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

Task 1: Cleaning Sensor Data

```
def clean_sensor_data(data):
    # Handle non-list input
    if not isinstance(data, list):
        return "Error: Input must be a list."

    # Empty list case
    if len(data) == 0:
        return "The list is empty."

    # Check for non-numeric values
    non_numeric = [x for x in data if not isinstance(x, (int, float))]
    if non_numeric:
        return f"Warning: Non-numeric values detected: {non_numeric}. Only
numeric values processed."

    # Keep only non-negative numbers
    valid_data = [x for x in data if x >= 0]

    if len(valid_data) == 0:
        return "All values are negative or invalid. No valid sensor
readings."
    elif len(valid_data) == len(data):
        return "All values are valid (no negatives found)."
    else:
        return {
            "Original List": data,
            "Cleaned List": valid_data
        }

# Test cases
print("=== Task 1: Sensor Data Cleaning ===")
test_cases = [
    [1, -2, 3, -4, 5],
    [-1, -2, -3],
    [],
    [10, 20, 30],
    ['a', -1, 2, 'b'],
    [0, -0.0, 5.5]
```

```
]

for i, case in enumerate(test_cases, 1):
    print(f"\nTest Case {i}: {case}")
    result = clean_sensor_data(case)
    print(result)
```

Output:

```
=== Task 1: Sensor Data Cleaning ===

Test Case 1: [1, -2, 3, -4, 5]
{'Original List': [1, -2, 3, -4, 5], 'Cleaned List': [1, 3, 5]}

Test Case 2: [-1, -2, -3]
All values are negative or invalid. No valid sensor readings.

Test Case 3: []
The list is empty.

Test Case 4: [10, 20, 30]
All values are valid (no negatives found).

Test Case 5: ['a', -1, 2, 'b']
Warning: Non-numeric values detected: ['a', 'b']. Only numeric values
processed.

Test Case 6: [0, -0.0, 5.5]
All values are valid (no negatives found).
```

Prompt:

Write a python code that filters out negative values from IoT sensor data. Enhance it to handle edge cases like empty lists, non-numeric entries, or all-negative inputs.

Comments & Explanation:

The function first checks if the input is a list. If not, it returns an error. It then handles empty lists and filters out non-numeric values with a warning. Only non-negative numbers are kept. Special messages are returned when all values are invalid, all are valid, or the list is empty—making the output more informative for real-world debugging.

Task 2: String Character

```
def analyze_string(text):
    # Input validation
    if not isinstance(text, str):
        return "Error: Input must be a string."
    if len(text) == 0:
```

```python
        return "Input string is empty."

    vowels = "aeiouAEIOU"
    vowel_count = consonant_count = digit_count = space_count = special_count
= 0

    for char in text:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char.isdigit():
            digit_count += 1
        elif char.isspace():
            space_count += 1
        else:
            special_count += 1

    return {
        "Input String": text,
        "Vowels": vowel_count,
        "Consonants": consonant_count,
        "Digits": digit_count,
        "Spaces": space_count,
        "Special Characters": special_count,
        "Total Characters": len(text)
    }

# Test cases
print("\n=== Task 2: String Character Analysis ===")
samples = ["Hello World! 123", "", "AEIOU", "12345!@#$%", "Python3.12"]
for s in samples:
    print(f"\nAnalyzing: '{s}'")
    print(analyze_string(s))
```

OutPut:

```
=== Task 2: String Character Analysis ===

Analyzing: 'Hello World! 123'
{'Input String': 'Hello World! 123', 'Vowels': 3, 'Consonants': 7, 'Digits':
3, 'Spaces': 2, 'Special Characters': 1, 'Total Characters': 16}

Analyzing: ''
Input string is empty.

Analyzing: 'AEIOU'
{'Input String': 'AEIOU', 'Vowels': 5, 'Consonants': 0, 'Digits': 0,
'Spaces': 0, 'Special Characters': 0, 'Total Characters': 5}
```

```
Analyzing: '12345!@#$%'
{'Input String': '12345!@#$%', 'Vowels': 0, 'Consonants': 0, 'Digits': 5,
'Spaces': 0, 'Special Characters': 5, 'Total Characters': 10}

Analyzing: 'Python3.12'
{'Input String': 'Python3.12', 'Vowels': 1, 'Consonants': 5, 'Digits': 3,
'Spaces': 0, 'Special Characters': 1, 'Total Characters': 10}
```

Prompt:

Write a Python code that counts vowels, consonants, digits, spaces, and special characters in a given string, including proper handling of edge cases.

Comments & Explanation:

This function validates that the input is a string and handles empty inputs gracefully. It iterates through each character, categorizing it using built-in string methods (isalpha, isdigit, etc.). Vowels are checked against a predefined set. The result is returned as a clear dictionary with counts for all character types, which is useful for text preprocessing or analytics.

Task 3: Palindrome Check – Tool Comparison

```
def is_palindrome_gemini_style(s):
 # Input validation
 if not isinstance(s, str):
 return "Error: Input must be a string."
 if len(s) == 0:
 return "Empty string is considered a palindrome."

 # Keep only alphanumeric characters and convert to lowercase
 cleaned = ''.join(ch.lower() for ch in s if ch.isalnum())

 if len(cleaned) == 0:
 return "No alphanumeric characters found. Treated as palindrome."

 is_pal = cleaned == cleaned[::-1]
 return {
 "Original String": s,
 "Cleaned String": cleaned,
 "Is Palindrome?": is_pal
 }

# Test cases
print("\n=== Task 3: Palindrome Checker ===")
```

```python
test_strings = [
 "A man, a plan, a canal: Panama",
 "race a car",
 "",
 "Madam",
 "12321",
 "!@#$%"
]

for ts in test_strings:
    print(f"\nInput: '{ts}'")
    print(is_palindrome_gemini_style(ts))
```

OutPut:

```
=== Task 3: Palindrome Checker ===

Input: 'A man, a plan, a canal: Panama'
{'Original String': 'A man, a plan, a canal: Panama', 'Cleaned String':
'amanaplanacanalpanama', 'Is Palindrome?': True}

Input: 'race a car'
{'Original String': 'race a car', 'Cleaned String': 'raceacar', 'Is
Palindrome?': False}

Input: ''
Empty string is considered a palindrome.

Input: 'Madam'
{'Original String': 'Madam', 'Cleaned String': 'madam', 'Is Palindrome?':
True}

Input: '12321'
{'Original String': '12321', 'Cleaned String': '12321', 'Is Palindrome?':
True}

Input: '!@#$%'
No alphanumeric characters found. Treated as palindrome.
```

Prompt:

Write a palindrome checker and compare its clarity and functionality .

Comments & Explanation:

Unlike a simple reverse-comparison, this version ignores case, spaces, and punctuation—focusing only on alphanumeric characters. This mimics real-world use (e.g., checking phrases like "A man, a plan…"). It also handles edge cases like empty strings or strings with no letters/digits, returning meaningful messages instead of just True/False.

Task 4: Code Explanation Using AI

```
def is_prime(n):
 # Input must be an integer
 if not isinstance(n, int):
 return "Error: Input must be an integer."
 # Numbers less than 2 are not prime
    if n < 2:
        return False
    # 2 is prime
    if n == 2:
        return True
    # Even numbers > 2 are not prime
    if n % 2 == 0:
        return False

    # Check odd divisors up to sqrt(n)
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True

# Test cases
print("\n=== Task 4: Prime Checker ===")
numbers = [2, 3, 4, 17, 25, -5, 1, 97]
for num in numbers:
    result = is_prime(num)
    print(f"is_prime({num}) → {result}")
```

OutPut:

```
=== Task 4: Prime Checker ===
is_prime(2) → True
is_prime(3) → True
is_prime(4) → False
is_prime(17) → True
is_prime(25) → False
is_prime(-5) → False
is_prime(1) → False
is_prime(97) → True
```

Prompt:

Prompt:

Write a code that should explain a prime-checking function line by line.

Comments & Explanation:

The is_prime function efficiently checks for primality by first handling small cases ($n < 2$, $n == 2$, even numbers), then testing only odd divisors up to $\sqrt{n}$. This reduces unnecessary computations. The comments clarify why each condition exists—like why 1 isn't prime or why we skip even divisors after 2. Understanding this logic helped me appreciate algorithmic efficiency in basic number theory functions.