

AI ASSISTED CODING

ASSIGNMENT-2

2303A51666-batch-23

Sony Dodla

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

Task 1: Statistical Summary for Survey Data

❖ Scenario:

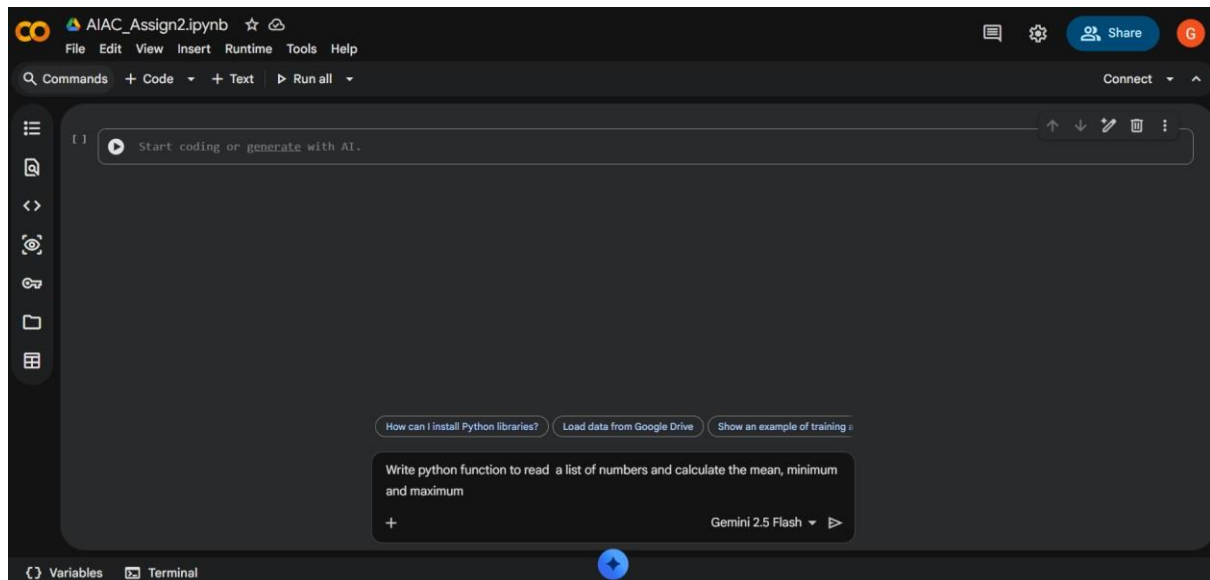
You are a data analyst intern working with survey responses stored as numerical lists.

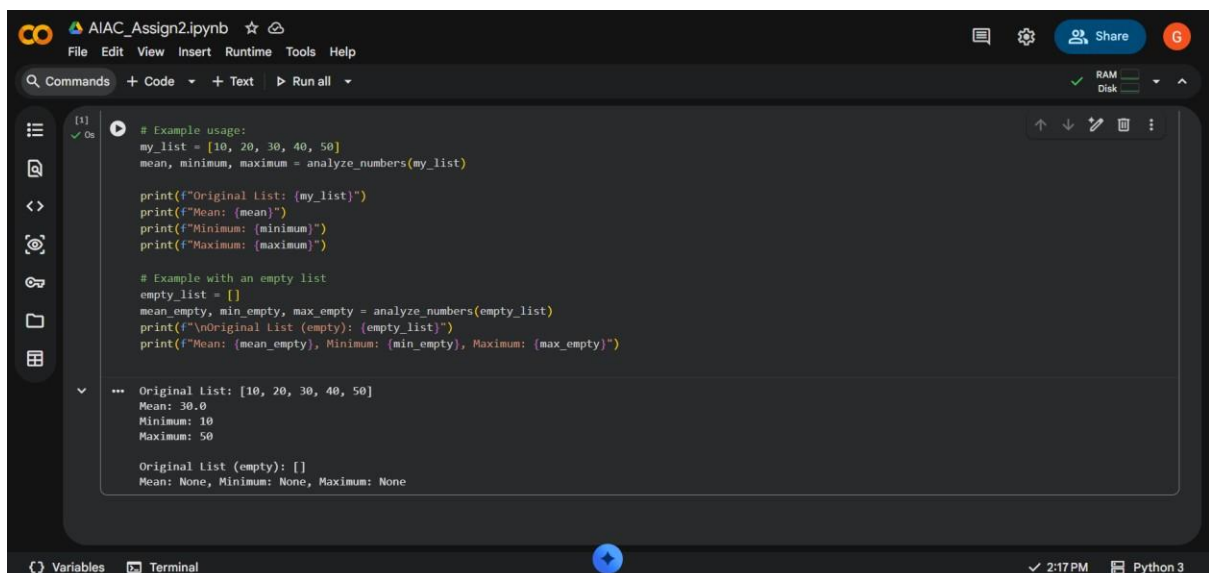
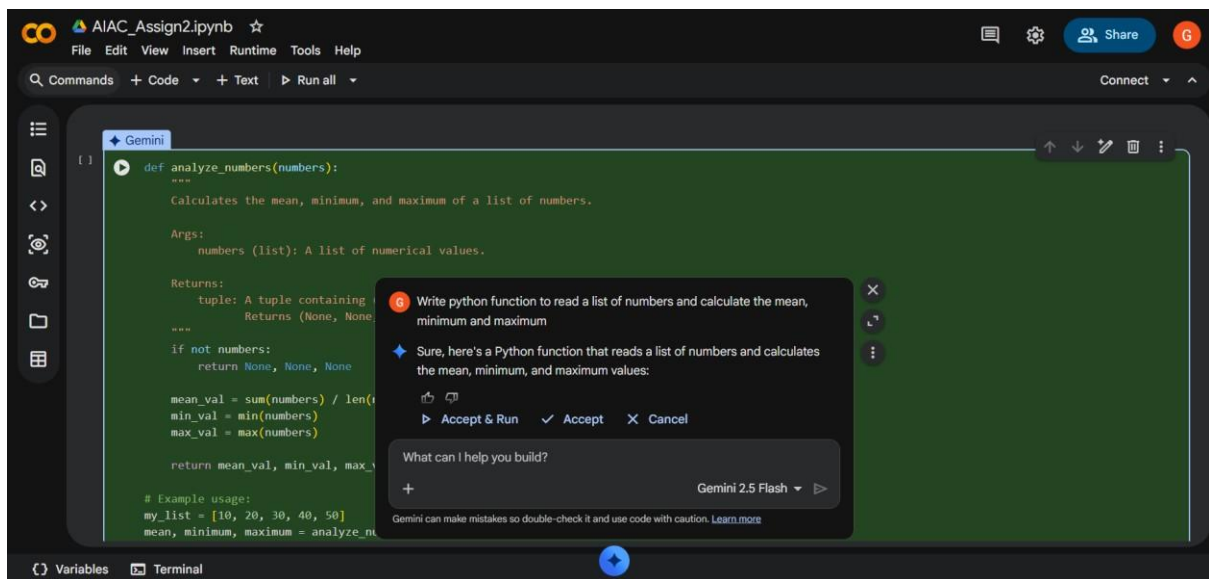
❖ Task:

Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

❖ Expected Output:

- Correct Python function
- Output shown in Colab
- Screenshot of Gemini prompt and result





Task 2: Armstrong Number – AI Comparison

❖ Scenario:

You are evaluating AI tools for numeric validation logic.

❖ Task:

Generate an Armstrong number checker using Gemini and GitHub

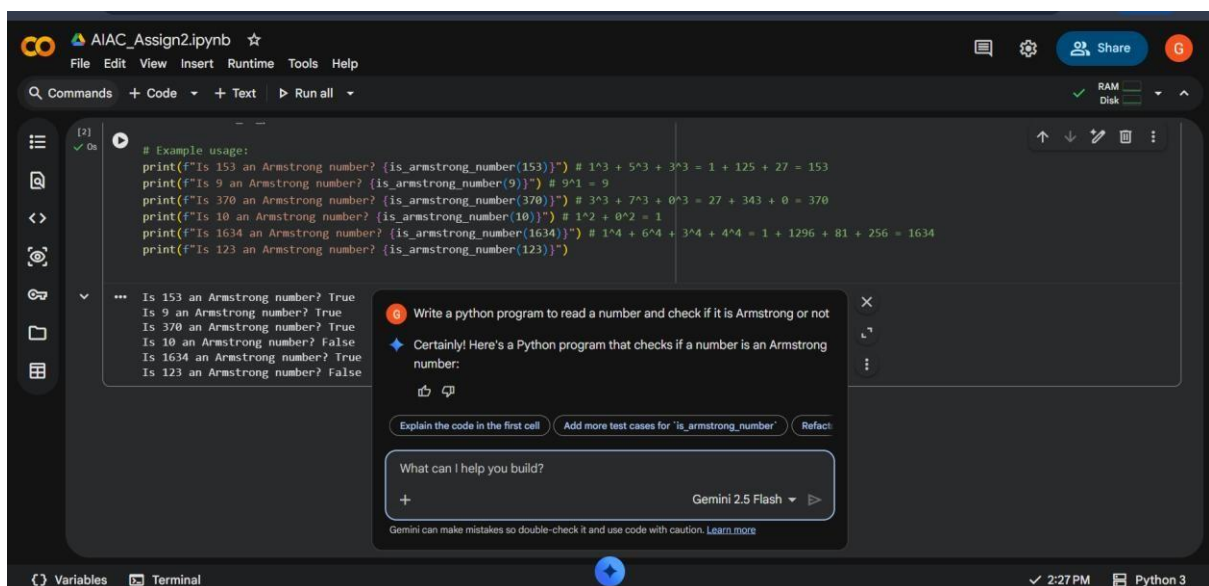
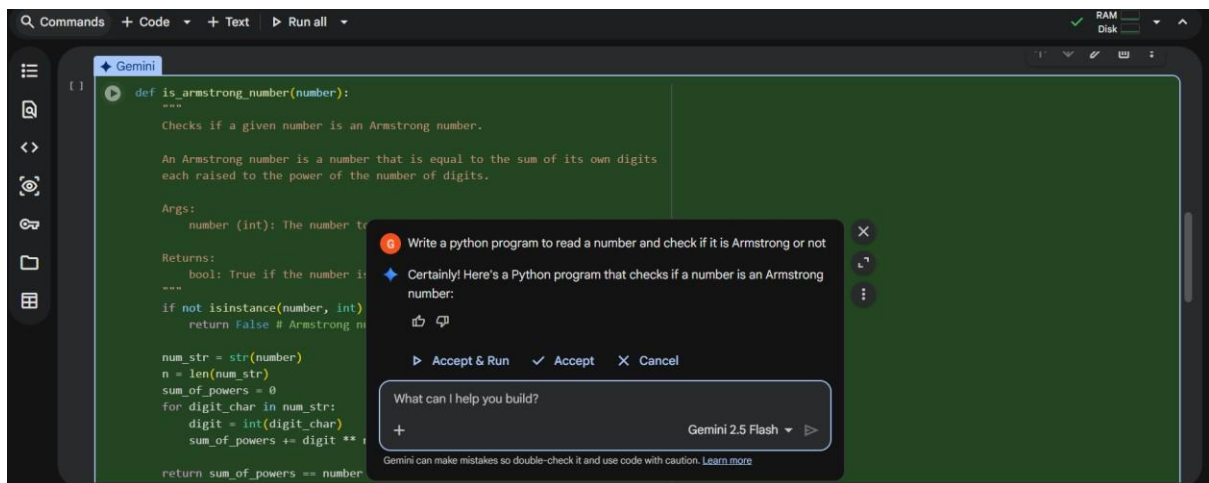
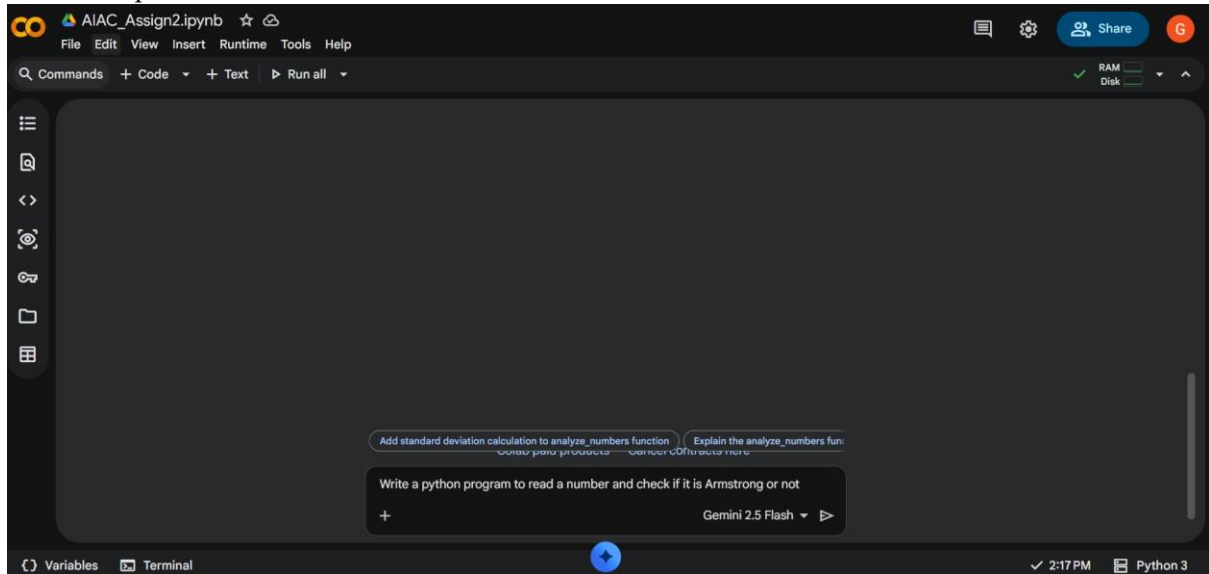
Copilot.

Compare their outputs, logic style, and clarity.

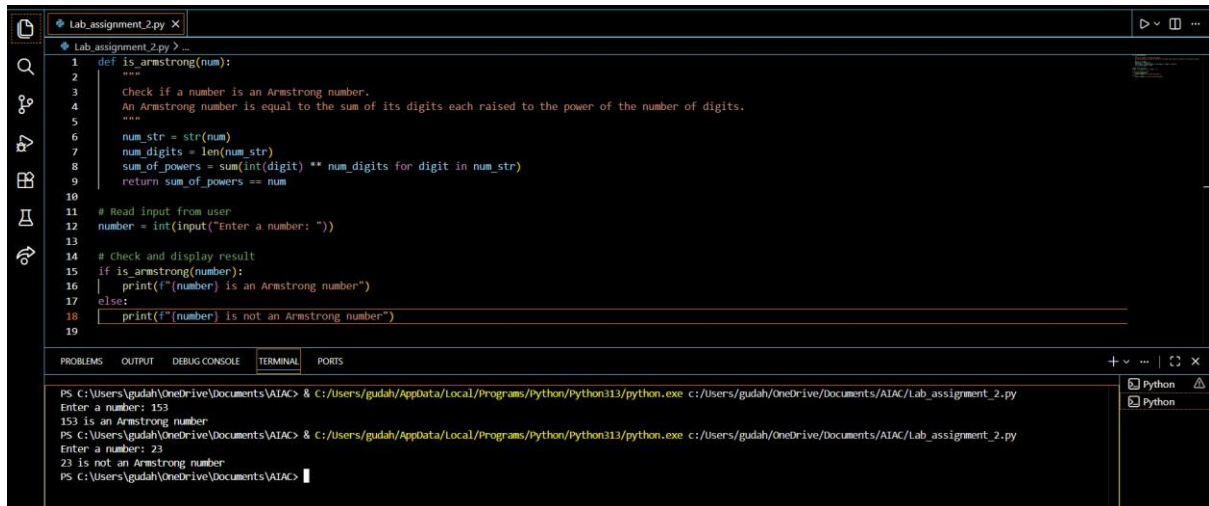
❖ Expected Output:

- Side-by-side comparison table
- Screenshots of prompts and generated code

Gemini Copilot



Github Copilot



Comparison: Armstrong Number Programs

| Aspect | GitHub Copilot Code | Google Colab Gemini Copilot Code |
|---------------------------|---|--|
| Purpose | Checks if a number is an Armstrong number | Checks if a number is an Armstrong number |
| Input Handling | Takes input from the user using input() | Uses predefined test values (no user input) |
| Output Style | Prints a clear message (is / is not an Armstrong number) | Prints True or False with explanation comments |
| Return Value | Returns True or False internally | Returns True or False |
| Logic Style | Uses Pythonic one-line sum with generator expression | Uses step-by-step loop (beginnerfriendly) |
| Code Length | Shorter and more compact | Slightly longer and more detailed |
| Readability | Clean, but may feel compact for beginners | Very clear and easy to understand |
| Validation Checks | No validation for negative or non-integer input | Checks for non-integer and negative numbers |
| Documentation | Short docstring | Detailed docstring with Args & Returns |
| Examples / Testing | Depends on user input | Multiple example test cases included |
| Best For | Quick use, competitive coding, clean scripts | Learning, teaching, and clarity |
| Error Safety | May crash if input is invalid | Safer due to input validation |

Task 3: Leap Year Validation Using Cursor AI

❖ Scenario:

You are validating a calendar module for a backend system.

❖ Task:

Use Cursor AI to generate a Python program that checks whether a given year is a leap year.

Use at least two different prompts and observe changes in code.

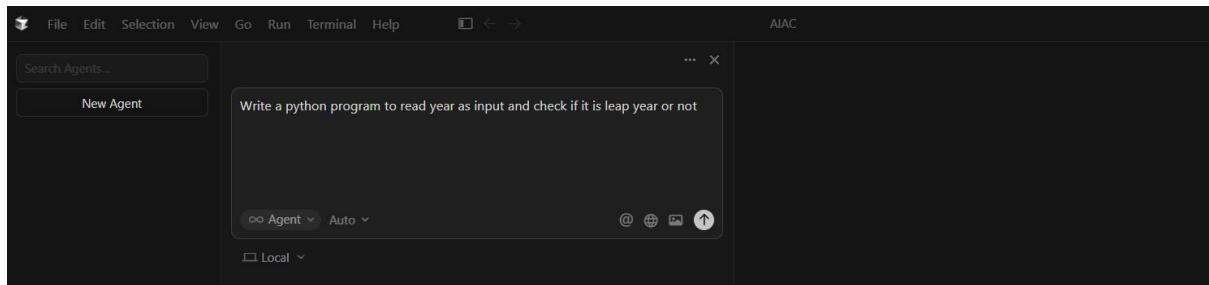
❖ Expected Output:

➤ Two versions of code

➤ Sample inputs/outputs

➤ Brief comparison

Version-1



```
def is_leap_year(year):  
    """  
    Check if a given year is a leap year.  
  
    A year is a leap year if:  
    - It is divisible by 400, OR  
    - It is divisible by 4 but NOT divisible by 100  
  
    Args:  
        year (int): The year to check  
  
    Returns:  
        bool: True if the year is a leap year, False otherwise  
    """  
    if year % 400 == 0:  
        return True  
    elif year % 100 == 0:  
        return False  
    elif year % 4 == 0:  
        return True  
    else:  
        return False
```

Write a python code to read year as input and check if it is leap year or not

```
        return False  
  
# Main program  
if __name__ == "__main__":  
    try:  
        # Read year as input  
        year = int(input("Enter a year: "))  
  
        # Check if it's a Leap year  
        if is_leap_year(year):  
            print(f"{year} is a leap year.")  
        else:  
            print(f"{year} is not a leap year.")  
  
    except ValueError:  
        print("Error: Please enter a valid integer for the year.")
```

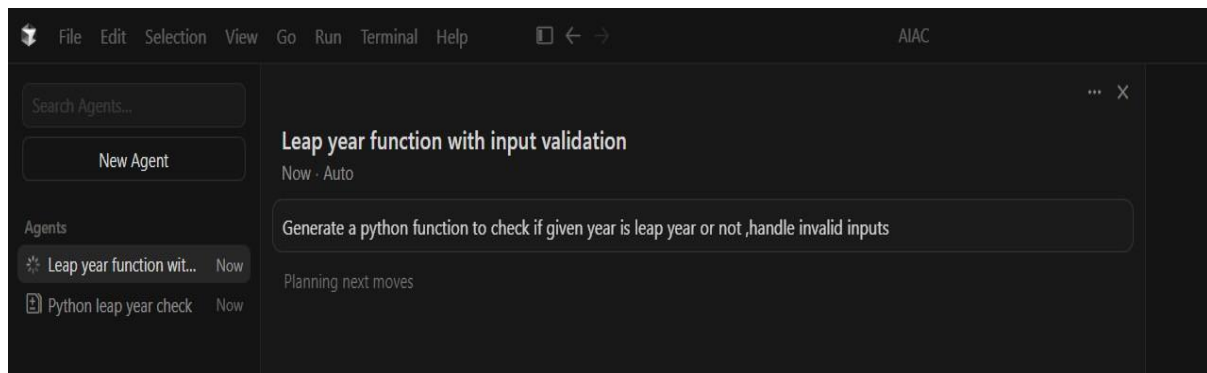
No linter errors found in leap_year.py

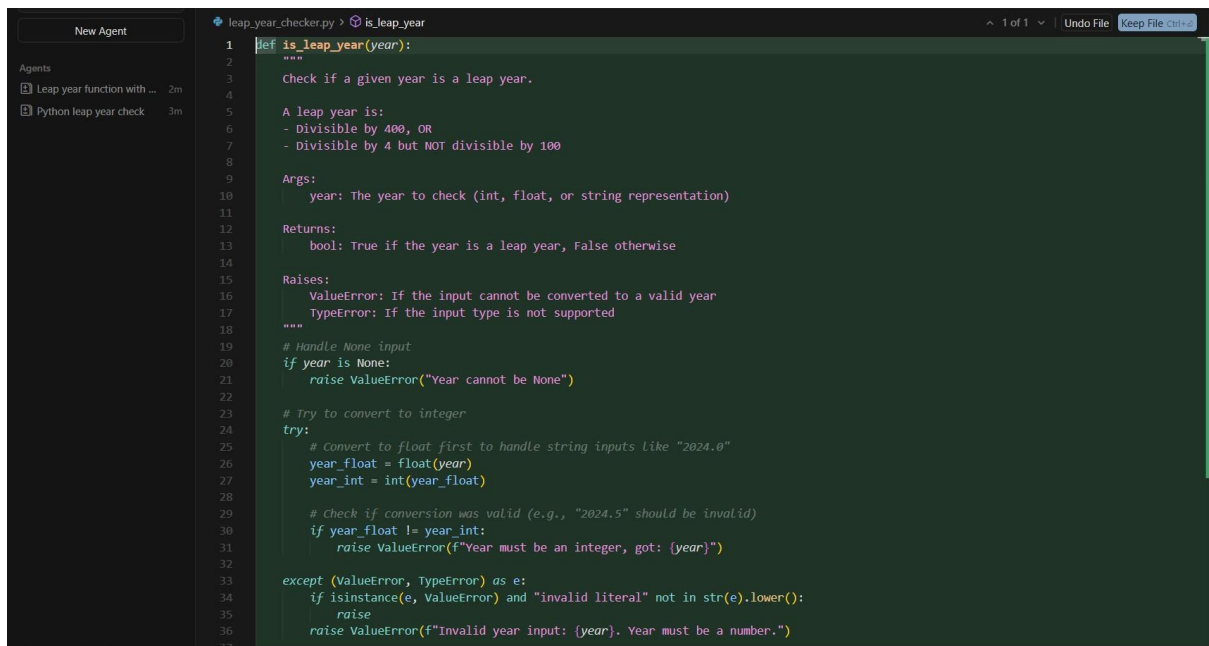
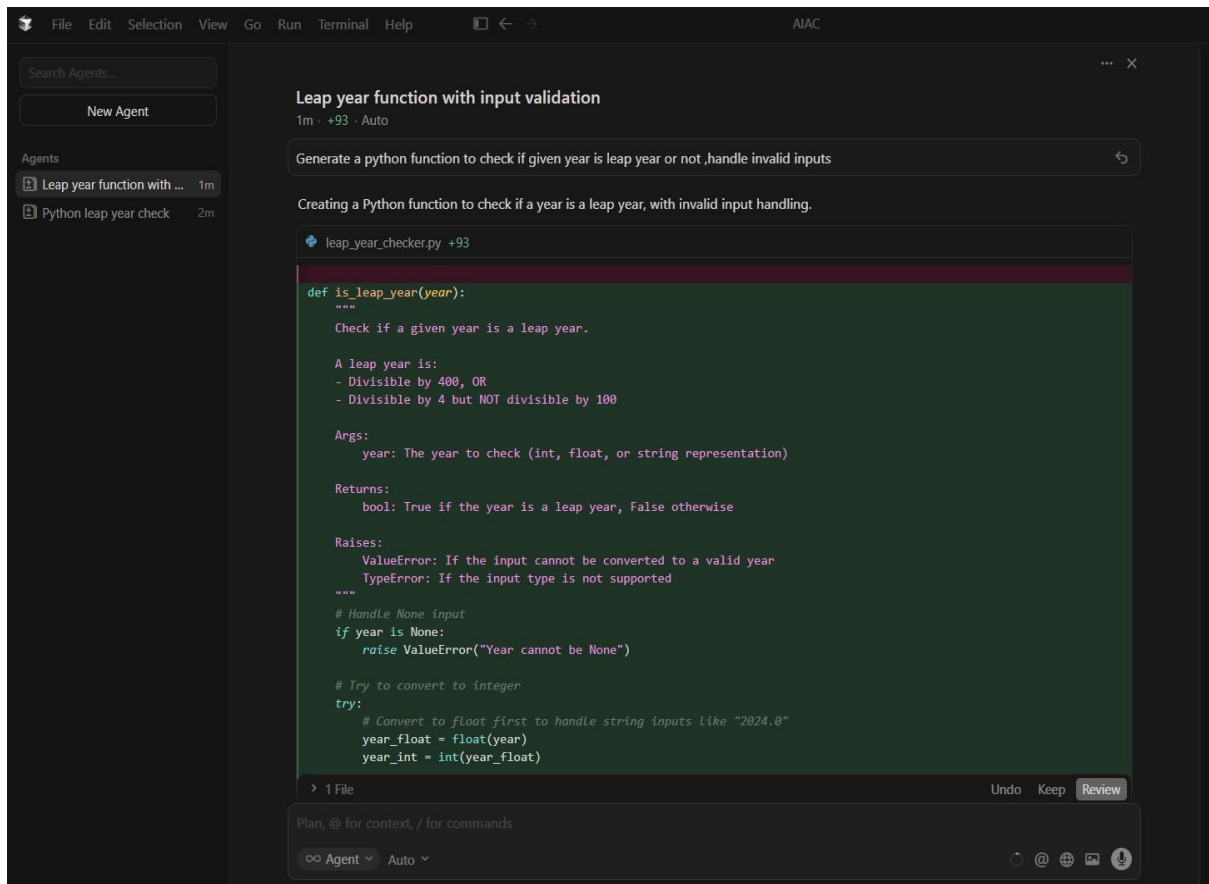
Created leap_year.py that:

1. Reads a year from user input
2. Checks if it's a leap year using:
 - Divisible by 400 → leap year
 - Divisible by 100 (but not 400) → not a leap year
 - Divisible by 4 (but not 100) → leap year
 - Otherwise → not a leap year

```
leap_year.py > ...
1  def is_leap_year(year):
2      """
3      Check if a given year is a leap year.
4
5      A year is a leap year if:
6      - It is divisible by 400, OR
7      - It is divisible by 4 but NOT divisible by 100
8
9      Args:
10     year (int): The year to check
11
12     Returns:
13     bool: True if the year is a leap year, False otherwise
14     """
15     if year % 400 == 0:
16         return True
17     elif year % 100 == 0:
18         return False
19     elif year % 4 == 0:
20         return True
21     else:
22         return False
23
Problems Output Debug Console Terminal Ports
Python + - [ ] [X] ... ^ X
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/leap_year.py
Enter a year: 2004
2004 is a leap year.
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/leap_year.py
Enter a year: 2007
2007 is not a leap year.
PS C:\Users\gudah\OneDrive\Documents\AIAC> [ ]
```

Version-2






```
New Agent
Agents
Leap year function with ... 2m
Python leap year check 3m

1 def is_leap_year(year):
2     # Check if year is positive (leap year calculation doesn't make sense for negative years)
3     if year < 0:
4         raise ValueError(f"Year must be a positive number, got: {year}")
5
6     # Leap year logic:
7     # 1. If divisible by 400, it's a Leap year
8     # 2. Else if divisible by 100, it's NOT a Leap year
9     # 3. Else if divisible by 4, it's a Leap year
10    # 4. Otherwise, it's not a Leap year
11
12    if year % 400 == 0:
13        return True
14    elif year % 100 == 0:
15        return False
16    elif year % 4 == 0:
17        return True
18    else:
19        return False
20
21 # Example usage and test cases
22 if __name__ == "__main__":
23     # Test cases for valid leap years
24     test_cases = [
25         (2000, True), # Divisible by 400
26         (2004, True), # Divisible by 4, not by 100
27         (2020, True), # Divisible by 4, not by 100
28         (2024, True), # Divisible by 4, not by 100
29         (1900, False), # Divisible by 100 but not by 400
30         (2001, False), # Not divisible by 4
31         (2023, False), # Not divisible by 4
32     ]
33
34     # Test cases for invalid inputs
35     invalid_inputs = [
36         None,
37         "abc",
38         "2024.5",
39         -2024,
40         [],
41         {}
42     ]
43
44     # Test valid leap years
45     print("Testing valid leap years:")
46     for year, expected in test_cases:
47         result = is_leap_year(year)
48         status = "✓" if result == expected else "X"
49         print(f"{status} {year}: {result} (expected {expected})")
50
51     # Test invalid inputs
52     print("\nTesting invalid inputs:")
53     for invalid_input in invalid_inputs:
54         try:
55             result = is_leap_year(invalid_input)
56             print(f"X {invalid_input}: Should have raised an error, but returned {result}")
57         except (ValueError, TypeError) as e:
58             print(f"✓ {invalid_input}: Correctly raised {type(e).__name__}: {e}")
```

```
print("Testing valid leap years:")
for year, expected in test_cases:
    result = is_leap_year(year)
    status = "✓" if result == expected else "X"
    print(f"{status} {year}: {result} (expected {expected})")

print("\nTesting invalid inputs:")
invalid_inputs = [
    None,
    "abc",
    "2024.5",
    -2024,
    [],
    {}
]

for invalid_input in invalid_inputs:
    try:
        result = is_leap_year(invalid_input)
        print(f"X {invalid_input}: Should have raised an error, but returned {result}")
    except (ValueError, TypeError) as e:
        print(f"✓ {invalid_input}: Correctly raised {type(e).__name__}: {e}")
```

```
leap_year_checker.py > ...
1 def is_leap_year(year):
19     # Handle None input
20     if year is None:
21         raise ValueError("Year cannot be None")
22
23     # Try to convert to integer
24     try:
25         # Convert to float first to handle string inputs like "2024.0"
26         year_float = float(year)
27         year_int = int(year_float)
28
29         # Check if conversion was valid (e.g., "2024.5" should be invalid)
30         if year_float != year_int:
31             raise ValueError(f"Year must be an integer, got: {year}")
32
33     except ValueError as e:
34         raise ValueError(f"Year must be a positive number, got: {year}")
35
36     # Leap year logic:
37     # 1. If divisible by 400, it's a Leap year
38     # 2. Else if divisible by 100, it's NOT a Leap year
39     # 3. Else if divisible by 4, it's a Leap year
40     # 4. Otherwise, it's not a Leap year
41
42     if year % 400 == 0:
43         return True
44     elif year % 100 == 0:
45         return False
46     elif year % 4 == 0:
47         return True
48     else:
49         return False
50
51 # Example usage and test cases
52 if __name__ == "__main__":
53     # Test cases for valid leap years
54     test_cases = [
55         (2000, True), # Divisible by 400
56         (2004, True), # Divisible by 4, not by 100
57         (2020, True), # Divisible by 4, not by 100
58         (2024, True), # Divisible by 4, not by 100
59         (1900, False), # Divisible by 100 but not by 400
60         (2001, False), # Not divisible by 4
61         (2023, False), # Not divisible by 4
62     ]
63
64     # Test cases for invalid inputs
65     invalid_inputs = [
66         None,
67         "abc",
68         "2024.5",
69         -2024,
70         [],
71         {}
72     ]
73
74     # Test valid leap years
75     print("Testing valid leap years:")
76     for year, expected in test_cases:
77         result = is_leap_year(year)
78         status = "✓" if result == expected else "X"
79         print(f"{status} {year}: {result} (expected {expected})")
80
81     # Test invalid inputs
82     print("\nTesting invalid inputs:")
83     for invalid_input in invalid_inputs:
84         try:
85             result = is_leap_year(invalid_input)
86             print(f"X {invalid_input}: Should have raised an error, but returned {result}")
87         except (ValueError, TypeError) as e:
88             print(f"✓ {invalid_input}: Correctly raised {type(e).__name__}: {e}")
```

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

❖ Scenario:

Company policy requires developers to write logic before using AI.

❖ Task:

Write a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it using any AI tool.

❖ Expected Output:

➤ Original code

➤ Refactored code

➤ Explanation of improvements

Original code



```
22 #Sum of even and odd numbers in tuple
23 t=tuple(map(int,input().split()))
24 es=0
25 os=0
26 for i in t:
27     if i%2==0:
28         es+=i
29     else:
30         os+=i
31 print("Sum of even numbers is ",es)
32 print("sum of odd numbers is ",os)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

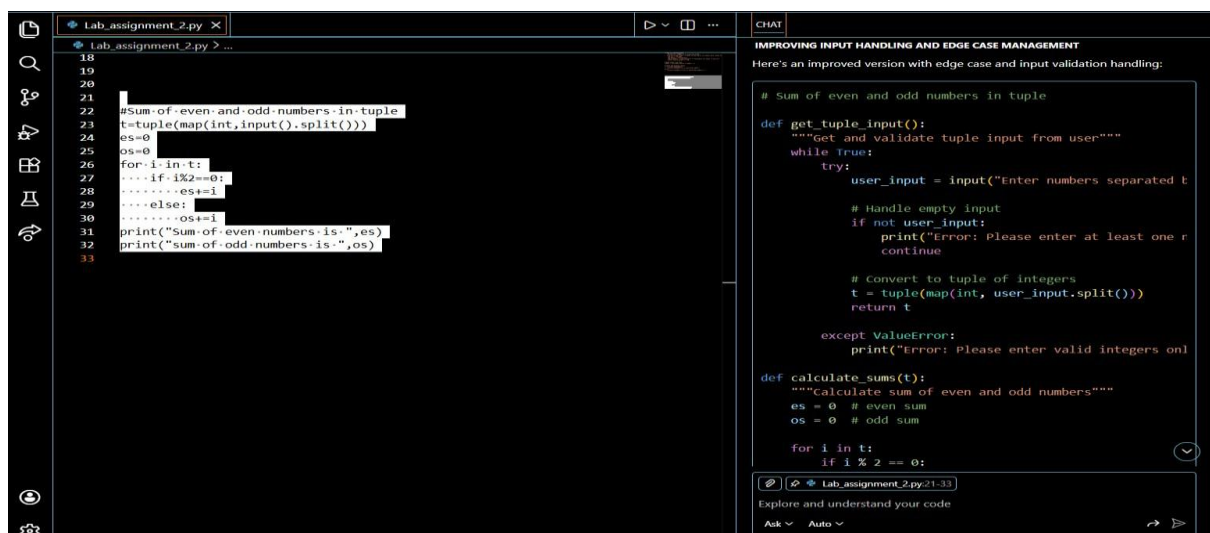
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/documents/AIAC/Lab_assignment_2.py

3 0 1 0 3 2 4 1

Sum of even numbers is 12

sum of odd numbers is 8

Refactored Code



```
18
19
20
21 #Sum of even and odd numbers in tuple
22 t=tuple(map(int,input().split()))
23 es=0
24 os=0
25
26 for i in t:
27     if i%2==0:
28         es+=i
29     else:
30         os+=i
31 print("Sum of even numbers is ",es)
32 print("sum of odd numbers is ",os)
```

CHAT

IMPROVING INPUT HANDLING AND EDGE CASE MANAGEMENT

Here's an improved version with edge case and input validation handling:

```
# Sum of even and odd numbers in tuple

def get_tuple_input():
    """Get and validate tuple input from user"""
    while True:
        try:
            user_input = input("Enter numbers separated by space: ")
            # Handle empty input
            if not user_input:
                print("Error: Please enter at least one number")
                continue
            # Convert to tuple of integers
            t = tuple(map(int, user_input.split()))
            return t
        except ValueError:
            print("Error: Please enter valid integers only")

def calculate_sums(t):
    """Calculate sum of even and odd numbers"""
    es = 0 # even sum
    os = 0 # odd sum

    for i in t:
        if i % 2 == 0:
            es += i
        else:
            os += i

    return es, os
```

Lab_assignment_2.py:21-33

Explore and understand your code

Ask Auto

```
Lab_assignment_2.py X
Lab_assignment_2.py > get_tuple_input
37 # Sum of even and odd numbers in tuple
38 def get_tuple_input():
39     """Get and validate tuple input from user"""
40     while True:
41         try:
42             user_input = input("Enter numbers separated by spaces: ").strip()
43             # Handle empty input
44             if not user_input:
45                 print("Error: Please enter at least one number.")
46                 continue
47             # Convert to tuple of integers
48             t = tuple(map(int, user_input.split()))
49             return t
50         except ValueError:
51             print("Error: Please enter valid integers only.")
52 def calculate_sums(t):
53     """Calculate sum of even and odd numbers"""
54     es = 0 # even sum
55     os = 0 # odd sum
56     for i in t:
57         if i % 2 == 0:
58             es += i
59         else:
60             os += i
61     return es, os
62 # Main program
63 if __name__ == "__main__":
64     t = get_tuple_input()
65     if len(t) == 0:
66         print("Error: Tuple is empty.")
67     else:
68         es, os = calculate_sums(t)
69         print(f"Sum of even numbers: {es}")
70         print(f"Sum of odd numbers: {os}")
71         print(f"Total numbers: {len(t)}")
```

```
Lab_assignment_2.py X
Lab_assignment_2.py > ...
37 # Sum of even and odd numbers in tuple
38 def get_tuple_input():
39     """Get and validate tuple input from user"""
40     while True:
41         try:
42             user_input = input("Enter numbers separated by spaces: ").strip()
43             # Handle empty input
44             if not user_input:
45                 print("Error: Please enter at least one number.")
46                 continue
47             # Convert to tuple of integers
48             t = tuple(map(int, user_input.split()))
49             return t
50         except ValueError:
51             print("Error: Please enter valid integers only.")
52 def calculate_sums(t):
53     """Calculate sum of even and odd numbers"""
54     es = 0 # even sum
55     os = 0 # odd sum
56     for i in t:
57         if i % 2 == 0:
58             es += i
59         else:
60             os += i
61     return es, os
62 # Main program
63 if __name__ == "__main__":
64     t = get_tuple_input()
65     if len(t) == 0:
66         print("Error: Tuple is empty.")
67     else:
68         es, os = calculate_sums(t)
69         print(f"Sum of even numbers: {es}")
70         print(f"Sum of odd numbers: {os}")
71         print(f"Total numbers: {len(t)}")
```

| PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS |
|----------|--------|---------------|---|-------|
| | | | Enter numbers separated by spaces: 7 5 -9 0 7 1 3 Sum of even numbers: 0 Sum of odd numbers: 14 Sum of odd numbers: 14 Total numbers: 7 | |

Key improvements:

- **Input validation:** Checks for empty input and non-integer values
- **Error handling:** Try-except block catches invalid inputs
- **Retry logic:** Prompts user again if input is invalid
- **Modular functions:** Separated logic into reusable functions
- **Better formatting:** Uses f-strings for cleaner output
- **Edge cases handled:** Empty tuples, zero values, negative numbers
- **User-friendly messages:** Clear error and result messages