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, ang maximum values

Code:

AI assist-Lab2.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[ ]  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)

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

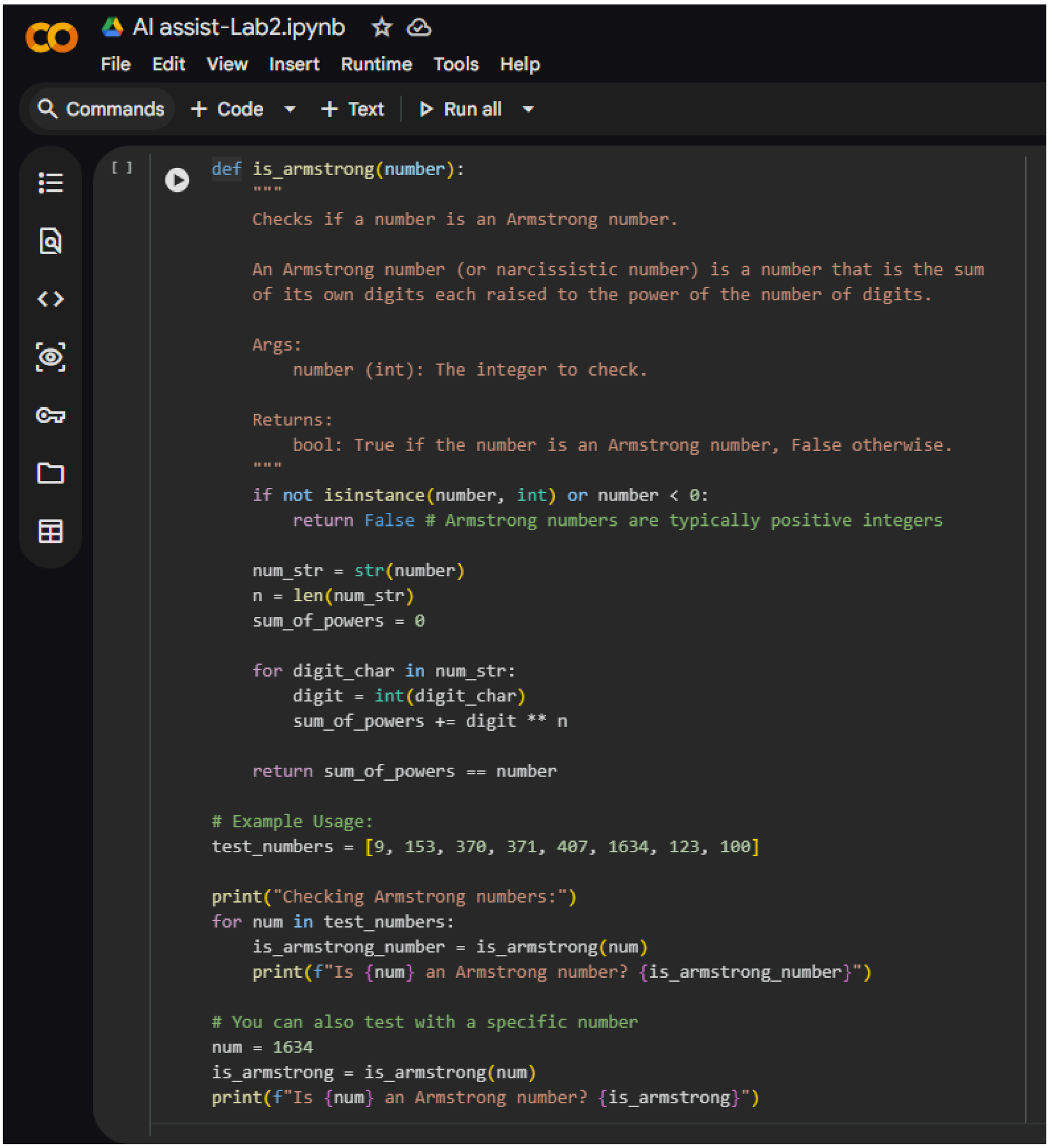
Justification:

- 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:



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 > ...

```python
#give python code to check if a number is amstrong number or not
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of each digit raised to the power of number of digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)

    # An Armstrong number is equal to the sum of its own digits each raised to the power of the number of digits
    return sum_of_powers == num
# Example usage
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    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/Deskt
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/Desk
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:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

2021 is not 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/ai lab-2(leap_year).pyt"
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/ai lab-2(leap_year).pyt"
Enter a year: 2020
2020 is a leap year.
PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab>
```

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

Code:

```python
#generate a optimized and well-commented python program to validateleap year with user input
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
# Get user input
year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Output:

```
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 > ...

```python
t = (1,2,3,4,5)
even_sum = 0
odd_sum = 0

for i in t:
    if i % 2 == 0:
        even_sum += i
    else:
        odd_sum += i

print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

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

Refactored code with output:

refactor-AI.pyt > ...

```python
#Refactor this python code to improve its readability and efficiency.
#Improve readability and efficiency by using list comprehension and built-in sum
t = (1,2,3,4,5)
even_sum = sum(i for i in t if i % 2 == 0)
odd_sum = sum(i for i in t if i % 2 != 0)

print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

Justification:

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