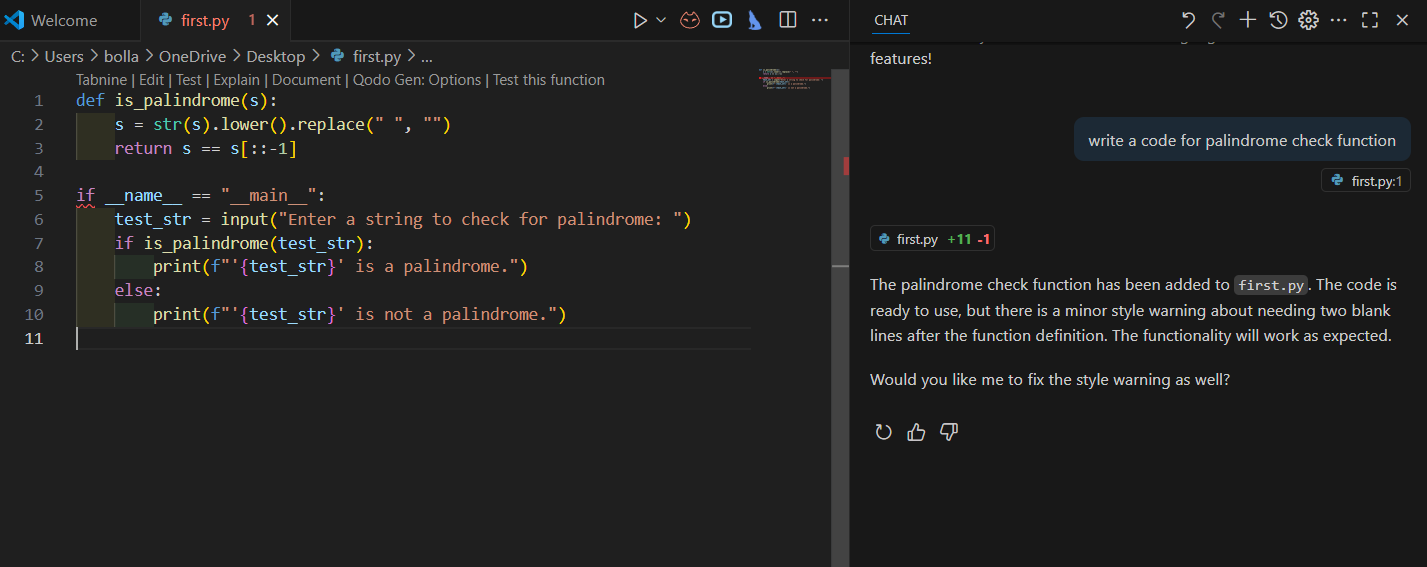
****

**Output:**

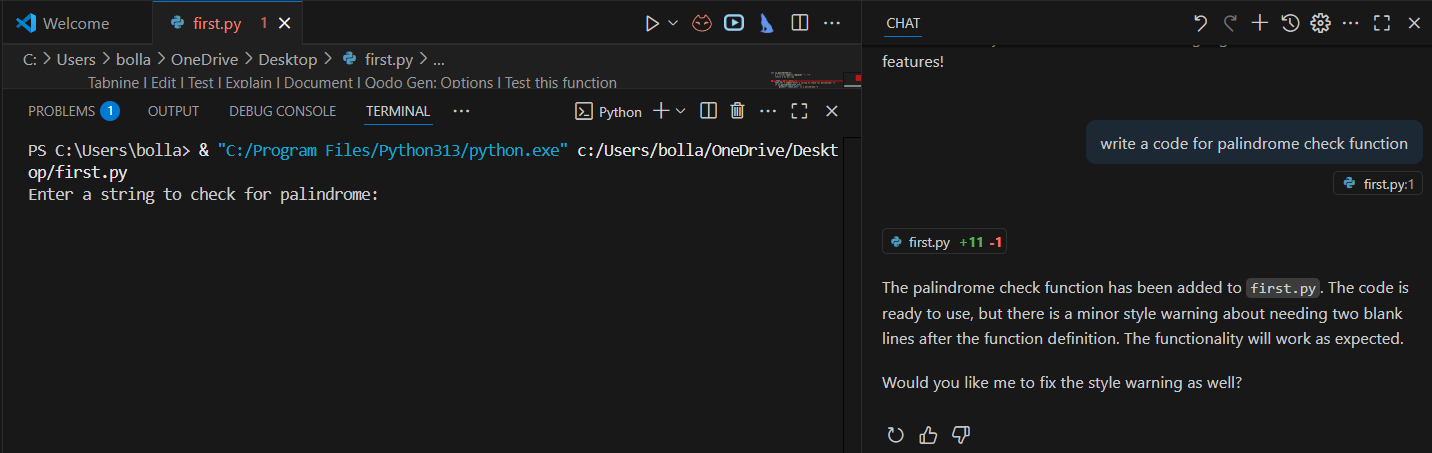
****

**Copilot-VS code version:**

**Code:**

****

**Output:**



**Comparison table of Gemini and Copilot output.**

| **Feature** | **Gemini Output** | **Copilot Output** |
| --- | --- | --- |
| Function Name | is\_palindrome | is\_palindrome |
| Input Cleaning | Lowercase + removes spaces | Lowercase + removes non-alphanumeric chars |
| Case Sensitivity | Not sensitive | Not sensitive |
| Spaces Handling | Yes | Yes |
| Special Characters | Not handled | Handles special characters |
| Code Length | More lines, includes docstring and examples | Shorter, direct function usage |
| Readability | Beginner-friendly with docstring | Concise and practical |
| Robustness | Handles spaces and case, but not special symbols | Handles spaces, case, and special symbols |

**Observation:**

Gemini’s version feels like it’s written with beginners in mind. It’s longer, includes a helpful docstring, and even provides examples—great for someone just starting out and trying to understand how things work step by step. It’s friendly and educational.

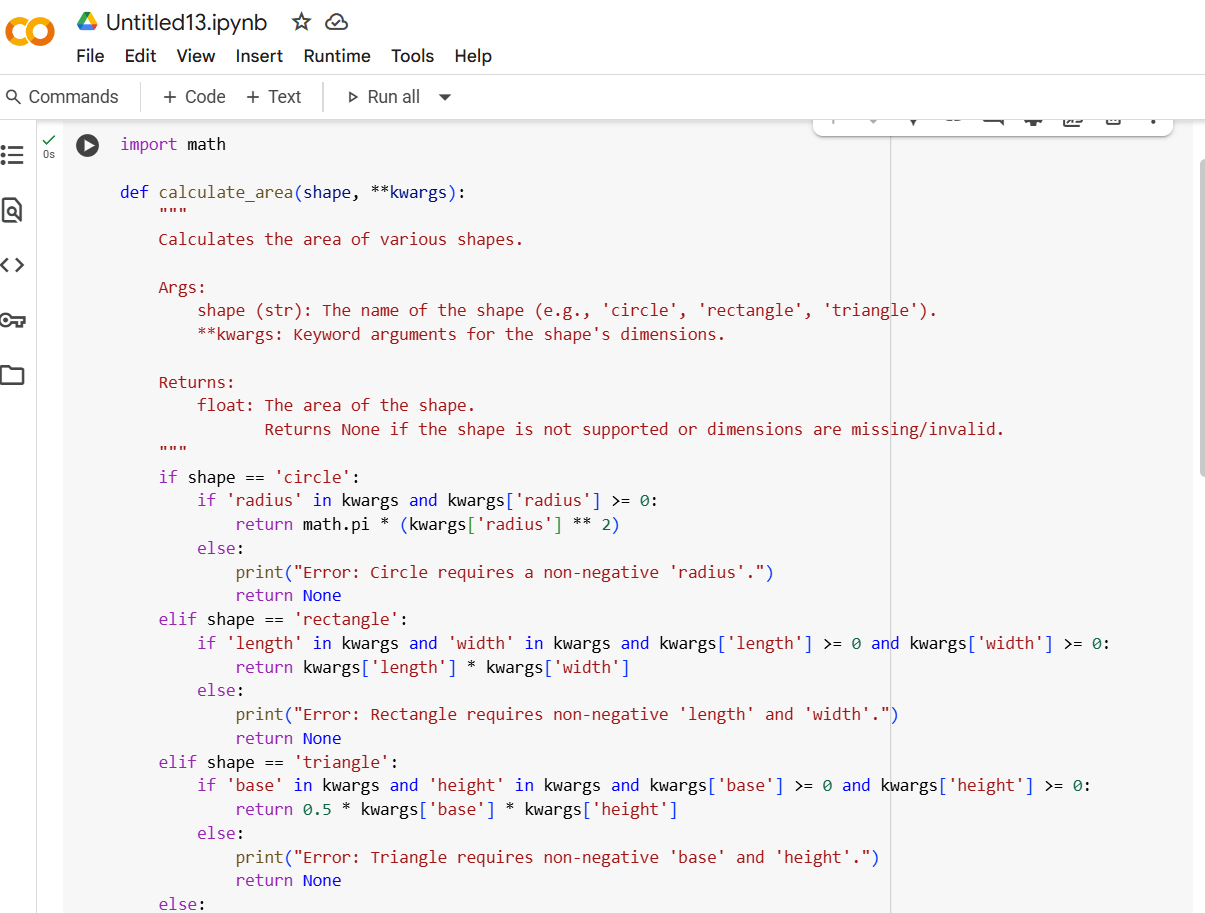
On the other hand, Copilot’s version is more compact and input-safe. It strips out anything that could mess with the logic—like special characters—making it more robust for real-world use. It’s clean, efficient, and gets straight to the point, which is perfect if you already know the basics and just want a reliable function.

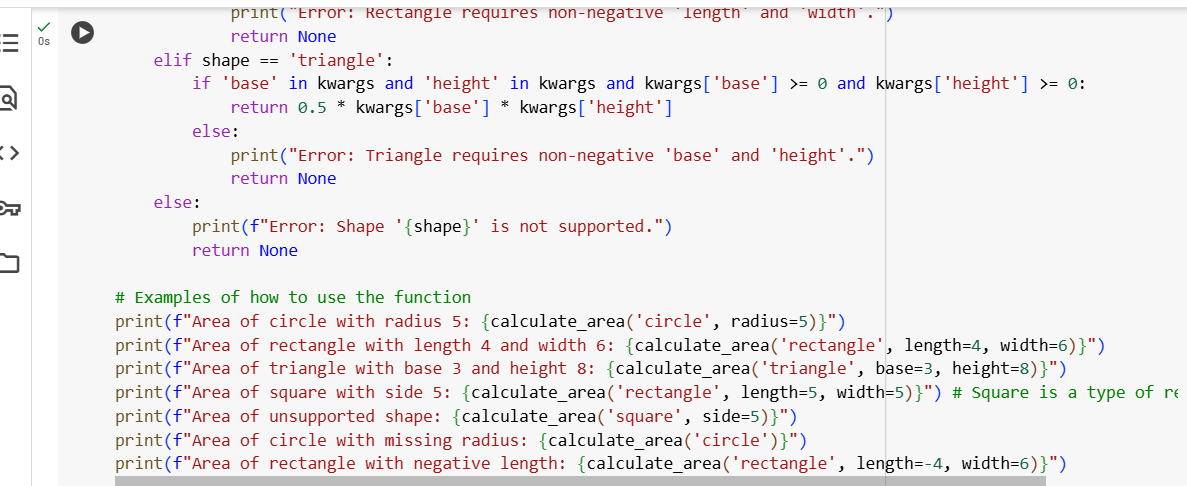
Both are solid in terms of performance, but they serve slightly different purposes: Gemini teaches, Copilot protects.

**Task 3:**

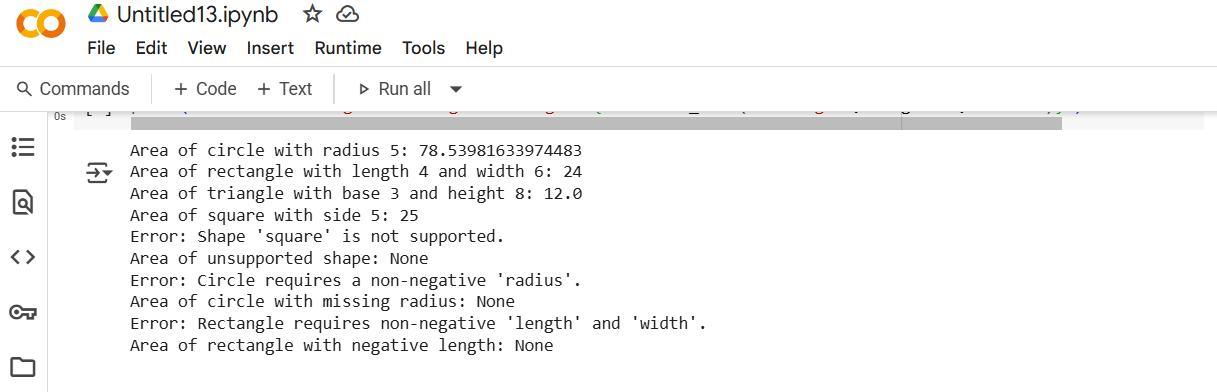
**Prompt:** Explain a python function to calculate are of various shapes line by line.

**CODE:**

****

****

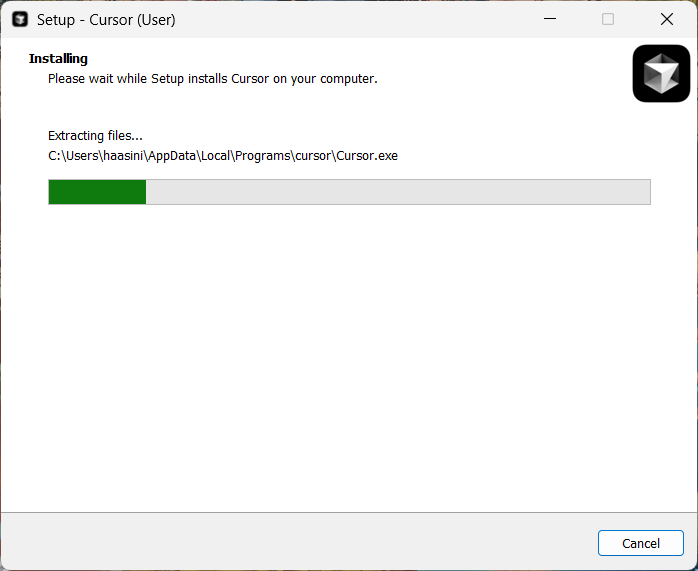
**OUTPUT:**

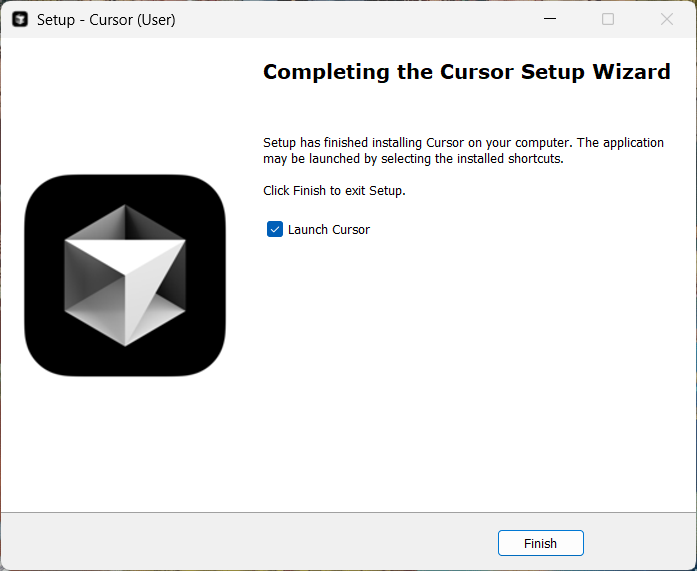
****

**EXPLANATION:**

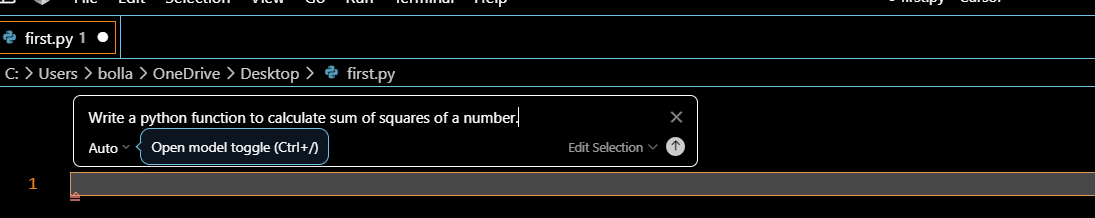
* **import math**: This line imports the math module, which provides access to mathematical functions and constants, such as math.pi for calculating the area of a circle.
* **def calculate\_area(shape, \*\*kwargs):**: This line defines a function named calculate\_area that takes two arguments:
  + shape: A string representing the name of the shape (e.g., 'circle', 'rectangle', 'triangle').
  + \*\*kwargs: This is a special syntax in Python that allows the function to accept an arbitrary number of keyword arguments. These keyword arguments will be used to provide the dimensions of the shape (e.g., radius, length, width, base, height).
* **""" ... """**: This is a docstring, which provides a brief explanation of what the function does, its arguments, and what it returns.
* **if shape == 'circle':**: This line starts a conditional block that checks if the value of the shape argument is the string 'circle'.
* **if 'radius' in kwargs and kwargs['radius'] >= 0:**: This nested conditional block checks if the keyword argument 'radius' is present in kwargs and if its value is a non-negative number.
* **return math.pi \* (kwargs['radius'] \*\* 2)**: If the shape is 'circle' and a valid 'radius' is provided, this line calculates the area of the circle using the formula π \* radius^2 and returns the result.
* **else:**: This is the else block for the inner if statement.
* **print("Error: Circle requires a non-negative 'radius'.")**: If the shape is 'circle' but 'radius' is missing or negative, this line prints an error message.
* **return None**: This line returns None to indicate that the area could not be calculated due to an invalid radius.
* **elif shape == 'rectangle':**: This line is part of the outer conditional block and checks if the value of shape is 'rectangle' if it wasn't 'circle'.
* **if 'length' in kwargs and 'width' in kwargs and kwargs['length'] >= 0 and kwargs['width'] >= 0:**: This nested conditional block checks if the keyword arguments 'length' and 'width' are present in kwargs and if their values are non-negative numbers.
* **return kwargs['length'] \* kwargs['width']**: If the shape is 'rectangle' and valid 'length' and 'width' are provided, this line calculates the area of the rectangle using the formula length \* width and returns the result.
* **else:**: This is the else block for the inner if statement.
* **print("Error: Rectangle requires non-negative 'length' and 'width'.")**: If the shape is 'rectangle' but 'length' or 'width' is missing or negative, this line prints an error message.
* **return None**: This line returns None to indicate that the area could not be calculated due to invalid dimensions.
* **elif shape == 'triangle':**: This line is part of the outer conditional block and checks if the value of shape is 'triangle' if it wasn't 'circle' or 'rectangle'.
* **if 'base' in kwargs and 'height' in kwargs and kwargs['base'] >= 0 and kwargs['height'] >= 0:**: This nested conditional block checks if the keyword arguments 'base' and 'height' are present in kwargs and if their values are non-negative numbers.
* **return 0.5 \* kwargs['base'] \* kwargs['height']**: If the shape is 'triangle' and valid 'base' and 'height' are provided, this line calculates the area of the triangle using the formula 0.5 \* base \* height and returns the result.
* **else:**: This is the else block for the inner if statement.
* **print("Error: Triangle requires non-negative 'base' and 'height'.")**: If the shape is 'triangle' but 'base' or 'height' is missing or negative, this line prints an error message.
* **return None**: This line returns None to indicate that the area could not be calculated due to invalid dimensions.
* **else:**: This is the else block for the outer conditional block.
* **print(f"Error: Shape '{shape}' is not supported.")**: If the shape argument does not match any of the supported shapes ('circle', 'rectangle', 'triangle'), this line prints an error message indicating that the shape is not supported.
* **return None**: This line returns None to indicate that the area could not be calculated because the shape is not supported.
* **print(f"Area of circle with radius 5: {calculate\_area('circle', radius=5)}")**: These lines are examples of how to call the calculate\_area function with different shapes and dimensions and print the returned area. They demonstrate how to calculate the area of a circle, rectangle, triangle, a square (as a type of rectangle), an unsupported shape, a circle with a missing radius, and a rectangle with a negative length.

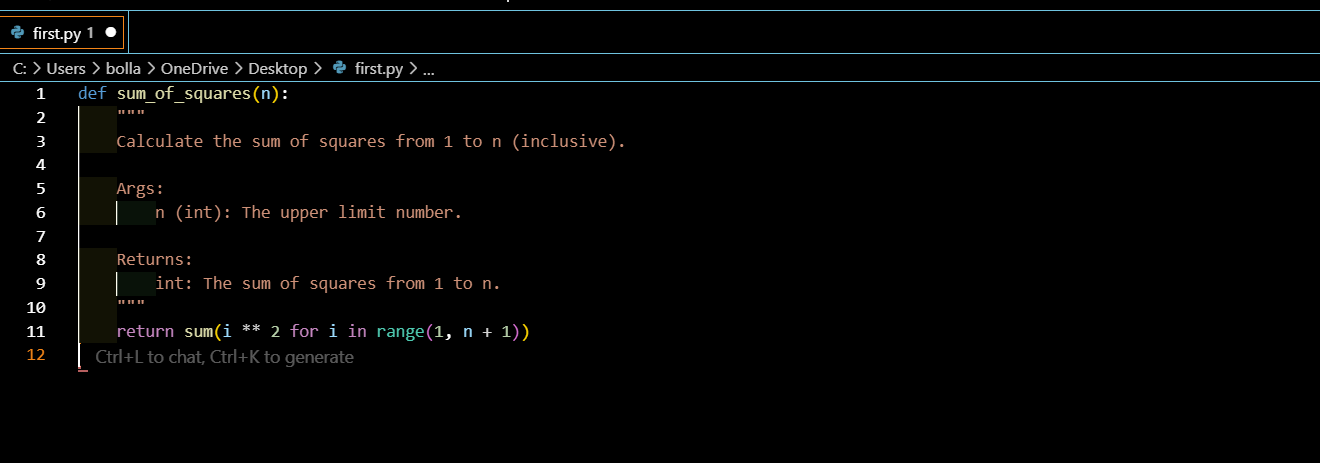
**TASK 4:**

**Installation of cursor ai:**

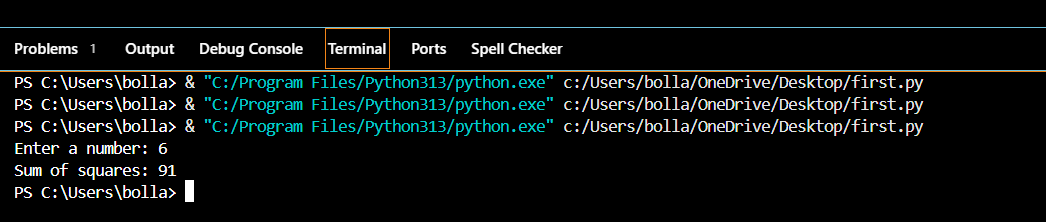


**PROMPT**: Write a python function to calculate sum of squares of a number.

****

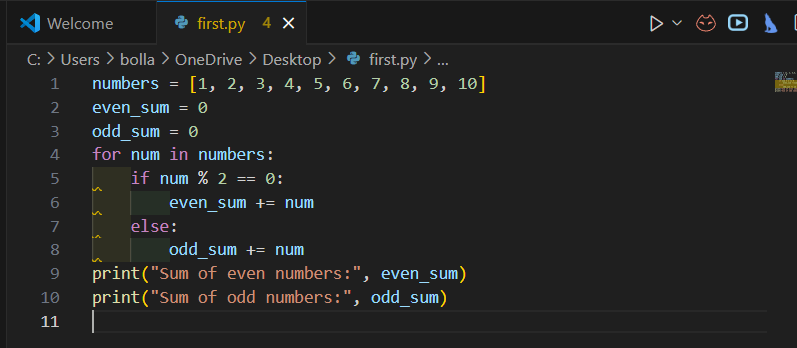
**Code:  
**

**Output:**

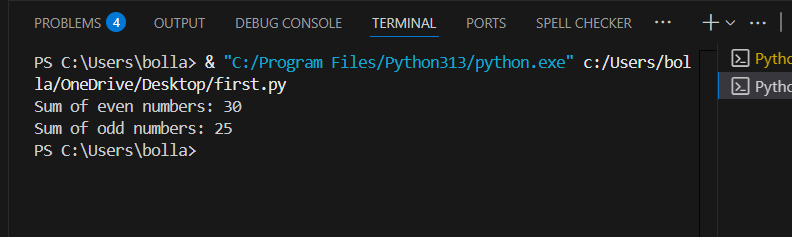
****

**Task 5:**

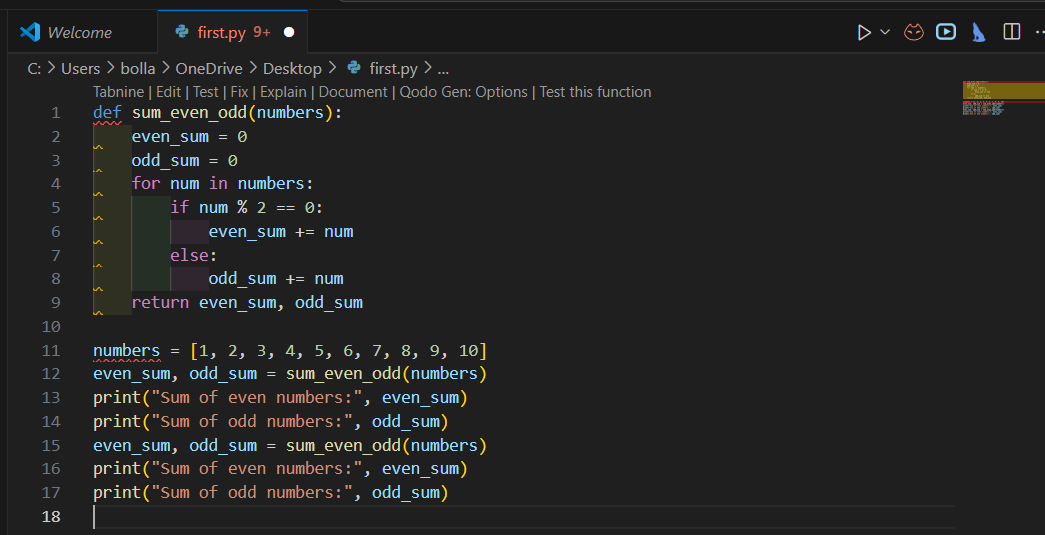
**Original Code:**

****

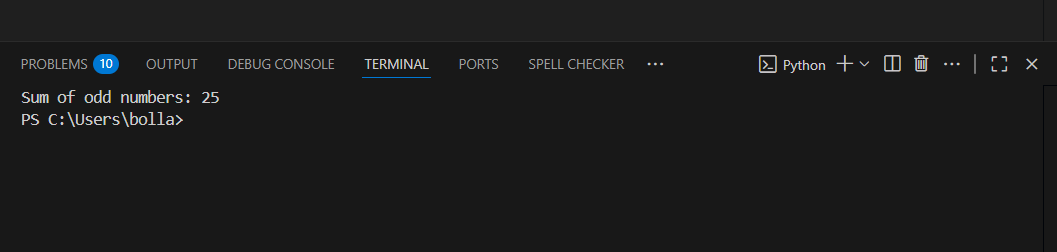
**Output:**

****

**REFACTORED CODE:**

****

**Output:**

****

COMPARISION TABLE

| **Feature** | **Original Code** | **Refactored Code** |
| --- | --- | --- |
| Structure | Simple loop | Function + List Comprehension |
| Length | 10 lines | 8 lines |
| Readability | Easier for beginners | Compact and concise |
| Reusability | No | Yes (function can be reused) |
| Efficiency | O(n) | O(n) (same, but cleaner syntax) |

**OBSERVATIONS:**

* The **original version** is more beginner-friendly but longer.
* The **refactored version** is more concise, reusable, and Pythonic.
* Both give the same output, but AI’s approach is better for professional coding standards.