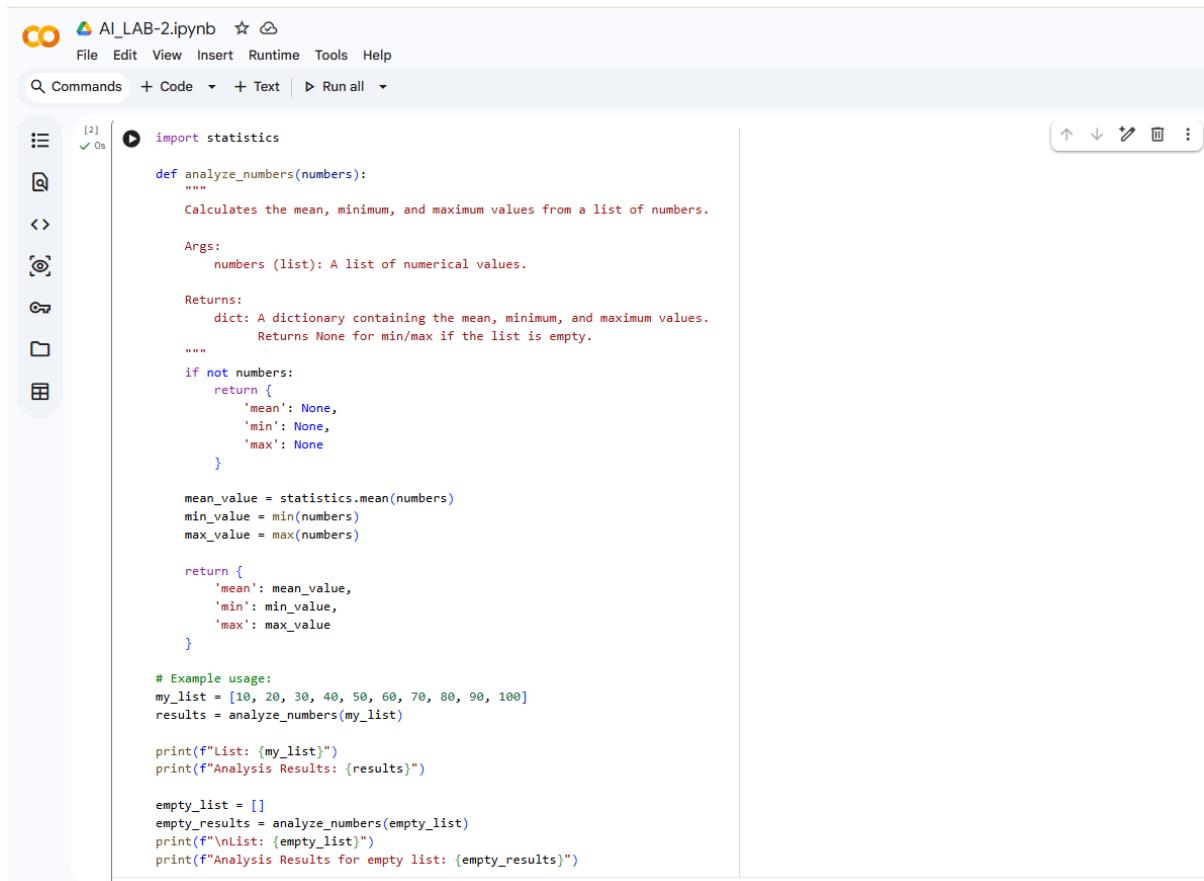


## ASSIGNMENT-2

### Task 1: Statistical Summary for Survey Data

**Prompt:** Generate a python code function that takes a list of numbers and calculates mean, minimum, and maximum values

**Code:**



```
AI_LAB-2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[2] ✓ 0s
import statistics

def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum values from a list of numbers.

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

    Returns:
        dict: A dictionary containing the mean, minimum, and maximum values.
        Returns None for min/max if the list is empty.
    """
    if not numbers:
        return {
            'mean': None,
            'min': None,
            'max': None
        }
    mean_value = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

    return {
        'mean': mean_value,
        'min': min_value,
        'max': max_value
    }

# Example usage:
my_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
results = analyze_numbers(my_list)

print(f"List: {my_list}")
print(f"Analysis Results: {results}")

empty_list = []
empty_results = analyze_numbers(empty_list)
print(f"\nList: {empty_list}")
print(f"Analysis Results for empty list: {empty_results}")
```

**Output:**

```
... List: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
      Analysis Results: {'mean': 55, 'min': 10, 'max': 100}

      List: []
      Analysis Results for empty list: {'mean': None, 'min': None, 'max': None}
```

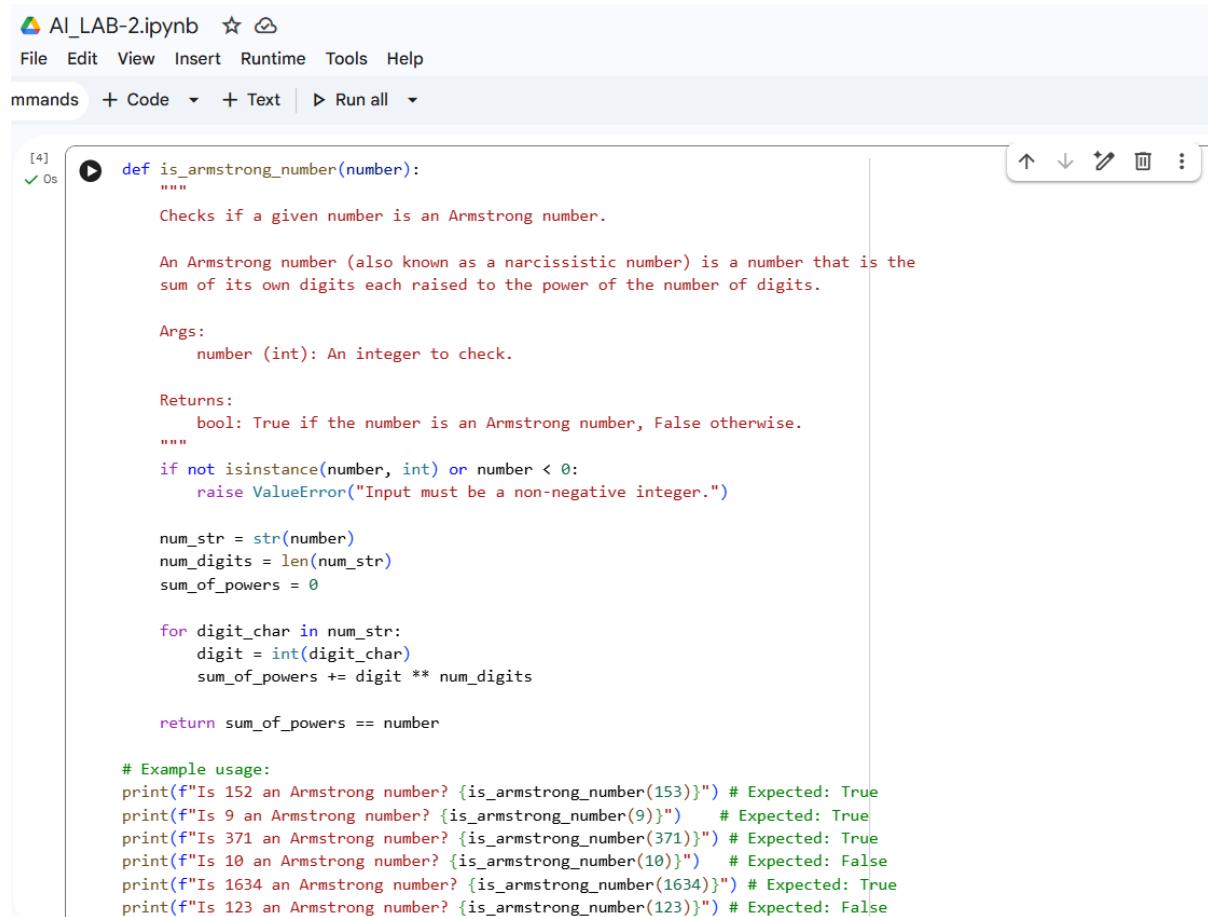
**Justification:**

- Gemini generated correct and efficient Python code using built-in functions.
- The code is simple, readable, and suitable for beginner-level data analysis.

### Task 2: Armstrong Number – AI Comparison

**Prompt:** give python code for Armstrong number checker

### Gemini Code:



The screenshot shows a Jupyter Notebook cell with the following code:

```
[4] ✓ 0s
def is_armstrong_number(number):
    """
    Checks if a given number is an Armstrong number.

    An Armstrong number (also known as a narcissistic number) is a number that is the sum of its own digits each raised to the power of the number of digits.

    Args:
        number (int): An integer 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.")

    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

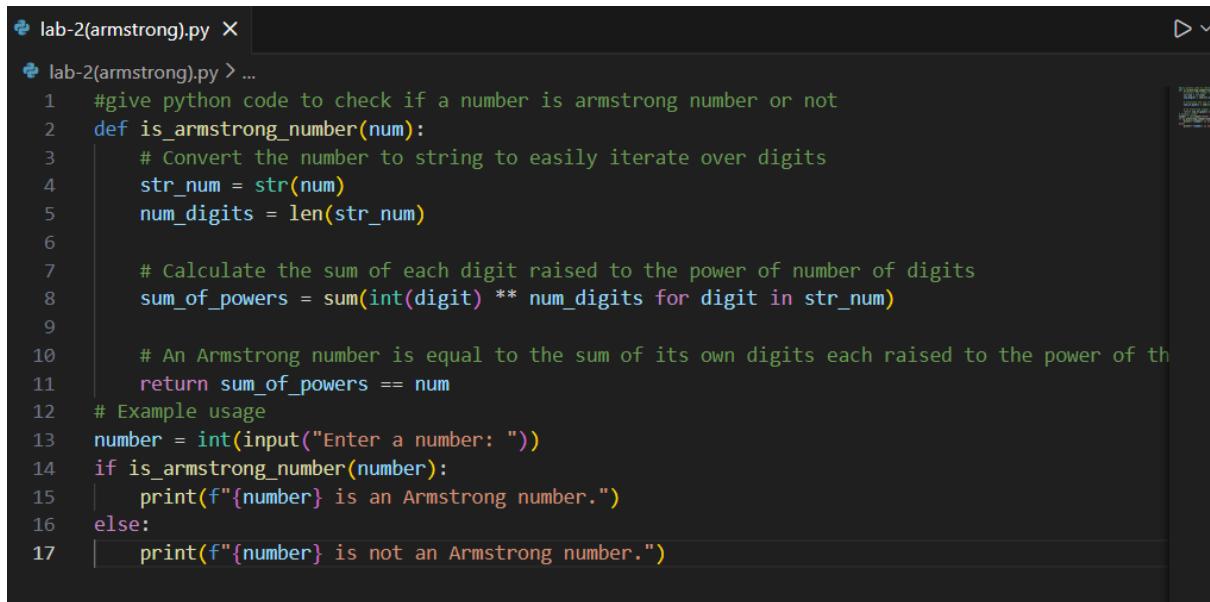
    return sum_of_powers == number

# Example usage:
print(f"Is 152 an Armstrong number? {is_armstrong_number(153)}") # Expected: True
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}") # Expected: True
print(f"Is 371 an Armstrong number? {is_armstrong_number(371)}") # Expected: True
print(f"Is 10 an Armstrong number? {is_armstrong_number(10)}") # Expected: False
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
```

### Output:

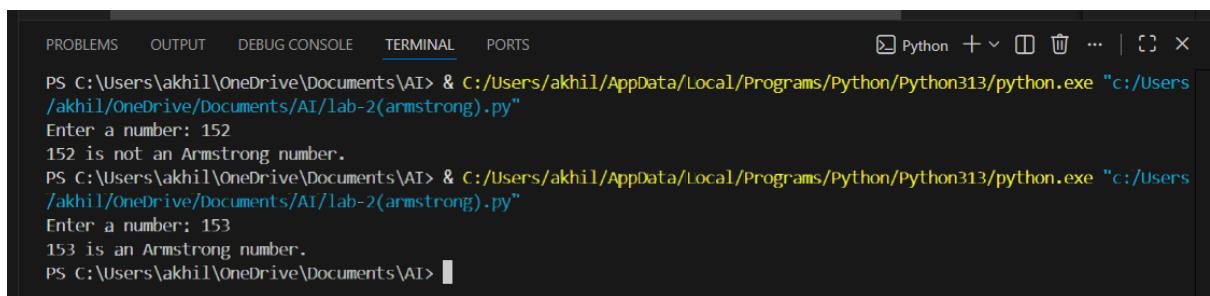
```
... Is 152 an Armstrong number? True
Is 9 an Armstrong number? True
Is 371 an Armstrong number? True
Is 10 an Armstrong number? False
Is 1634 an Armstrong number? True
Is 123 an Armstrong number? False
```

### Github copilot code:



```
lab-2/armstrong.py X
lab-2/armstrong.py > ...
1  #give python code to check if a number is armstrong number or not
2  def is_armstrong_number(num):
3      # Convert the number to string to easily iterate over digits
4      str_num = str(num)
5      num_digits = len(str_num)
6
7      # Calculate the sum of each digit raised to the power of number of digits
8      sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
9
10     # An Armstrong number is equal to the sum of its own digits each raised to the power of th
11     return sum_of_powers == num
12
13 # Example usage
14 number = int(input("Enter a number: "))
15 if is_armstrong_number(number):
16     print(f"{number} is an Armstrong number.")
17 else:
18     print(f"{number} is not an Armstrong number.)
```

## Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ... | ⌂ X
PS C:\Users\akhil\OneDrive\Documents\AI> & c:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/AI/lab-2/armstrong.py"
Enter a number: 152
152 is not an Armstrong number.
PS C:\Users\akhil\OneDrive\Documents\AI> & c:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/AI/lab-2/armstrong.py"
Enter a number: 153
153 is an Armstrong number.
PS C:\Users\akhil\OneDrive\Documents\AI>
```

## Comparison Table:

| feature     | Gemini code        | GitHub copilot      |
|-------------|--------------------|---------------------|
| logic       | with comprehension | Loop-based          |
| comments    | minimal            | Extensive docstring |
| style       | short              | detailed            |
| readability | no                 | yes                 |

## Justification:

- Gemini focuses on concise and modern Python syntax.
- Copilot uses classic looping logic, making it suitable for understanding algorithm flow.

## Task 3: Leap Year Validation Using Cursor AI

**Prompt1:** write a python code to check given year is leap year or not

## Code:

```
leap_yaer.py > ...  
leap_yaer.py > ...  
1 #write a python code to check given year is leap year or not  
2 year = int(input("Enter a year: "))  
3 if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):  
4     print(year, "is a leap year.")  
5 else:  
6     print(year, "is not a leap year.")
```

## Output:

```
Problems Output Debug Console Terminal Ports  
Enter a year: 2026  
2026 is not a leap year.  
PS C:\Users\akhil\OneDrive\Documents\AI> & C:\Users\akhil\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akhil/OneDrive/Documents/AI/leap_yaer.py  
Enter a year: 2027  
2027 is not a leap year.  
PS C:\Users\akhil\OneDrive\Documents\AI> & C:\Users\akhil\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akhil/OneDrive/Documents/AI/leap_yaer.py  
Enter a year: 2028  
2028 is a leap year.  
PS C:\Users\akhil\OneDrive\Documents\AI>
```

**Prompt2:** Generate an optimized and well commented python program to validate leap year with user input

## Code:

```
leap_yaer.py > ...  
leap_yaer.py > ...  
1 #Generate an optimized and well-commented Python program to validate leap year with user input  
2 def is_leap_year(year):  
3     if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):  
4         return True  
5     else:  
6         return False  
7 year = int(input("Enter a year: "))  
8 if is_leap_year(year):  
9     print(year, "is a leap year.")  
10 else:  
11     print(year, "is not a leap year.")
```

## Output:

```
Problems Output Debug Console Terminal Ports  
PS C:\Users\akhil\OneDrive\Documents\AI> & C:\Users\akhil\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akhil/OneDrive/Documents/AI/leap_yaer.py  
Enter a year: 2028  
2028 is a leap year.  
PS C:\Users\akhil\OneDrive\Documents\AI> 2029  
2029  
PS C:\Users\akhil\OneDrive\Documents\AI> 2054  
2054  
PS C:\Users\akhil\OneDrive\Documents\AI> 2060  
2060  
PS C:\Users\akhil\OneDrive\Documents\AI>
```

## **Comparison:**

- **Prompt 1**

- Uses a simple, single-block structure
- Easy to understand and write
- Not reusable
- Uses basic leap year condition
- No comments included

- **Prompt 2**

- Uses a modular approach with a function
- Very clear and well-structured code
- Reusable for multiple checks
- Better optimization through function reuse
- Partially commented for clarity

## **Justification:**

- Cursor AI adapts code style based on prompt specificity.
- Detailed prompts result in cleaner, reusable, and documented code.

## **Task 4: Student Logic + AI Refactoring (Odd/Even Sum)**

### **Original code with output:**

```
1  t = (1, 2, 3, 4, 5)
2  even_sum = 0
3  odd_sum = 0
4
5  for i in t:
6      if i % 2 == 0:
7          even_sum += i
8      else:
9          odd_sum += i
10
11 print("Even Sum:", even_sum)
12 print("Odd Sum:", odd_sum)
13
```

Problems   Output   Debug Console   Terminal   Ports   powershell   +   ⌂   ⌚   ⌘   ⌛   ⌕

PS C:\Users\akhil\OneDrive\Documents\AI> & C:\Users\akhil\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akhil/OneDrive/Documents/AI/leap\_yaer.py
Even Sum: 6
Odd Sum: 9
PS C:\Users\akhil\OneDrive\Documents\AI>

### **Refactored code with output:**

```
1  #Refactor this Python code to improve readability and efficiency.
2  # Improved readability and efficiency by using list comprehensions and built-in sum
3  t = (1, 2, 3, 4, 5)
4
5  even_sum = sum(i for i in t if i % 2 == 0)
6  odd_sum = sum(i for i in t if i % 2 != 0)
7
8  print("Even Sum:", even_sum)
9  print("Odd Sum:", odd_sum)
10
```

```
Problems Output Debug Console Terminal Ports powershell + ·
PS C:\Users\akhil\OneDrive\Documents\AI> & C:\Users\akhil\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akhil/OneDrive/Documents/AI/leap_yaer.py
Even Sum: 6
Odd Sum: 9
PS C:\Users\akhil\OneDrive\Documents\AI>
```

## Justification:

- Refactored code is more concise and Pythonic.
- AI helped optimize logic without changing functionality.