NAME : M.Vignesh goud **ENROLL NO : 2403A53012** **BATCH NO: 24BTCAICYB01**	
Assignment Number: 2.1(Present assignment number)/**24**(Total number of assignments)	

Q.No.	**Question**	***Expected Time to complete***
	Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI **Lab Objectives:** • To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. • To understand and use Cursor AI for code generation, explanation, and refactoring. • To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. • To perform code optimization and documentation using AI tools. **Lab Outcomes (LOs):** After completing this lab, students will be able to: • Generate Python code using Google Gemini in Google Colab. • Analyze the effectiveness of code explanations and suggestions by Gemini. • Set up and use Cursor AI for AI-powered coding assistance. • Evaluate and refactor code using Cursor AI features. • Compare AI tool behavior and code quality across different platforms.	

Task Description #1

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

Expected Output #1

- Functional code with correct output and screenshot.

```
def calculate_stats(numbers):
  if not numbers:
    return None, None, None
  mean = sum(numbers) / len(numbers)
  minimum = min(numbers)
  maximum = max(numbers)
  return mean, minimum, maximum
my_list = [10, 20, 30, 40, 50]
mean_value, min_value, max_value = calculate_stats(my_list)

print(f"List: {my_list}")
print(f"Mean: {mean_value}")
print(f"Minimum: {min_value}")
print(f"Maximum: {max_value}")

empty_list = []
mean_empty, min_empty, max_empty = calculate_stats(empty_list)
print(f"\nList: {empty_list}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum: {max_empty}")
```

```
List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50

List: []
Mean: None
Minimum: None
Maximum: None
```

Task Description #2

- Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

Expected Output #2

- Side-by-side comparison table with observations and screenshots.

```
def is_armstrong_number(number):
  num_str = str(number)
  num_digits = len(num_str)
  sum_of_powers = 0
  for digit in num_str:
    sum_of_powers += int(digit) ** num_digits
  return sum_of_powers == number
num1 = 153
num2 = 9474
num3 = 123
print(f"{num1} is an Armstrong number: {is_armstrong_number(num1)}")
print(f"{num2} is an Armstrong number: {is_armstrong_number(num2)}")
print(f"{num3} is an Armstrong number: {is_armstrong_number(num3)}")
```

```
153 is an Armstrong number: True
9474 is an Armstrong number: True
123 is an Armstrong number: False
```

Task Description #3

- Ask Gemini to explain a Python function (e.g., is_prime(n) or is_palindrome(s)) line by line.
- Choose either a prime-checking or palindrome-checking function and document the explanation provided by Gemini.

1

Week1 - Monday

Expected Output #3

- Detailed explanation with the code snippet and Gemini's response.

```
def is_prime(n):
  if n <= 1:
    return False
  for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
      return False
  return True
print(f"7 is prime: {is_prime(7)}")
print(f"10 is prime: {is_prime(10)}")
print(f"1 is prime: {is_prime(1)}")
```

```
7 is prime: True
10 is prime: False
1 is prime: False
```

Task Description #4

- Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output.
- Optionally, compare Cursor AI's generated code with Gemini's output.

Expected Output #4

- Screenshots of Cursor AI setup, prompts used, and generated code with output.
-

```
def sum_first_n(n: int) -> int:
    """
    Return the sum of the first n natural numbers (1 + 2 + ... + n).
    Raises ValueError if n is negative.
    """
    if n < 0:
        raise ValueError("n must be non-negative")
    return n * (n + 1) // 2

# Example
if __name__ == "__main__":
    print(sum_first_n(10))  # 55
```

-

Task Description #5

- Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.
- Refactor the code to improve logic and readability.

Expected Output #5

- Student-written refactored code with explanations and output screenshots.

```python
def sum_odd_even(numbers):
  odd_sum = 0
  even_sum = 0
  for number in numbers:
    if number % 2 == 0:
      even_sum += number
    else:
      odd_sum += number

  return odd_sum, even_sum
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
odd_sum, even_sum = sum_odd_even(my_tuple)
print(f"Tuple: {my_tuple}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
empty_tuple = ()
odd_sum_empty, even_sum_empty = sum_odd_even(empty_tuple)
print(f"\nTuple: {empty_tuple}")
print(f"Sum of odd numbers: {odd_sum_empty}")
print(f"Sum of even numbers: {even_sum_empty}")
```

```
Tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of odd numbers: 25
Sum of even numbers: 30

Tuple: ()
Sum of odd numbers: 0
Sum of even numbers: 0
```

Note:

- Students must submit a single Word document including:
  - Prompts used for AI tools
  - Copilot/Gemini/Cursor outputs
  - Code explanations
  - Screenshots of outputs and environments

Evaluation Criteria: