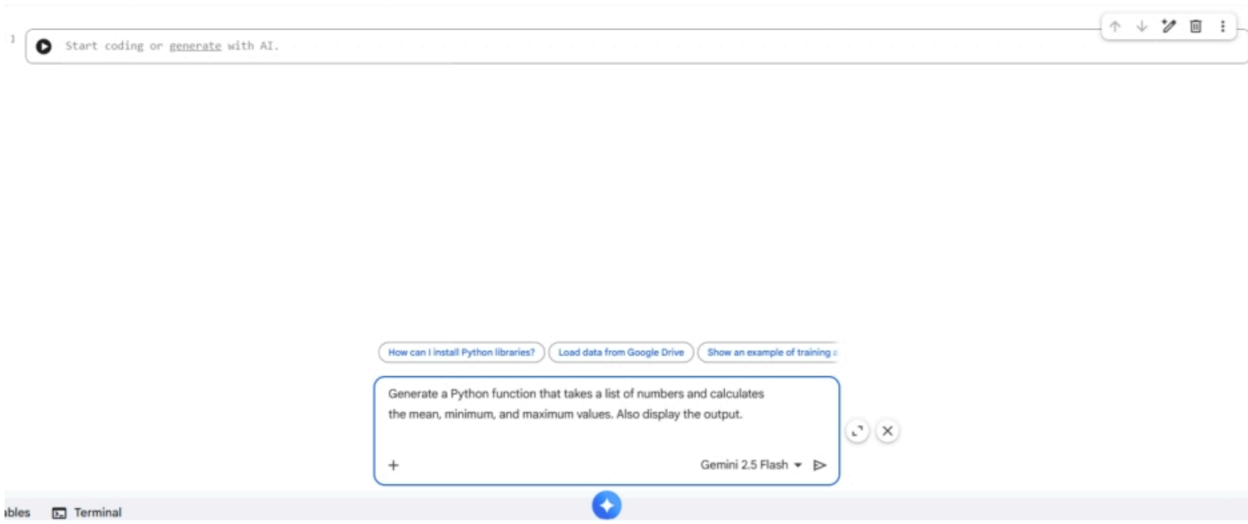
Assignment-2.1

HT No: 2303A51804

Batch No:28

Task1: Statistical Summary for Survey Data

Prompt: Generate a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values. Also display the output



Code:

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

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

    Returns:
        tuple: A tuple containing (mean, minimum, maximum).
               Returns (None, None, None) if the list is empty.
    """
    if not numbers:
        return None, None, None

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

    return mean_value, min_value, max_value
```

```python
# Example usage:
my_list = [10, 20, 5, 30, 15, 25]
mean, minimum, maximum = calculate_statistics(my_list)

print(f"Original List: {my_list}")
print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")

# Example with an empty list
empty_list = []
mean_empty, min_empty, max_empty = calculate_statistics(empty_list)
print(f"\nOriginal List (empty): {empty_list}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum: {max_empty}")
```

Output:

```
Original List: [10, 20, 5, 30, 15, 25]
Mean: 17.5
Minimum: 5
Maximum: 30

Original List (empty): []
Mean: None
Minimum: None
Maximum: None
```

Justification:

1.Defines a reusable function to calculate basic statistical values from a list.

2.Uses a docstring to clearly explain the purpose, inputs, and outputs of the function.

3. Checks for an empty list to prevent runtime errors like division by zero.

4.Applies Python built-in functions for efficient and accurate calculations.

5.Demonstrates function usage with a sample list for clarity and testing.

6.Validates edge-case handling by testing the function with an empty list.

Task2: Armstrong Number – AI Comparison

Prompt: Write a Python program to check whether a number is an Armstrong number.

Part A: GEMINI AI(Google Colab)

Code:

```python
def is_armstrong(number):
    """
    Checks if a given number is an Armstrong number.

    An Armstrong number (also known as a narcissistic number,
    pluperfect digital invariant, or plus perfect 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:
        raise ValueError("Input must be a non-negative integer.")

    # Convert number to string to easily get digits and count them
    num_str = str(number)
    n_digits = len(num_str)

    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** n_digits

    return sum_of_powers == number
```

```python
# Example usage:
print(f"Is 153 an Armstrong number? {is_armstrong(153)}") # Expected: True (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153)
print(f"Is 9 an Armstrong number? {is_armstrong(9)}")    # Expected: True (9^1 = 9)
print(f"Is 370 an Armstrong number? {is_armstrong(370)}") # Expected: True (3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370)
print(f"Is 120 an Armstrong number? {is_armstrong(120)}") # Expected: False (1^3 + 2^3 + 0^3 = 1 + 8 + 0 = 9 != 120)
print(f"Is 1634 an Armstrong number? {is_armstrong(1634)}") # Expected: True (1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634)

# Test with a non-Armstrong number
non_armstrong = 123
print(f"Is {non_armstrong} an Armstrong number? {is_armstrong(non_armstrong)}")
```

Output:

```
Is 153 an Armstrong number? True
Is 9 an Armstrong number? True
Is 370 an Armstrong number? True
Is 120 an Armstrong number? False
Is 1634 an Armstrong number? True
Is 123 an Armstrong number? False
```

PartB: GitHub Copilot(VS Code)

Code:

```python
#Write a Python program to check whether a number is an Armstrong number.
# An Armstrong number (also known as a narcissistic number) is a number that is equal to the sum of its own digits
#  each raised to the power of the number of digits.
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    # Get the number of digits
    num_digits = len(str_num)
    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    # Check if the sum of powers is equal to the original number
    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:

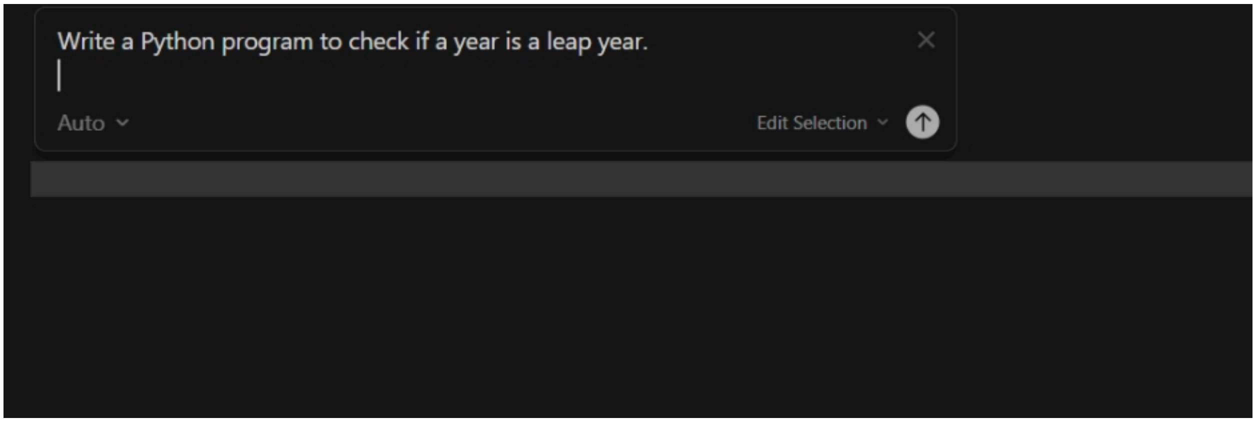
```
tivate.ps1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Sc
ripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/Assignment(2.1).py"
Enter a number: 153
153 is an Armstrong number.
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

Justification: (Comparison between Gemini AI and GitHub Copilot)

Feature	Gemini (Google Colab)	Copilot (VS Code)
Code clarity	High – code is structured with clear flow and validation	Medium – logic is correct but less structured
Use of functions	Yes – uses a well-defined function with validation	Yes – uses a function but with minimal safeguards
Readability	Good – descriptive variable names and spacing	Good – concise but slightly compact
Comments	More – includes detailed docstring and inline comments	Less – basic comments only
Logic approach	Direct – step-by-step logic easy to follow	Optimized – uses concise expressions (generator)
Error handling	Present – checks input type and negative values	Not present – assumes valid input
Suitability for learning	High – beginner-friendly and explanatory	Medium – better for experienced users

Task3: Leap Year Validation Using Cursor AI

Prompt1: Write a Python program to check if a year is a leap year or not.



Code:

```python
year = int(input("Enter a year: "))

if (year % 4 == 0):
    if (year % 100 == 0):
        if (year % 400 == 0):
            print("Leap year")
        else:
            print("Not a leap year")
    else:
        print("Leap year")
else:
    print("Not a leap year")
Ctrl+L to chat, Ctrl+K to generate
```

Output :

```
Enter a year: 2020
Leap year
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/DELL/OneDrive/Desktop/LEARNING C
g/assignment-(2.1 CURSOR).py"
Enter a year: 2021
Not a leap year
PS C:\Users\DELL>
```

Prompt2: Write a clean and optimized Python function to validate leap year with input and output handling.

Code:

```python
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False

year = int(input("Enter a year: "))
if is_leap_year(year):
    print("Leap year")
else:
    print("Not a leap year")
```

Output:

```
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/DELL/OneDrive/
OURSES/AI_Assisant_Coding/assignment-(2.1 CURSOR).py"
Enter a year: 2020
2020 is a leap year.
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/DELL/OneDrive/
OURSES/AI_Assisant_Coding/assignment-(2.1 CURSOR).py"
Enter a year: 2025
2025 is not a leap year.
PS C:\Users\DELL>
```

Justification:

1.The simple code uses nested if–else statements, which makes the leap year logic easy to follow for beginners.

2.The optimized code improves structure by placing the logic inside a function, making it reusable.

3.Both codes correctly implement leap year rules using divisibility by 4, 100, and 400.

4.The optimized version reduces repetition and separates logic from input and output.

5.Overall, the simple code is good for learning, while the optimized code is better for maintainable and real-world programs.

Task4: Student Logic + AI Refactoring (Odd/Even Sum)

Creating Student own code on even Or odd number without using AI

```python
#Adding Student own code on even Or odd number without using AI
#create  a function for sum of even and odd numbers
def sum_even_odd(nums):
    sum_even=0
    sum_odd=0
    for i in nums:
        if i%2==0:
            sum_even+=i
        else:
            sum_odd+=i
    print("sum of Even Numbers: ",sum_even)
    print("sum of Odd Numbers: ",sum_odd)
li=[1,2,3,4,5,6]
sum_even_odd(li)
```

Output:

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_As
ant_Coding/Assignment(2.1).py"
sum of Even Numbers:  12
sum of Odd Numbers:  9
```

Prompt: Refactor this Python code to make it cleaner, more readable, and reusable.

Code:

```python
def sum_even_odd(nums):
    """
    Calculates the sum of even and odd numbers in a list.

    Args:
        nums (list): A list of numbers.

    Returns:
        tuple: A tuple containing (sum_even, sum_odd).
    """
    sum_even = 0
    sum_odd = 0
    for num in nums:
        if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num
    return sum_even, sum_odd

li = [1, 2, 3, 4, 5, 6]
even_sum, odd_sum = sum_even_odd(li)
print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

Output:

```
Sum of Even Numbers: 12
Sum of Odd Numbers: 9
```

Justification:

1.Both programs correctly separate even and odd numbers using the modulus (% 2) condition.

2.The first code is student-written and directly prints results, making it easy to understand for beginners.

3.The second code improves structure by using a docstring and returning values instead of printing inside the function.

4. Returning values in the optimized code increases reusability and makes it suitable for larger programs.

5.Overall, the first code is good for learning basic logic, while the second code follows better programming practices.