

Assignment-2

M.Paramesh

H.No. : 2303A53021

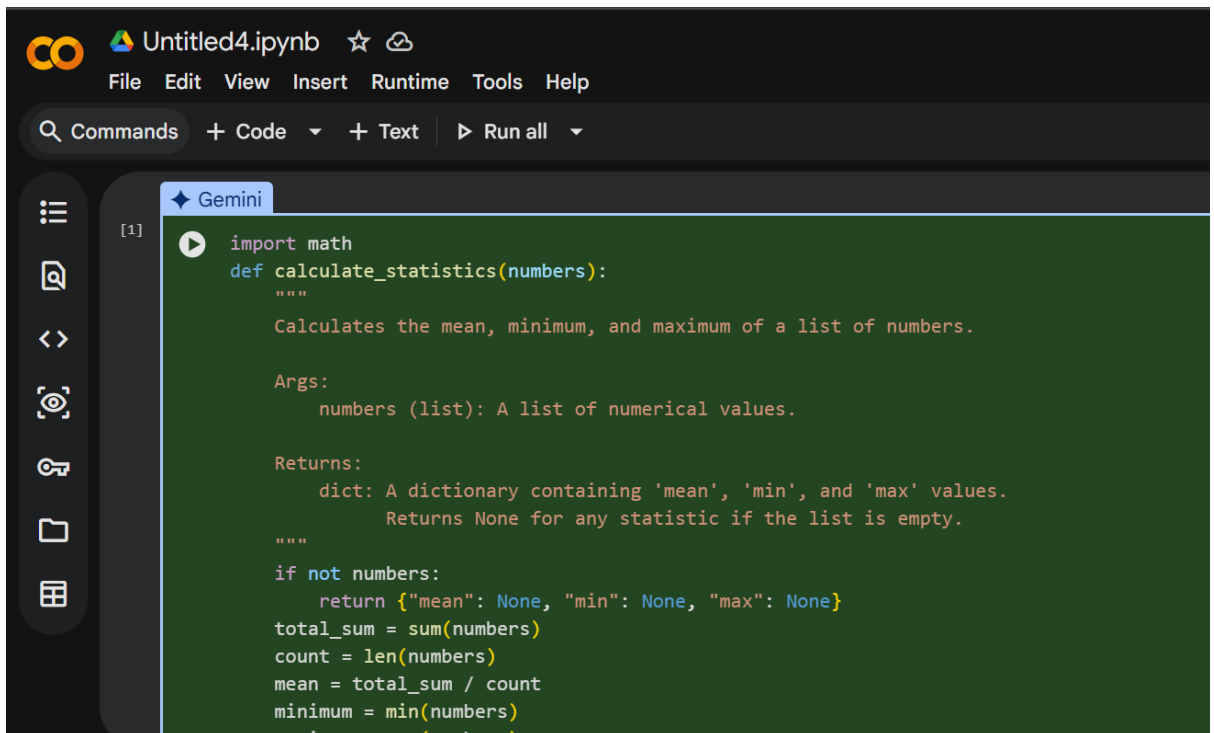
Batch : 46

Task 1 — Statistical Summary using Google Gemini (Colab)

Create a Python function that finds:

- Mean
- Minimum
- Maximum

from a list of numbers using **Gemini in Colab**.

A screenshot of the Google Colab interface. The top bar shows the Colab logo, the file name 'Untitled4.ipynb', and icons for star and share. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. A toolbar contains a search icon, 'Commands', '+ Code', '+ Text', and a 'Run all' button. On the left is a sidebar with icons for file explorer, search, and other tools. The main area is a code editor with a 'Gemini' tab. It contains a Python function definition for 'calculate_statistics' with docstrings for its purpose, arguments, and return values. The function logic includes a check for an empty list, a calculation of the mean using sum and len, and assignments for minimum and maximum values.

```
[1] import math
def calculate_statistics(numbers):
    """
    Calculates the mean, minimum, and maximum of a list of numbers.

    Args:
        numbers (list): A list of numerical values.



    Returns:
        dict: A dictionary containing 'mean', 'min', and 'max' values.
        Returns None for any statistic if the list is empty.
    """
    if not numbers:
        return {"mean": None, "min": None, "max": None}
    total_sum = sum(numbers)
    count = len(numbers)
    mean = total_sum / count
    minimum = min(numbers)
    maximum = max(numbers)
```





```
if not numbers:
    return {"mean": None, "min": None, "max": None}
total_sum = sum(numbers)
count = len(numbers)
mean = total_sum / count
minimum = min(numbers)
maximum = max(numbers)

return {"mean": mean, "min": minimum, "max": maximum}






# Example usage:
my_list = [10, 20, 5, 15, 30, 25]
stats = calculate_statistics(my_list)
print(f"For the list {my_list}:")
print(f"  Mean: {stats['mean']}")
print(f"  Minimum: {stats['min']}")
print(f"  Maximum: {stats['max']}")


empty_list = []
empty_stats = calculate_statistics(empty_list)
print(f"\nFor an empty list:")
```

s  Terminal 

  Untitled4.ipynb  

File Edit View Insert Runtime Tools Help

 Commands  + Code  + Text  Run all 

[1] 

```
print(f"  Statistics: {empty_stats}")




single_number_list = [42]
single_stats = calculate_statistics(single_number_list)
print(f"\nFor a list with a single number {single_number_list}:")
print(f"  Mean: {single_stats['mean']}")
print(f"  Minimum: {single_stats['min']}")
print(f"  Maximum: {single_stats['max']}")
```

...

```
For the list [10, 20, 5, 15, 30, 25]:
  Mean: 17.5
  Minimum: 5
  Maximum: 30

For an empty list:
  Statistics: {'mean': None, 'min': None, 'max': None}

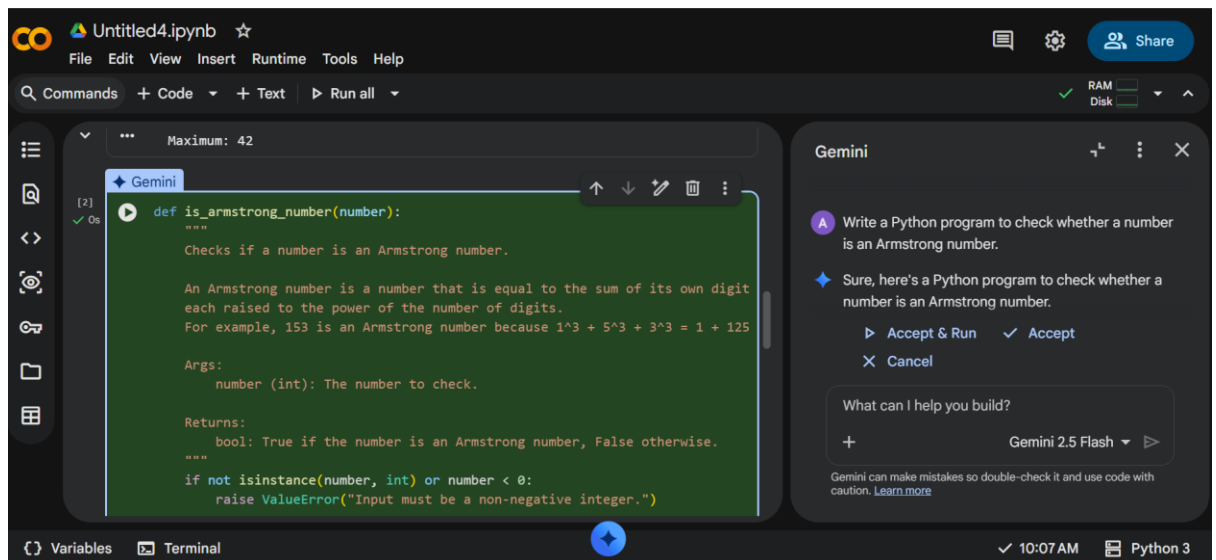
For a list with a single number [42]:
  Mean: 42.0
  Minimum: 42
  Maximum: 42
```

 Variables  Terminal 

Task 2 — Armstrong Number (Gemini vs Copilot)

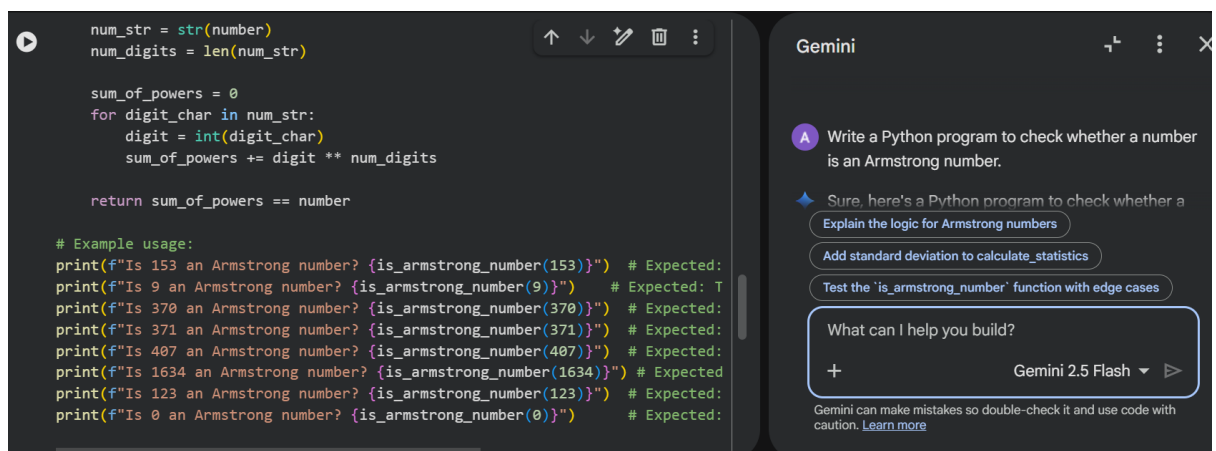
Generate Armstrong number checker using:

- Gemini
- GitHub Copilot ,Then compare them.



The screenshot shows a Jupyter Notebook titled 'Untitled4.ipynb'. The code cell contains a function `is_armstrong_number` that checks if a number is an Armstrong number. The function takes an integer `number` and returns a boolean. It includes a docstring and a `ValueError` for non-negative integers. The Gemini chat window on the right shows a prompt to write a Python program to check if a number is an Armstrong number, and the response is a Python program to check whether a number is an Armstrong number.

```
def is_armstrong_number(number):  
    """  
    Checks if a number is an Armstrong number.  
  
    An Armstrong number is a number that is equal to the sum of its own digit  
    each raised to the power of the number of digits.  
    For example, 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 1 + 125  
  
    Args:  
        number (int): The number to check.  
  
    Returns:  
        bool: True if the number is an Armstrong number, False otherwise.  
    """  
    if not isinstance(number, int) or number < 0:  
        raise ValueError("Input must be a non-negative integer.")
```



The screenshot shows the implementation of the `is_armstrong_number` function. It converts the number to a string, iterates over each digit, calculates the sum of each digit raised to the power of the number of digits, and returns the result. The Gemini chat window on the right shows a prompt to write a Python program to check if a number is an Armstrong number, and the response is a Python program to check whether a number is an Armstrong number.

```
num_str = str(number)  
num_digits = len(num_str)  
  
sum_of_powers = 0  
for digit_char in num_str:  
    digit = int(digit_char)  
    sum_of_powers += digit ** num_digits  
  
return sum_of_powers == number  
  
# Example usage:  
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}") # Expected:  
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}") # Expected: T  
print(f"Is 370 an Armstrong number? {is_armstrong_number(370)}") # Expected:  
print(f"Is 371 an Armstrong number? {is_armstrong_number(371)}") # Expected:  
print(f"Is 407 an Armstrong number? {is_armstrong_number(407)}") # Expected:  
print(f"Is 1634 an Armstrong number? {is_armstrong_number(1634)}") # Expected:  
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}") # Expected:  
print(f"Is 0 an Armstrong number? {is_armstrong_number(0)}") # Expected:
```

```
print(f"Is 371 an Armstrong number? {is_armstrong_number(371)}") # Expected: True
print(f"Is 407 an Armstrong number? {is_armstrong_number(407)}") # Expected: True
print(f"Is 1634 an Armstrong number? {is_armstrong_number(1634)}") # Expected: True
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}") # Expected: False
print(f"Is 0 an Armstrong number? {is_armstrong_number(0)}") # Expected: True
```

```
... Is 153 an Armstrong number? True
Is 9 an Armstrong number? True
Is 370 an Armstrong number? True
Is 371 an Armstrong number? True
Is 407 an Armstrong number? True
Is 1634 an Armstrong number? True
Is 123 an Armstrong number? False
Is 0 an Armstrong number? True
```

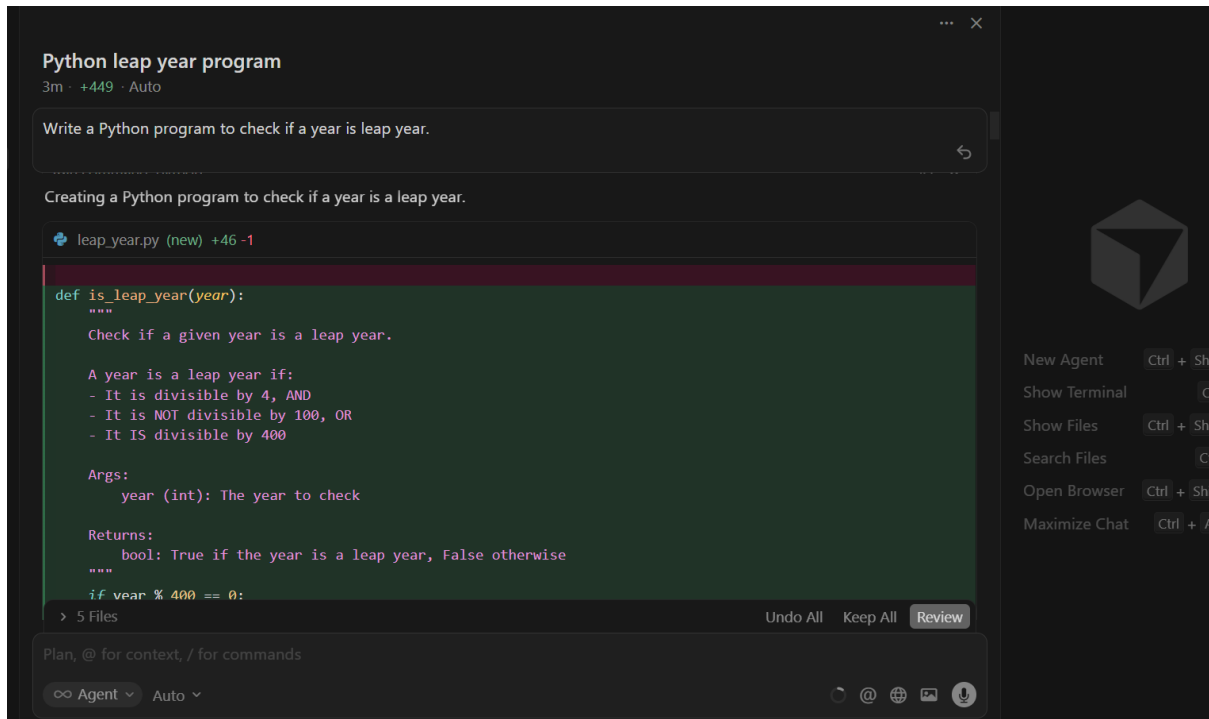
+ Code + Text

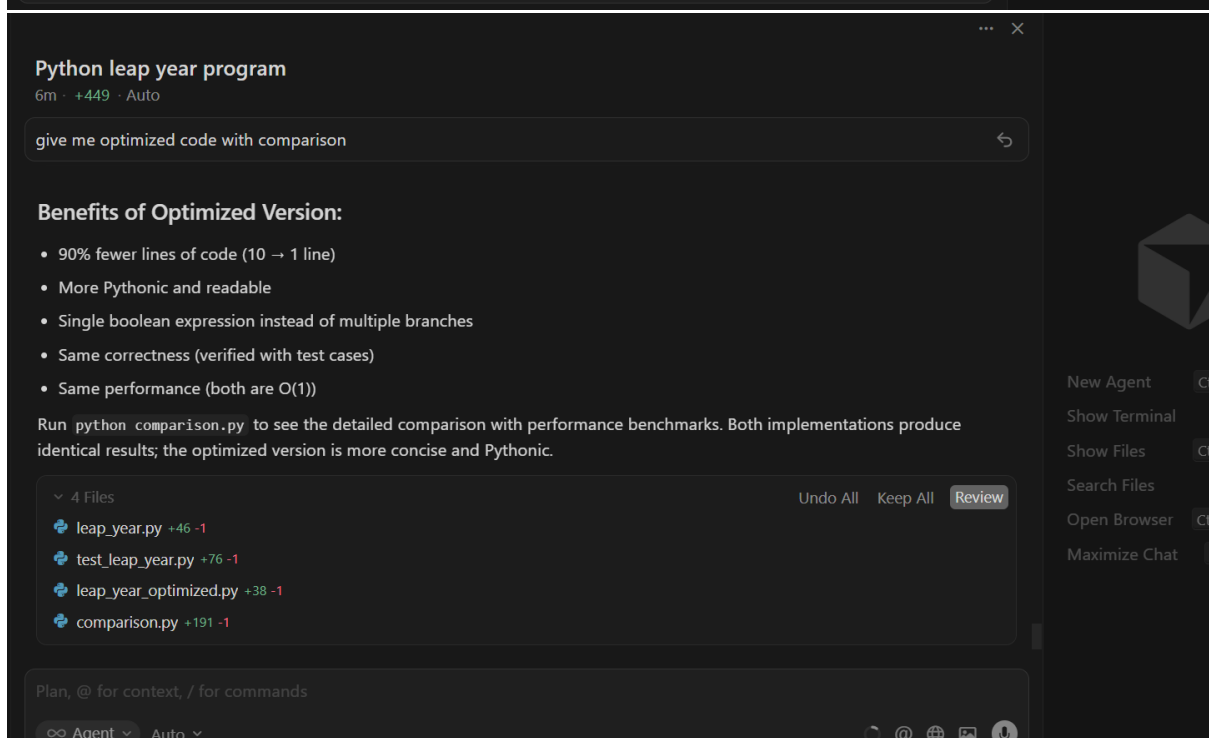
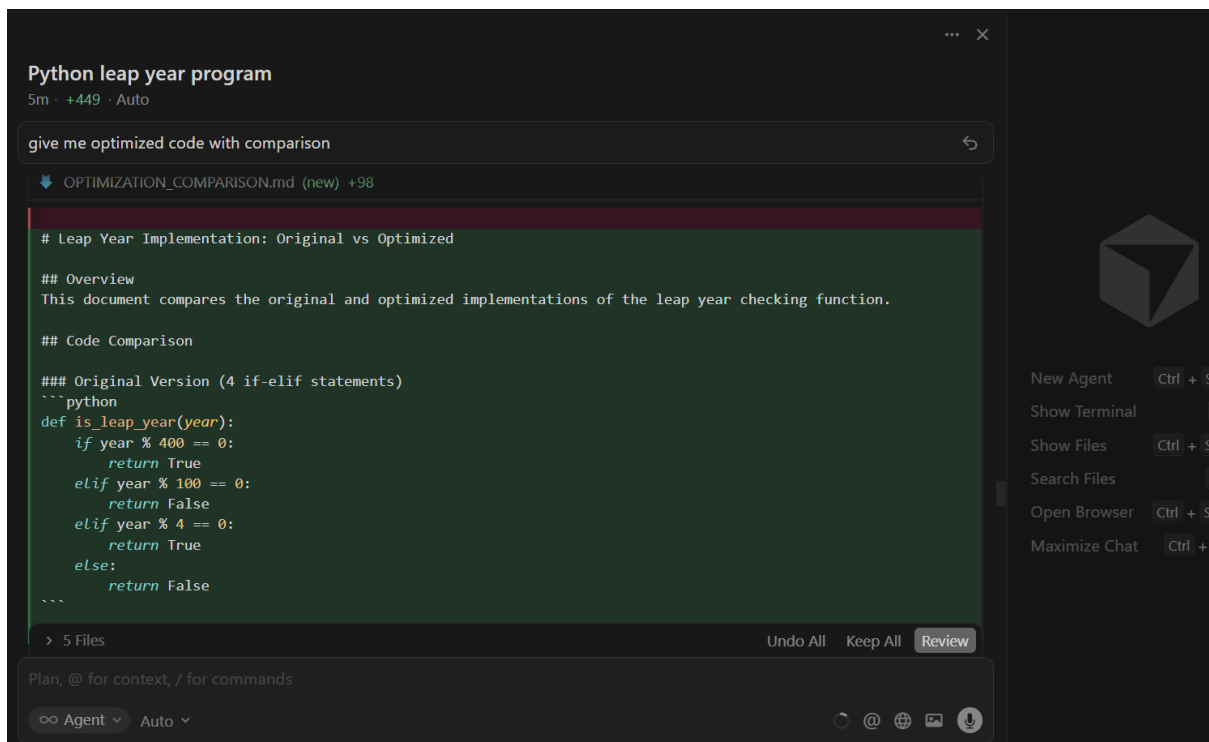
Terminal

Feature	Gemini	Copilot
Logic	Uses step-by-step digit extraction and power calculation, easy to follow	Uses the same core logic but often written in a more compact style
Code Clarity	Well-structured code with meaningful variable names and comments	Shorter code, fewer comments, more developer-style
Readability	Very beginner-friendly and easy to understand	Slightly less verbose but clean and professional

Task 3 — Leap Year using Cursor AI

Generate leap year program using **two different prompts**





Task 4 — Odd/Even Sum (Manual + AI Refactor)

Manual Code

```

# Tuple of numbers
t = (10, 15, 20, 25, 30, 35)

even_sum = 0
odd_sum = 0

# Loop through each number
for num in t:
    if num % 2 == 0:
        even_sum += num
    else:
        odd_sum += num

print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)

```

Output:

```

Sum of Even Numbers: 60
Sum of Odd Numbers: 75

```

AI Refactored Code (Optimized Version)

```

def calculate_even_odd_sum(numbers):
    """Returns the sum of even and odd numbers from a tuple."""
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum

numbers = (10, 15, 20, 25, 30, 35)
even, odd = calculate_even_odd_sum(numbers)

print(f"Sum of Even Numbers: {even}")
print(f"Sum of Odd Numbers: {odd}")

```

```

Sum of Even Numbers: 60
Sum of Odd Numbers: 75

```

Output:

