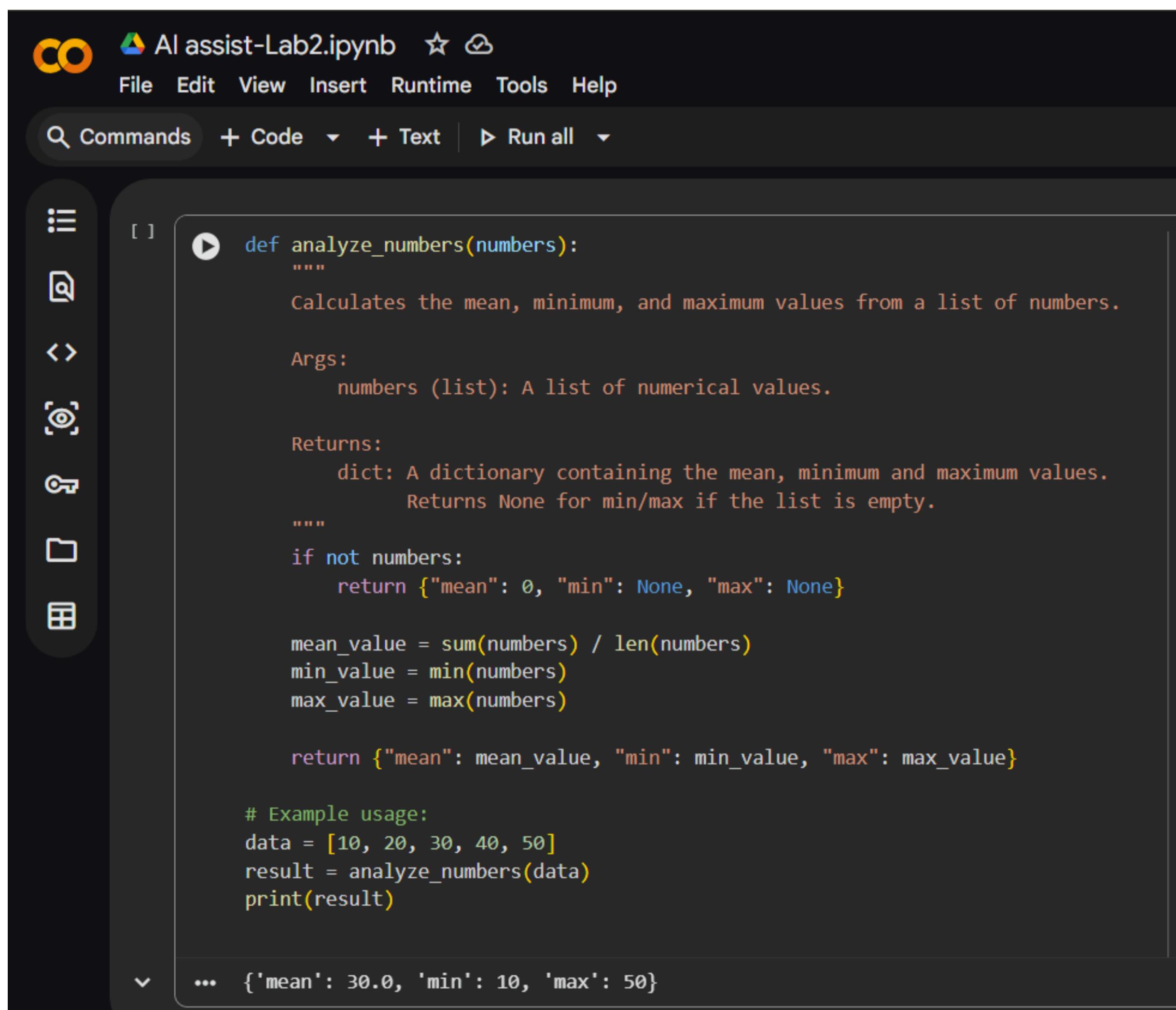


## ASSIGNMENT-2

### 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:**



The screenshot shows a Jupyter Notebook interface with a dark theme. On the left is a sidebar with various icons. The main area contains a cell with the following Python code:

```
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": 0, "min": None, "max": None}

    mean_value = sum(numbers) / len(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

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

# Example usage:
data = [10, 20, 30, 40, 50]
result = analyze_numbers(data)
print(result)
```

The output of the cell is shown at the bottom:

```
... {'mean': 30.0, 'min': 10, 'max': 50}
```

### 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 interface with a dark theme. The top bar includes the CO logo, the file name "AI assist-Lab2.ipynb", and standard menu options: File, Edit, View, Insert, Runtime, Tools, Help. Below the menu is a toolbar with icons for Commands, Code, Text, and Run all. The main area displays the following Python code:

```
[ ] def is_armstrong(number):
    """
    Checks if a number is an Armstrong number.

    An Armstrong number (or 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): The integer to check.

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically positive integers

    num_str = str(number)
    n = len(num_str)
    sum_of_powers = 0

    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** n

    return sum_of_powers == number

# Example Usage:
test_numbers = [9, 153, 370, 371, 407, 1634, 123, 100]

print("Checking Armstrong numbers:")
for num in test_numbers:
    is_armstrong_number = is_armstrong(num)
    print(f"Is {num} an Armstrong number? {is_armstrong_number}")

# You can also test with a specific number
num = 1634
is_armstrong = is_armstrong(num)
print(f"Is {num} an Armstrong number? {is_armstrong}")
```

**Output:**

```
Checking Armstrong numbers:  
Is 9 an Armstrong number? True  
Is 153 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 100 an Armstrong number? False  
Is 1634 an Armstrong number? True
```

## Github copilot code:

```
ai la-2.pyt > ...  
1 #give python code to check if a number is amstrong 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 the number of digits  
11    return sum_of_powers == num  
12 # Example usage  
13 number = int(input("Enter a number: "))  
14 if is_armstrong_number(number):  
15     print(f"{number} is an Armstrong number.")  
16 else:  
17     print(f"{number} is not an Armstrong number.")
```

## Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
● PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab> & "C:/Program Files/Python311/python.exe" "c:/Users/dell/OneDrive/Desktop/AI Assistent-lab.pyt"  
Enter a number: 153  
153 is an Armstrong number.  
● PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab> & "C:/Program Files/Python311/python.exe" "c:/Users/dell/OneDrive/Desktop/AI Assistent-lab.pyt"  
Enter a number: 152  
152 is not an Armstrong number.  
○ PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab>
```

## **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

**Prompt 1:** write a python code to check given year is leap year or not

# Code:

```
ai lab-2(leap_year).pyt > ...
1 #write a python program to check whether a given year is a 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(f"{year} is a leap year.")
5 else:
6     print(f"{year} is not a leap year.")
```

## **Output:**

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

**Code:**

```
.leap_year(AI).py > ...
1  #generate a 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  # Get user input
8  year = int(input("Enter a year: "))
9  if is_leap_year(year):
10     print(f"{year} is a leap year.")
11 else:
12     print(f"{year} is not a leap year.)
```

**Output:**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + v ⌂ ⌂ ⌂ ⌂ X
PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab> & "C:/Program Files/Python311/python.exe" "c:/Users/dell/OneDrive/Desktop/AI Assistent-lab/leap_year(AI).py"
Enter a year: 2028
2028 is a leap year.
PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab> & "C:/Program Files/Python311/python.exe" "c:/Users/dell/OneDrive/Desktop/AI Assistent-lab/leap_year(AI).py"
Enter a year: 2021
2021 is not a leap year.
```

**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:

```
ori leap_year.pyt > ...
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("Sum of even numbers:", even_sum)
12  print("Sum of odd numbers:", odd_sum)
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    Python + trash

```
PS C:\Users\dell\OneDrive\Desktop\AI Assistant-lab> & "C:/Program Files/Python311/python.exe" "c:/Users/dell/OneDrive/Desktop/AI Assistant-lab/orig_leap_year.pyt"
Sum of even numbers: 6
Sum of odd numbers: 9
PS C:\Users\dell\OneDrive\Desktop\AI Assistant-lab>
```

### Refactored code with output:

```
refactor-AI.pyt > ...
1  #Refactor this python code to improve its readability and efficiency.
2  #Improve readability and efficiency by using list comprehension and built-in sum
3  t = (1,2,3,4,5)
4  even_sum = sum(i for i in t if i % 2 == 0)
5  odd_sum = sum(i for i in t if i % 2 != 0)
6
7  print("Sum of even numbers:", even_sum)
8  print("Sum of odd numbers:", odd_sum)
```

The screenshot shows a terminal window in a dark-themed IDE. At the top, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. On the right side, there's a Python icon followed by '+ v [ ]'. The terminal output is as follows:

```
PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab> & "C:/Program Files/Python311/python.exe" "c:/Users/dell/OneDrive/Desktop/AI Assistent-lab/refactor-AI.py"
Sum of even numbers: 6
Sum of odd numbers: 9
PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab>
```

## Justification:

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



