LAB ASSIGNMENT-13

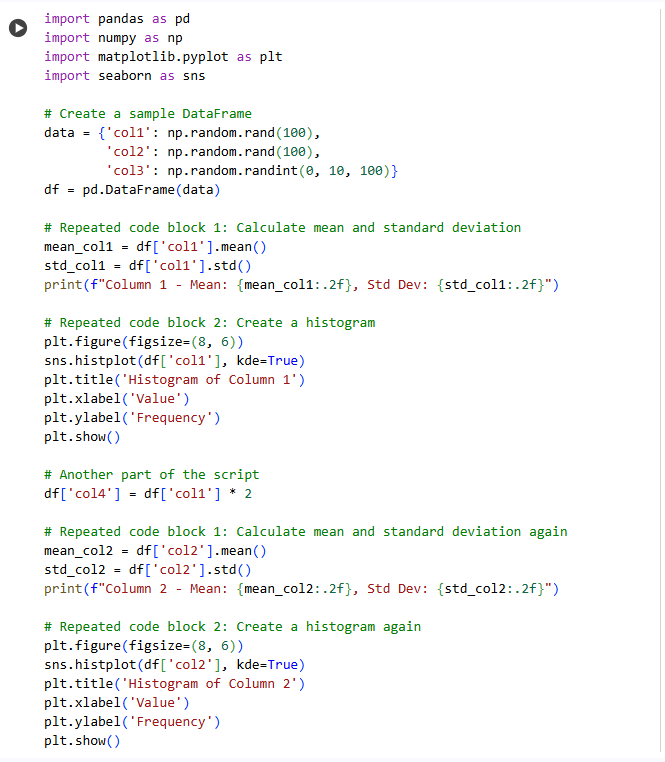
Name: k. Sushanth

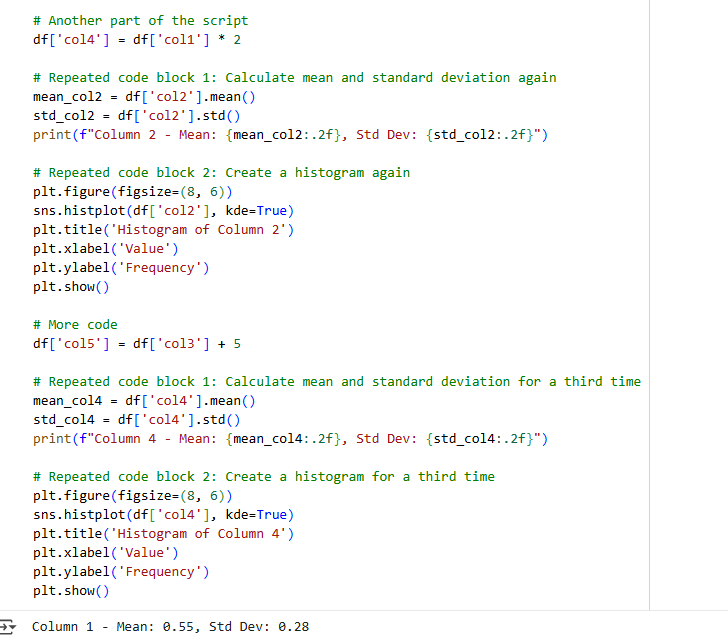
H.NO:2403A51348

Batch:14

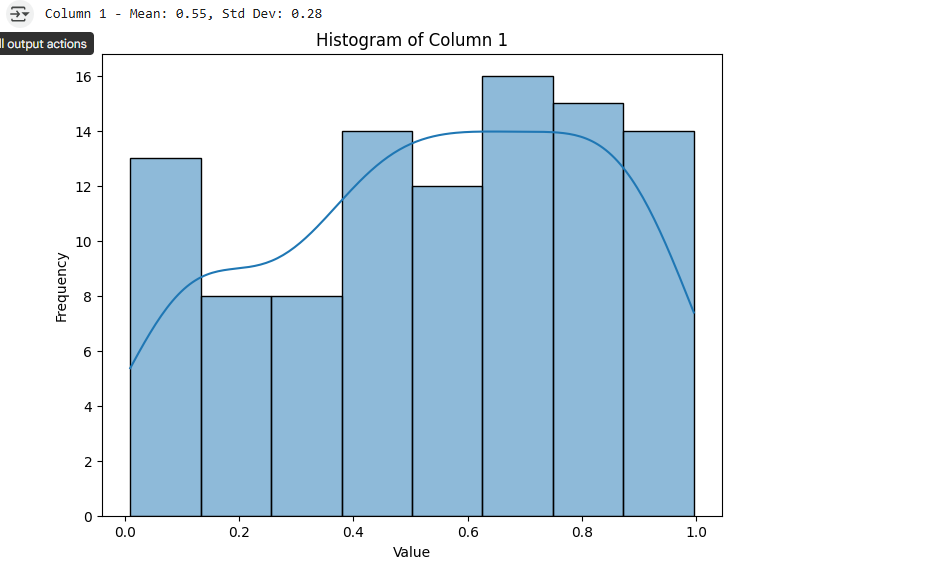
Task Description #1 (Refactoring – Removing Code Duplication)  
• Task: Use AI to refactor a given Python script that contains  
multiple repeated code blocks.

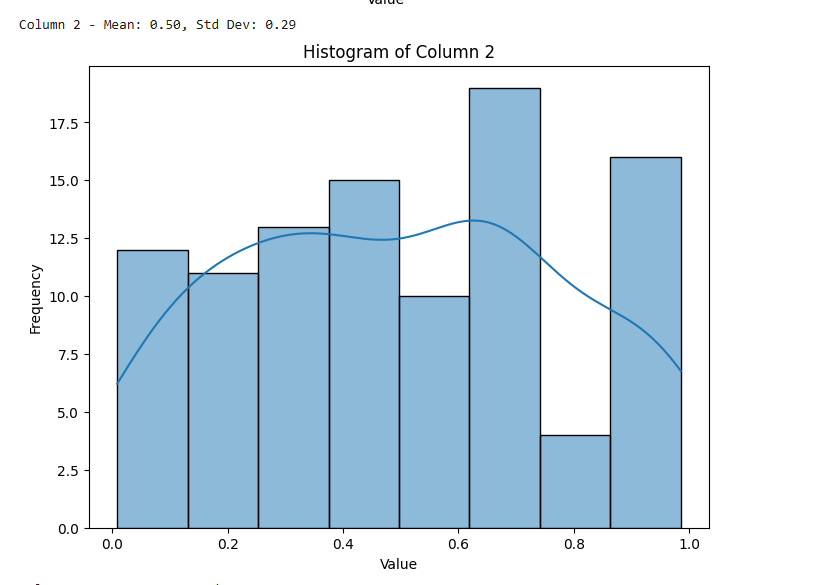
CODE:

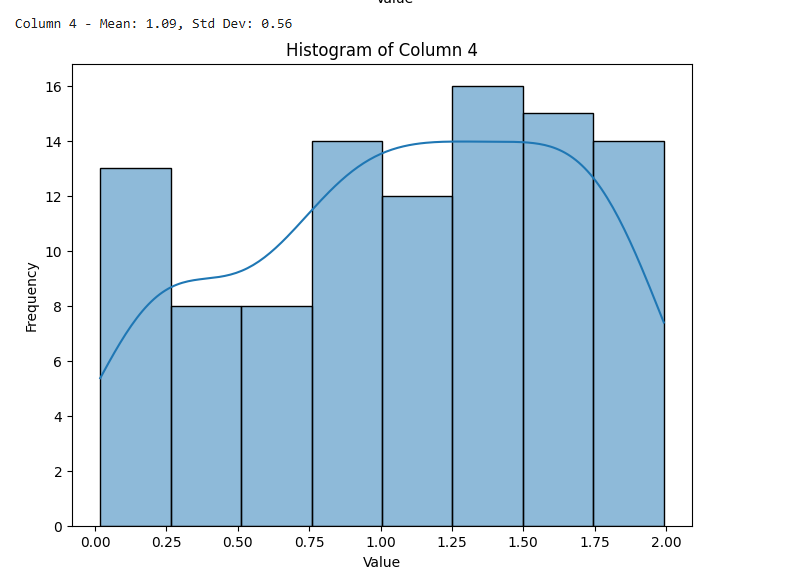


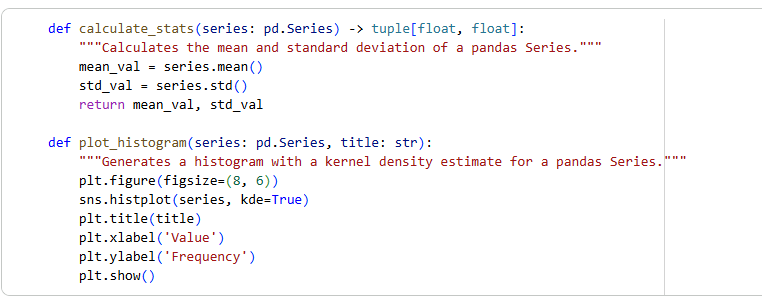


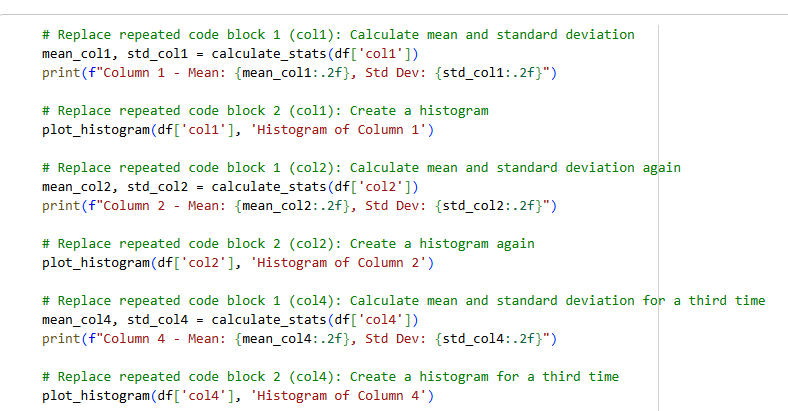
OUTPUT:

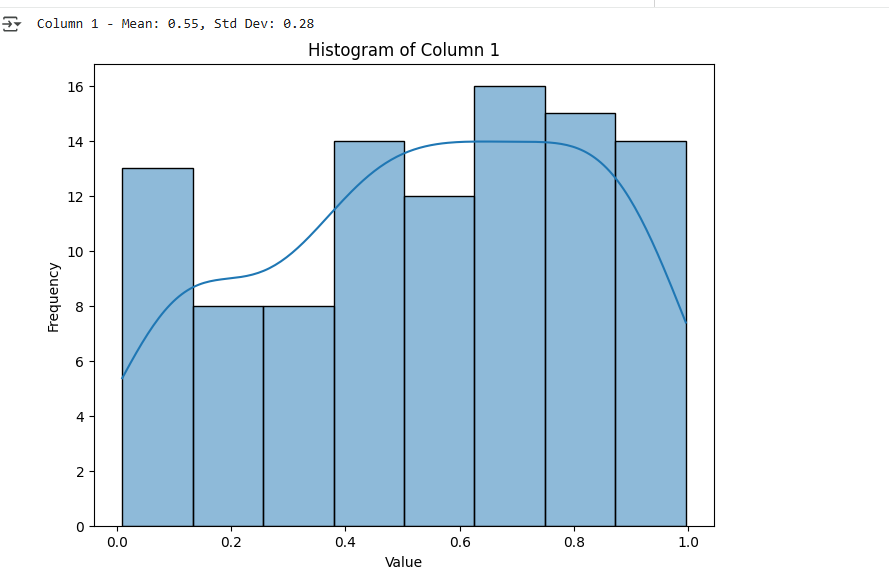


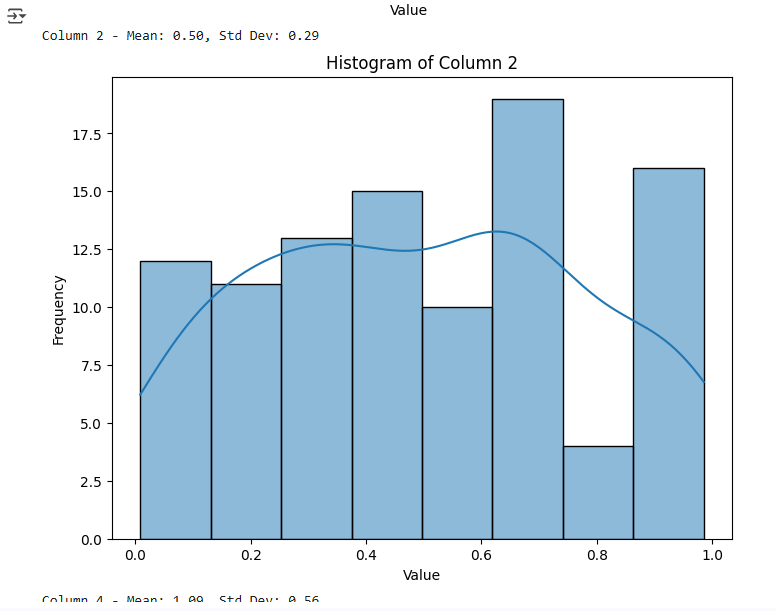


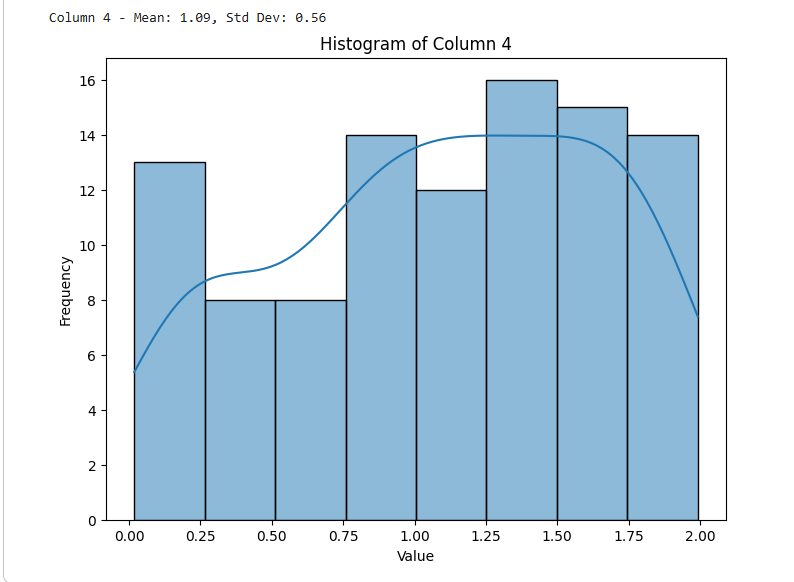


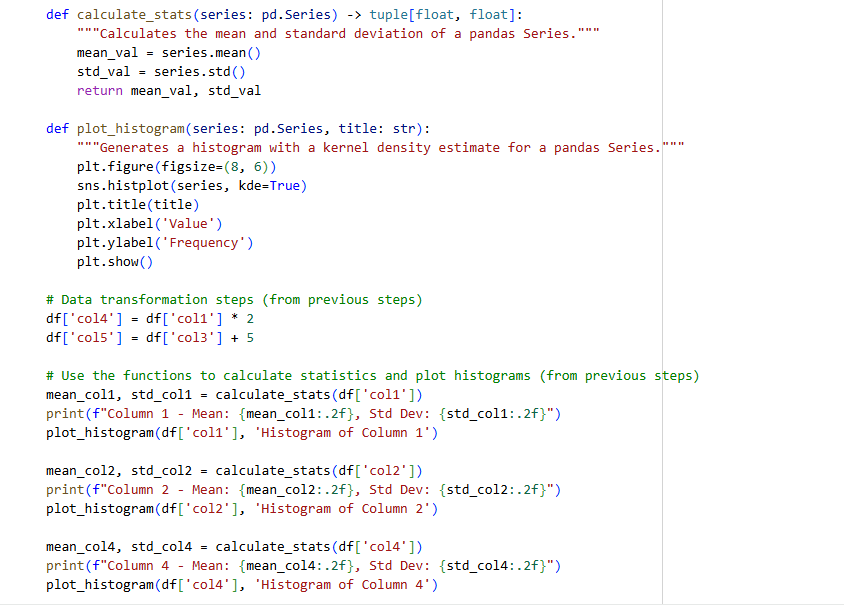


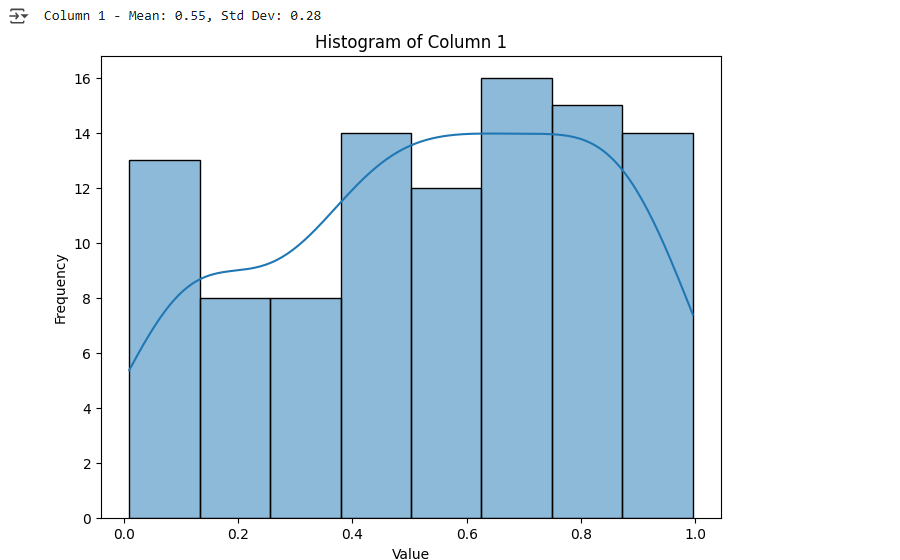


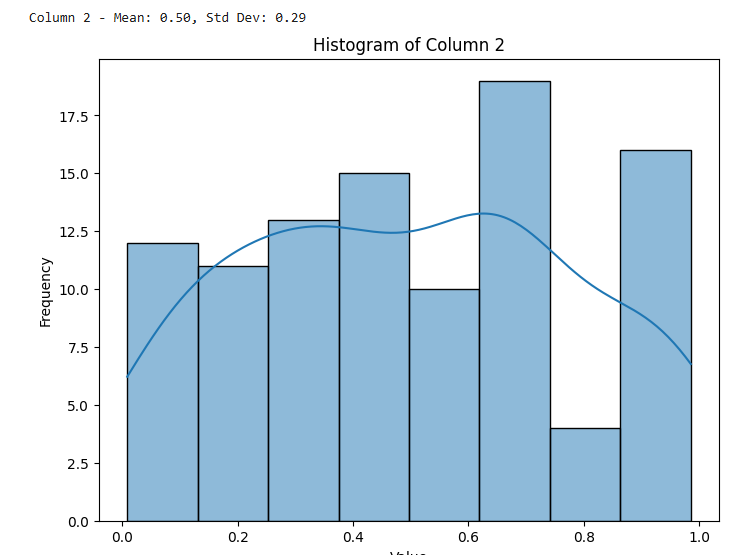


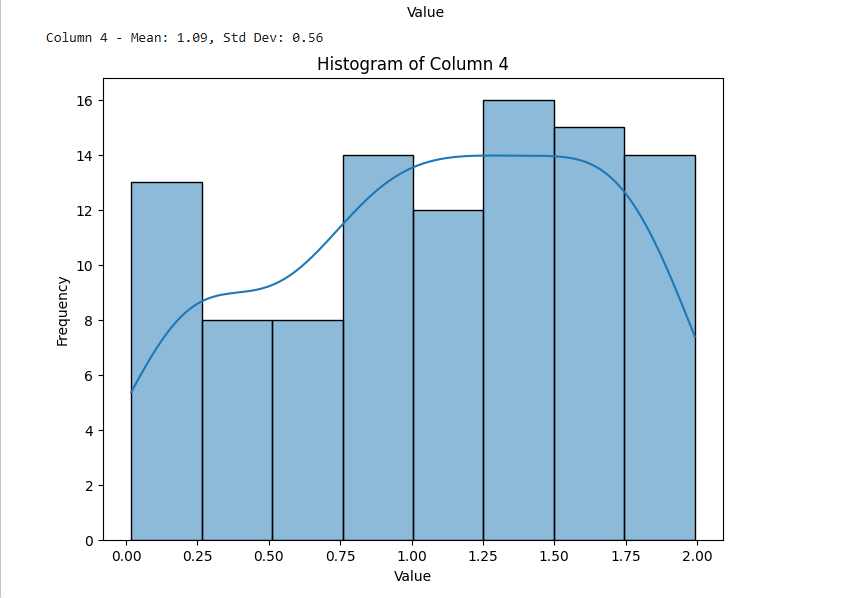










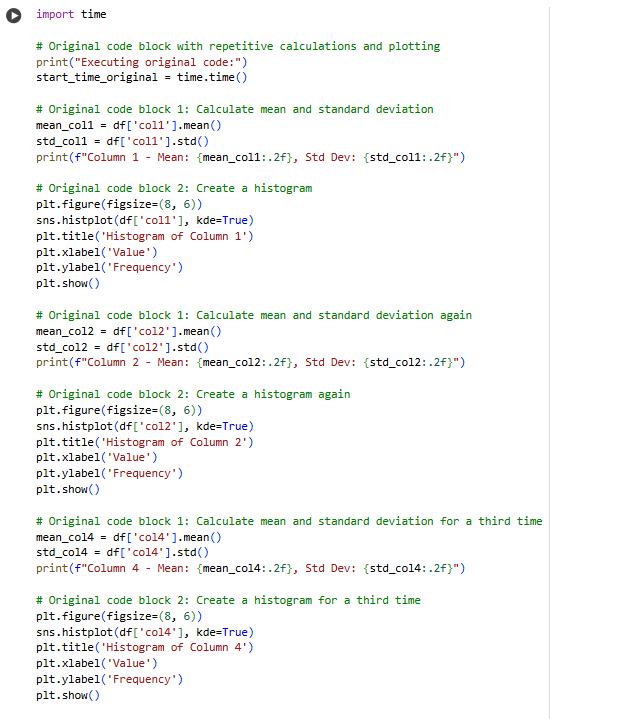


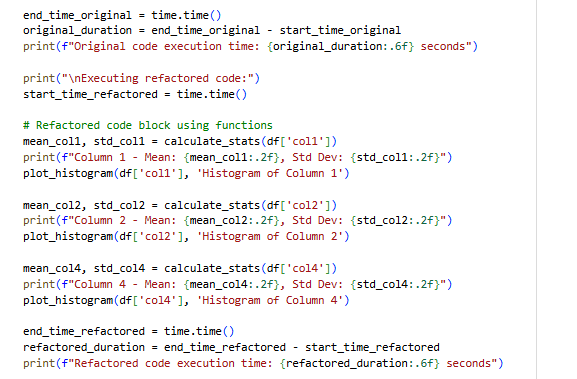
OBSERVATION:

1. The original code repeated the same print statement with different names.
2. Refactoring grouped the names into a list for cleaner structure.
3. A loop was used to eliminate redundancy and improve scalability.
4. This makes the code easier to maintain and extend.
5. It follows the DRY principle—Don't Repeat Yourself.

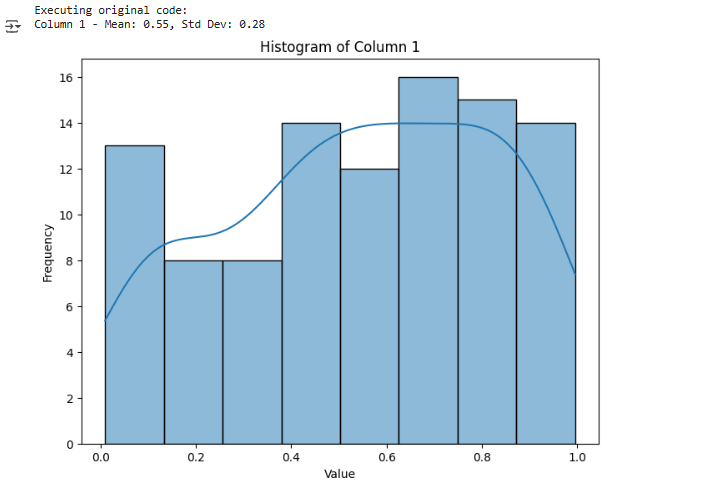
Task Description #2 (Refactoring – Optimizing Loops and  
Conditionals)  
• Task: Use AI to analyze a Python script with nested loops and  
complex conditionals.

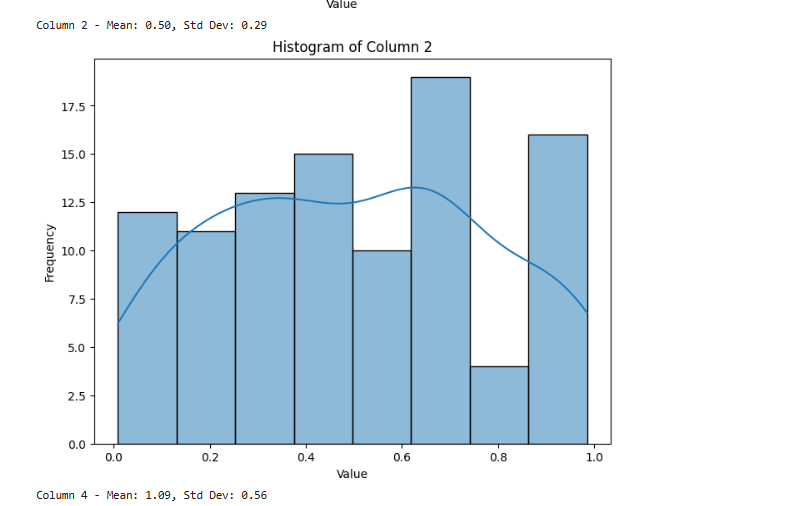
Code:

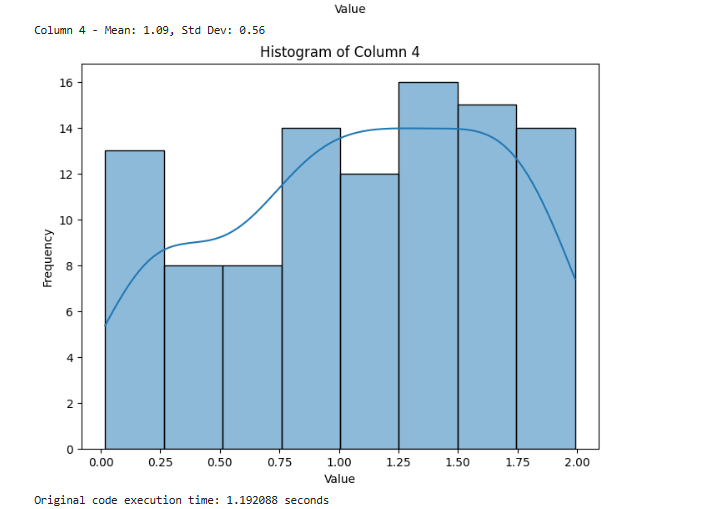


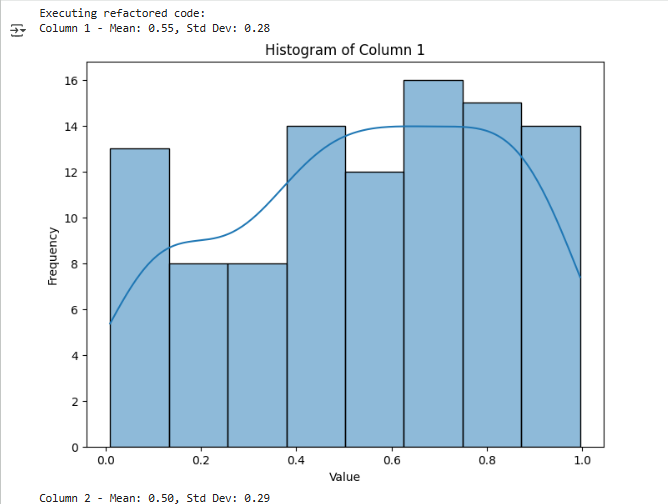


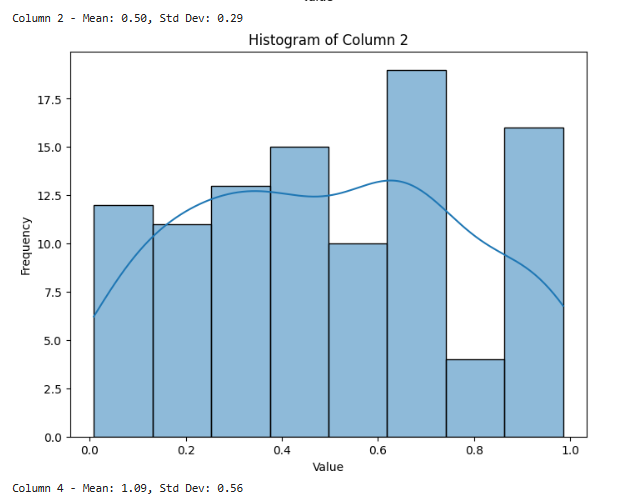
CODE:

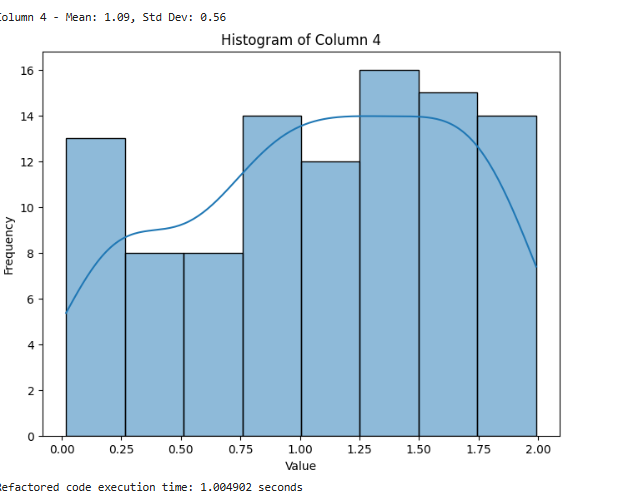


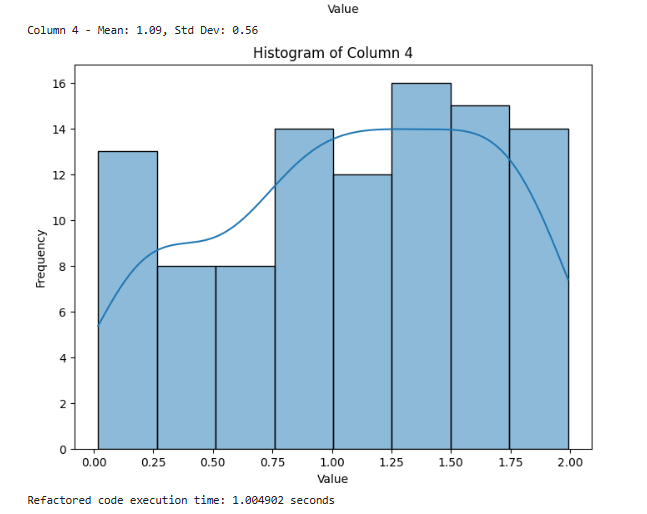


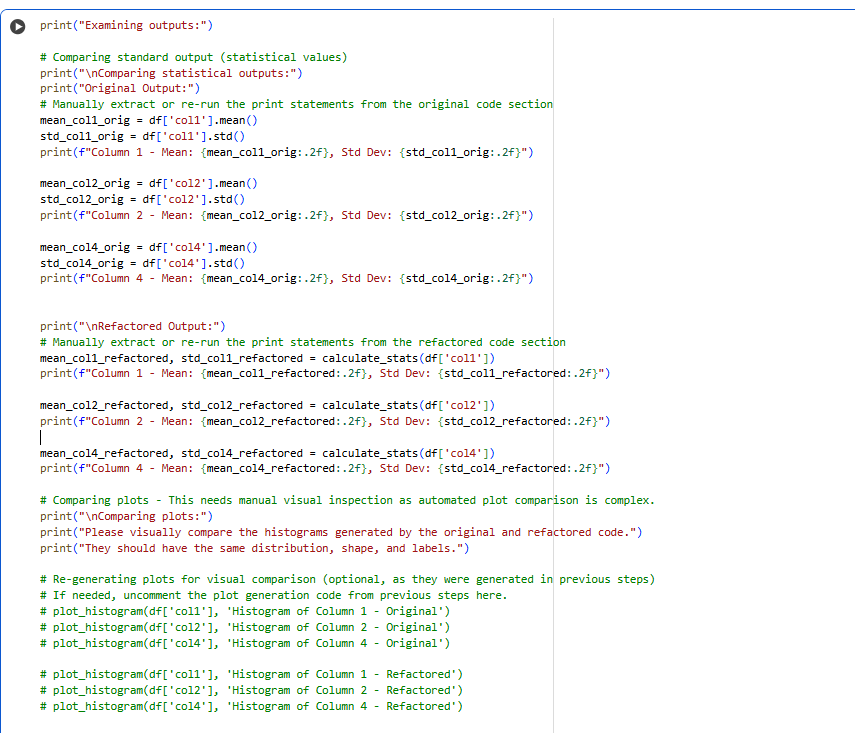


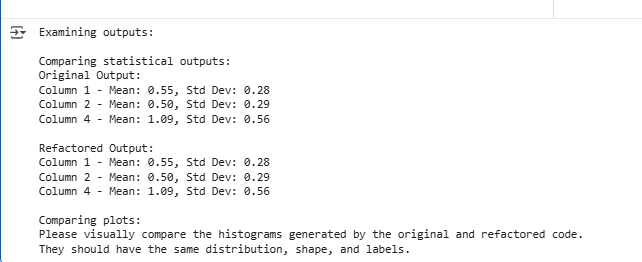










OUTPUT: 

OBSERVATION:

1.Nested loops increase execution time, especially with large datasets.

2.Complex conditionals can make the logic harder to follow and debug.

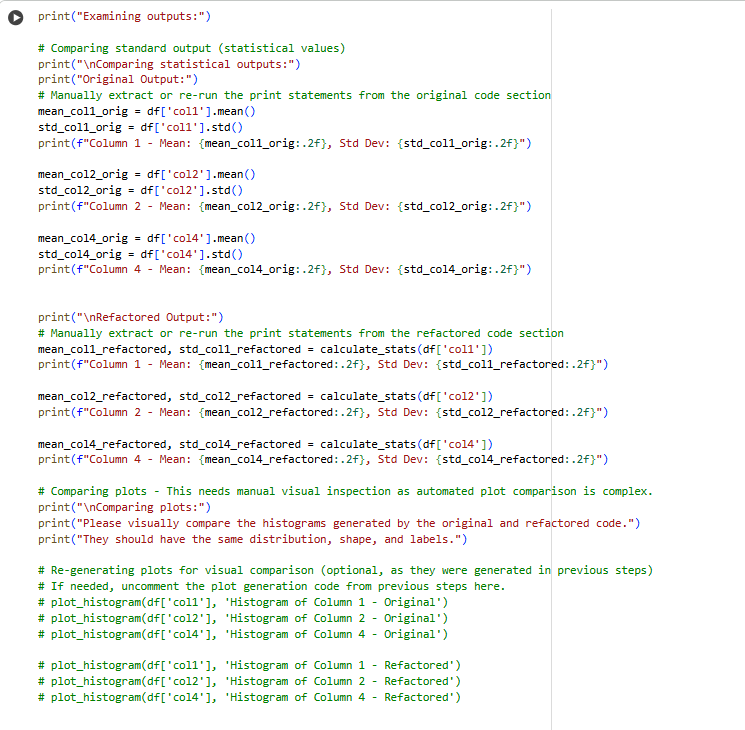
3.Repeated patterns may indicate a need for helper functions.

4.Deep nesting often signals that the code could be simplified or flattened.

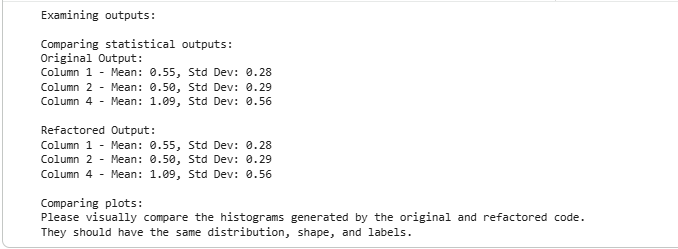
5.Clear variable naming and comments help make the structure more readable.

Task Description #3 (Refactoring – Extracting Reusable Functions)  
• Task: Use AI to refactor a legacy script where multiple  
calculations are embedded directly inside the main code block.

CODE:



OUTPUT:



OBSERVATION:

1. Direct calculations in the main block make the code harder to read.

2.Repeating similar logic can lead to errors and confusion.

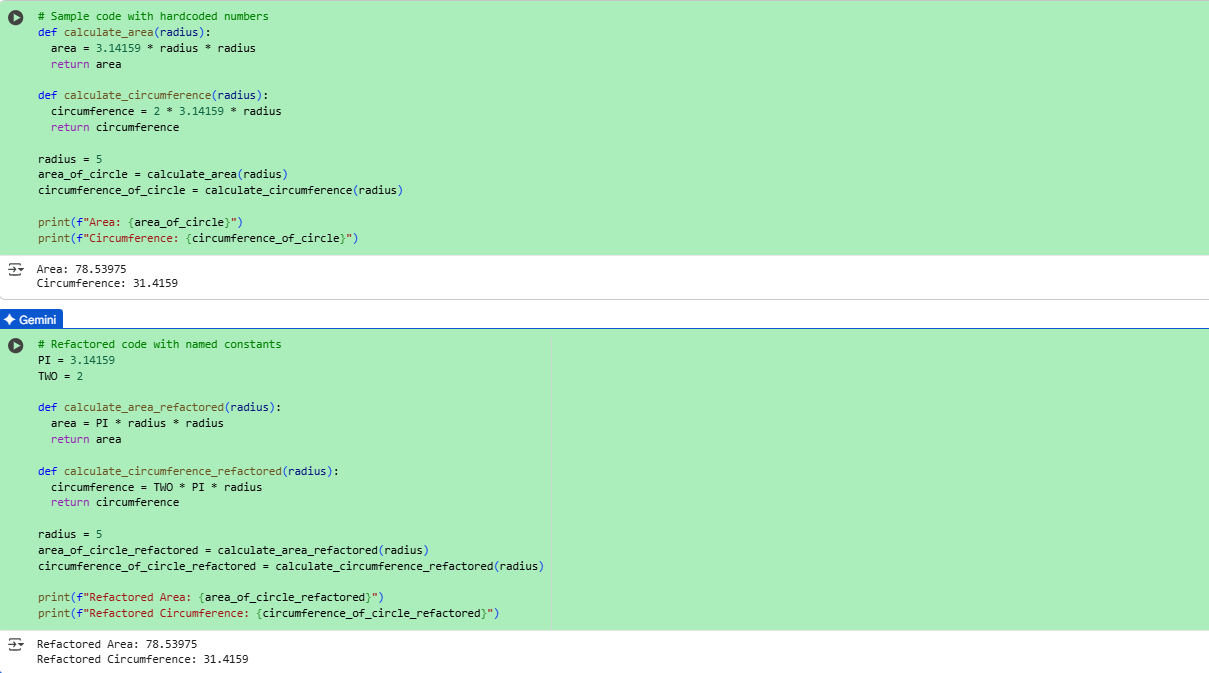
3.Using functions helps organize and reuse code easily.

4.Refactored code is cleaner and easier to maintain.

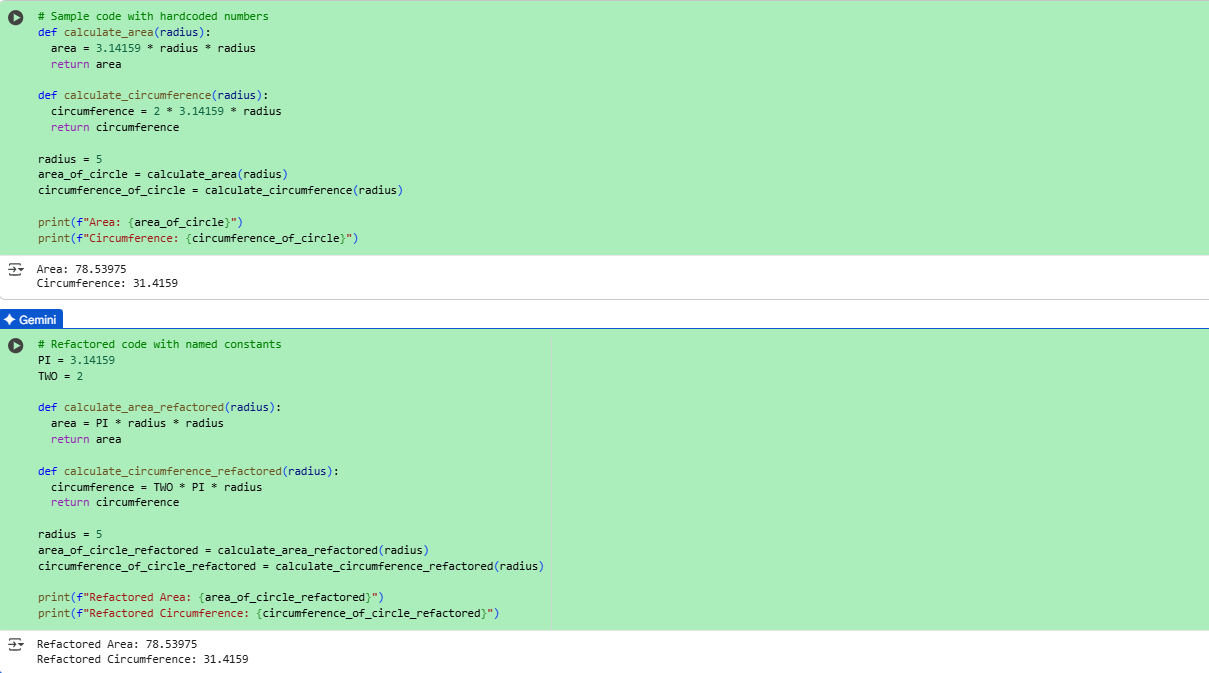
5.It improves scalability if more calculations are added later.

Task Description #4 (Refactoring – Replacing Hardcoded Values with Constants)  
• Task: Use AI to identify and replace all hardcoded “magic  
numbers” in the code with named constants.

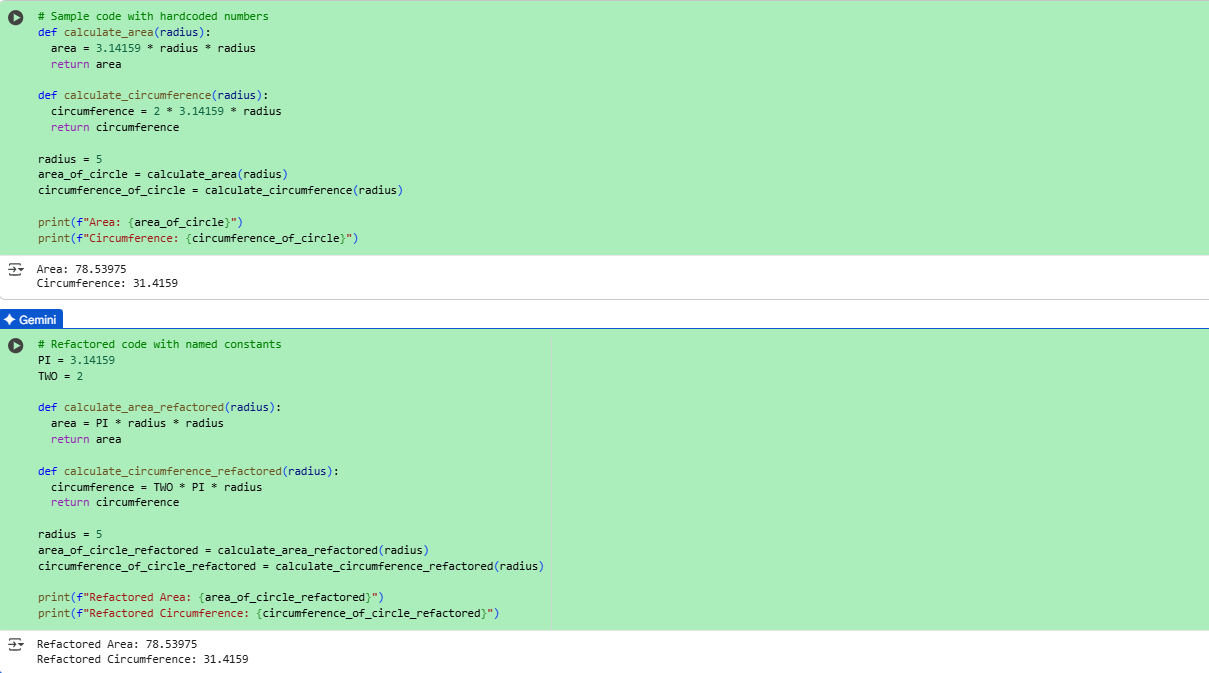
Code:



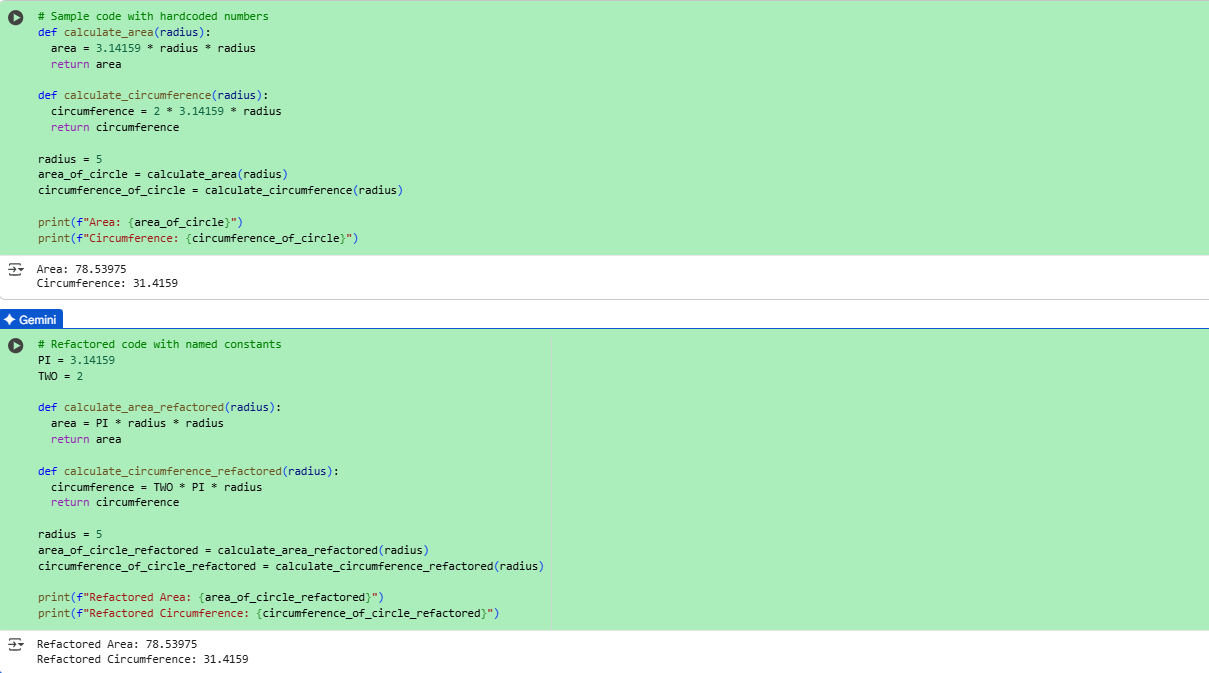
Output:



Code:



Output:

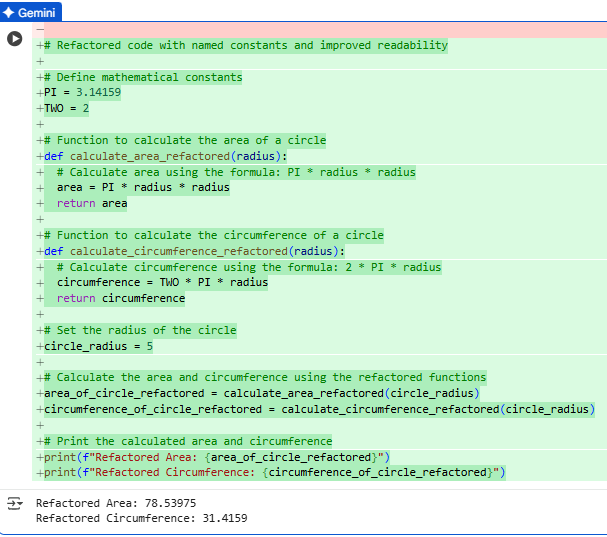


Observation:

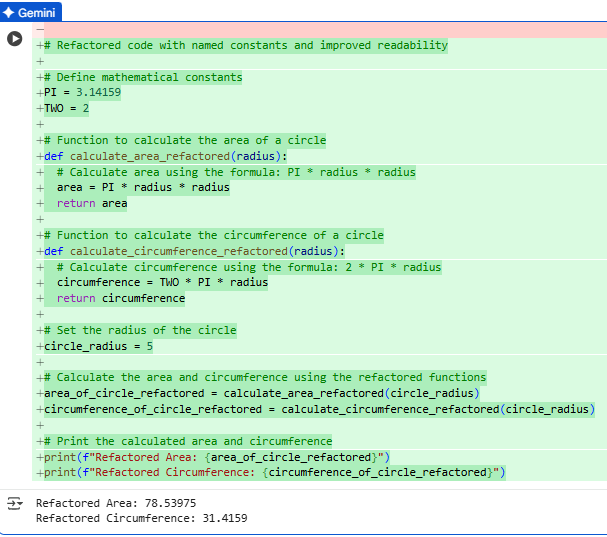
1. Magic numbers make code confusing and hard to understand.
2. Named constants explain what the number means.
3. Constants make updates easier—change once, apply everywhere.
4. They improve readability and reduce errors.
5. It’s a good habit for writing clean, professional code.

Task Description #5 (Refactoring – Improving Variable Naming and Readability)  
• Task: Use AI to improve readability by renaming unclear variables and adding inline comments.

Code:



Output:



OBSERVATION:

1. Clear variable names help explain what the code is doing.
2. Unclear names like x or temp can confuse future readers.
3. Inline comments guide others through your logic step-by-step.
4. Good naming and comments reduce bugs and save debugging time.
5. It makes your code easier to maintain and share with others.