Assignment – 2.2

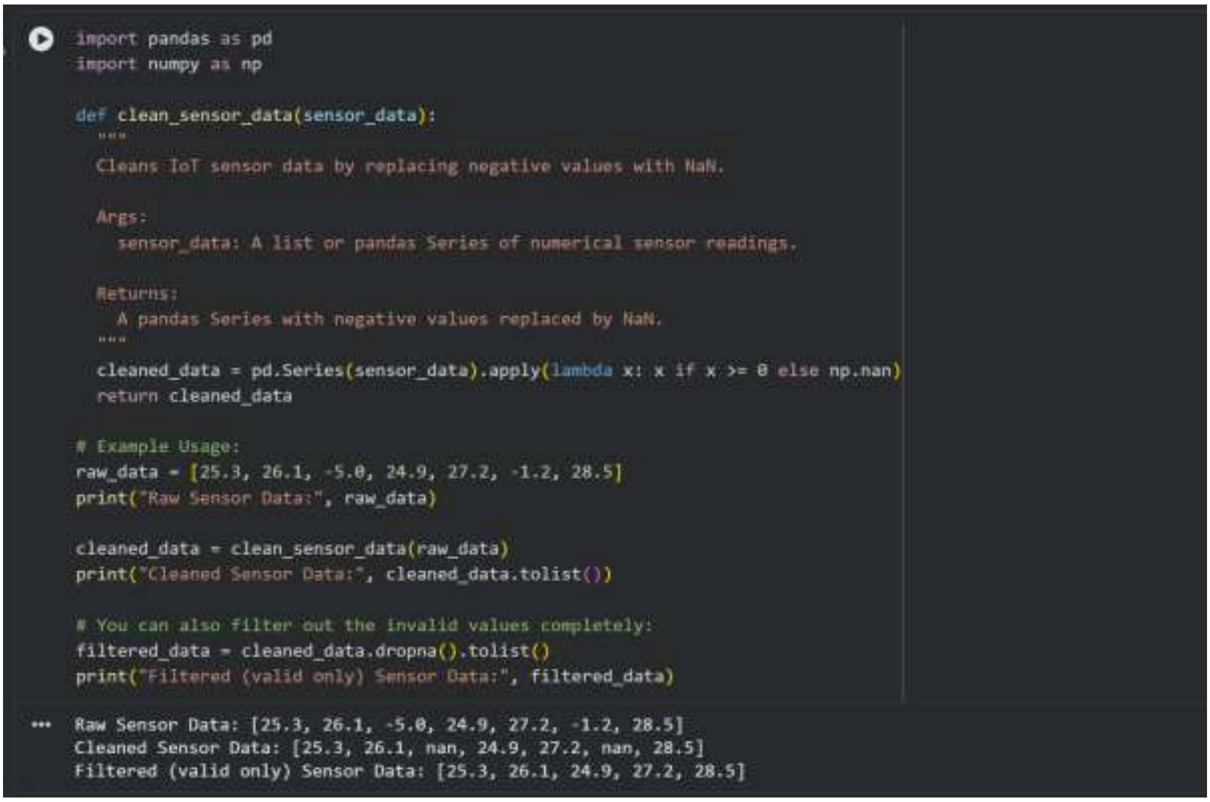
2303A51927

Task 1: Cleaning Sensor Data

PROMPT :

Generate a function that filters out all negative numbers from a list.

CODE and OUTPUT :



```
import pandas as pd
import numpy as np

def clean_sensor_data(sensor_data):
  """
  Cleans IoT sensor data by replacing negative values with NaN.

  Args:
    sensor_data: A list or pandas Series of numerical sensor readings.

  Returns:
    A pandas Series with negative values replaced by NaN.
  """
  cleaned_data = pd.Series(sensor_data).apply(lambda x: x if x >= 0 else np.nan)
  return cleaned_data

# Example Usage:
raw_data = [25.3, 26.1, -5.0, 24.9, 27.2, -1.2, 28.5]
print("Raw Sensor Data:", raw_data)

cleaned_data = clean_sensor_data(raw_data)
print("Cleaned Sensor Data:", cleaned_data.tolist())

# You can also filter out the invalid values completely:
filtered_data = cleaned_data.dropna().tolist()
print("Filtered (valid only) Sensor Data:", filtered_data)
```

```
Raw Sensor Data: [25.3, 26.1, -5.0, 24.9, 27.2, -1.2, 28.5]
Cleaned Sensor Data: [25.3, 26.1, nan, 24.9, 27.2, nan, 28.5]
Filtered (valid only) Sensor Data: [25.3, 26.1, 24.9, 27.2, 28.5]
```

Justification :

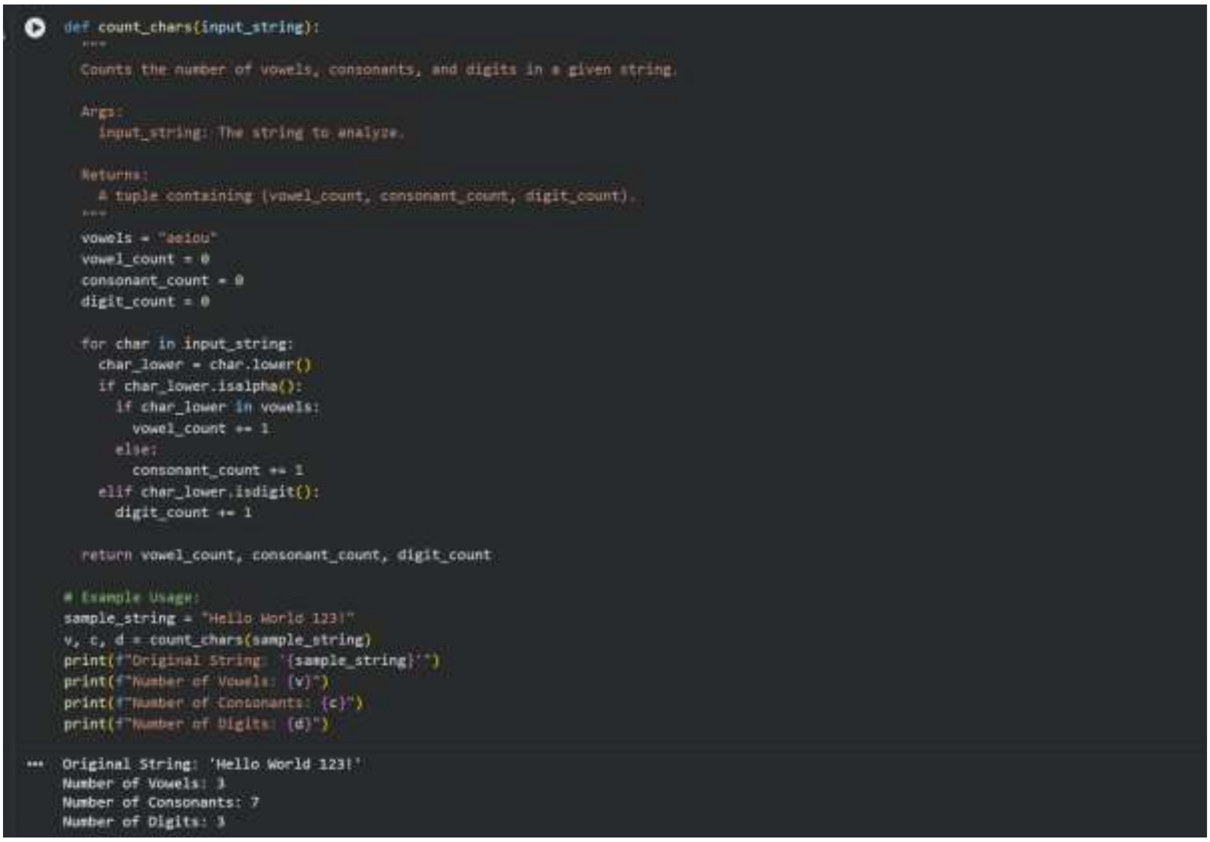
The initial task focused on eliminating negative readings from IoT sensor datasets. In practical scenarios, sensor data frequently includes noise or invalid values, making data cleaning a crucial part of the preprocessing stage. Gemini provided a concise and effective solution by using list comprehension to remove negative values. This highlights how AI-powered tools can rapidly generate clean, optimized, and easy-to-understand code for data preprocessing, helping developers save time while reducing the chances of human error.

Task 2: String Character Analysis

PROMPT :

Generate a Python function that counts vowels, consonants, and digits in a string.

CODE and OUTPUT :



```
def count_chars(input_string):
  """
  Counts the number of vowels, consonants, and digits in a given string.

  Args:
    input_string: The string to analyze.

  Returns:
    A tuple containing (vowel_count, consonant_count, digit_count).
  """
  vowels = "aeiou"
  vowel_count = 0
  consonant_count = 0
  digit_count = 0

  for char in input_string:
    char_lower = char.lower()
    if char_lower.isalpha():
      if char_lower in vowels:
        vowel_count += 1
      else:
        consonant_count += 1
    elif char_lower.isdigit():
      digit_count += 1

  return vowel_count, consonant_count, digit_count

# Example Usage:
sample_string = "Hello World 123!"
v, c, d = count_chars(sample_string)
print(f"Original String: '{sample_string}'")
print(f"Number of Vowels: {v}")
print(f"Number of Consonants: {c}")
print(f"Number of Digits: {d}")
```

```
Original String: 'Hello World 123!'
Number of Vowels: 3
Number of Consonants: 7
Number of Digits: 3
```

Justification :

This task involved identifying and counting vowels, consonants, and digits within a given string, a common requirement in text analysis and input validation. Gemini produced a well-organized function that effectively used built-in string methods to classify each character. This illustrates how AI tools can accurately implement conditional logic and create reusable, clean functions. Overall, the solution shows how AI can be valuable not only for learning programming concepts but also for handling real-world text-processing tasks efficiently.

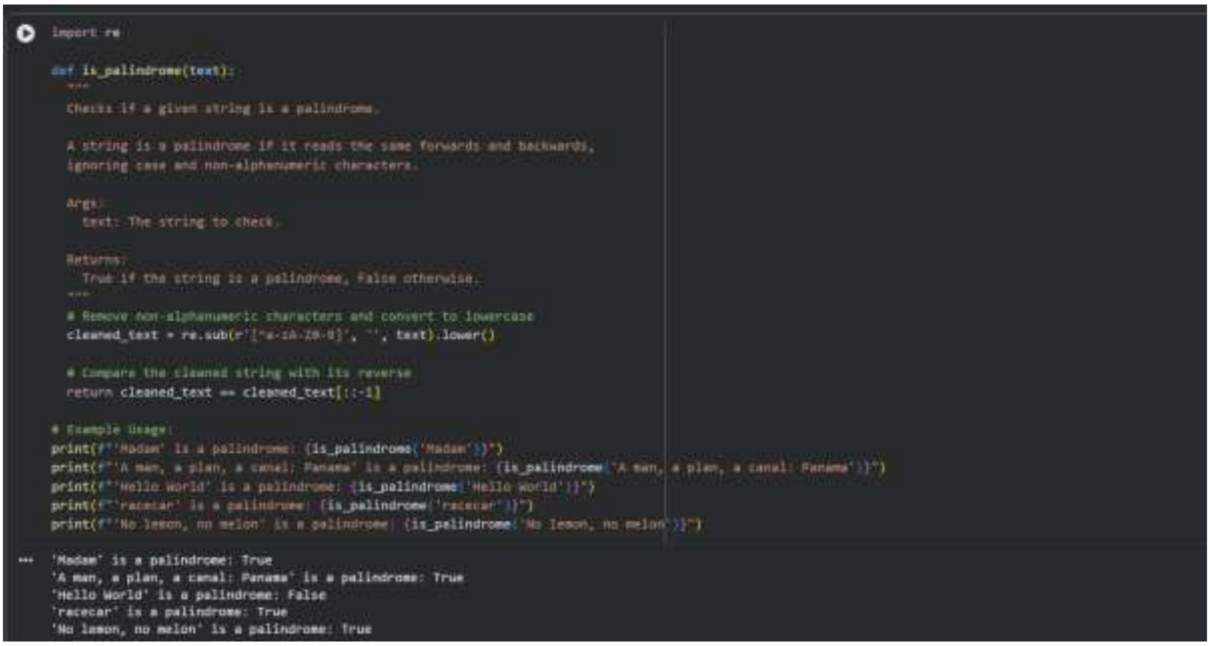
Task 3: Palindrome Check – Tool Comparison

PROMPT :

Generate a palindrome-checking function.

CODE and OUTPUT :

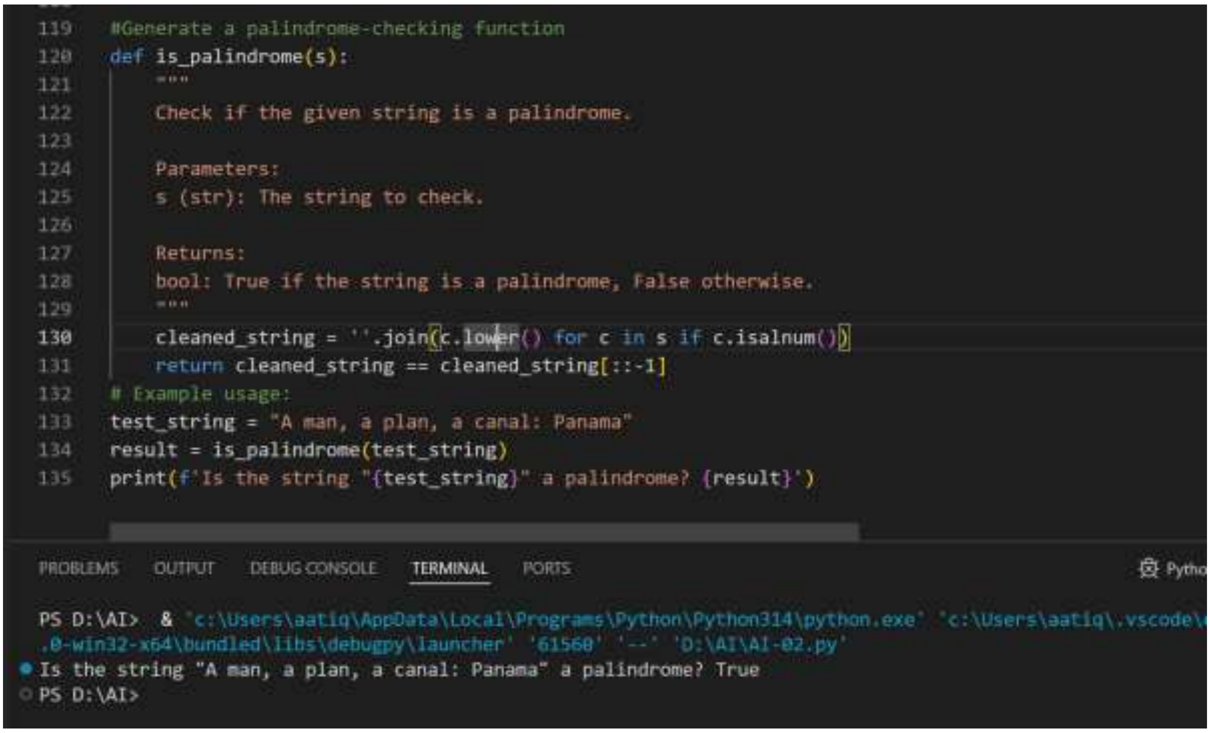
Gemini **:**



```
import re

def is_palindrome(text):
  """
  Checks if a given string is a palindrome.

  A string is a palindrome if it reads the same forwards and backwards,
  ignoring case and non-alphanumeric characters.

  Args:
    text: The string to check.

  Returns:
    True if the string is a palindrome, False otherwise.
  """
  # Remove non-alphanumeric characters and convert to lowercase
  cleaned_text = re.sub(r'[^a-zA-Z0-9]', '', text).lower()

  # Compare the cleaned string with its reverse
  return cleaned_text == cleaned_text[::-1]

# Example Usage:
print(f"'Madam' is a palindrome: {is_palindrome('Madam')}")
print(f"'A man, a plan, a canal: Panama' is a palindrome: {is_palindrome('A man, a plan, a canal: Panama')}")
print(f"'Hello World' is a palindrome: {is_palindrome('Hello World')}")
print(f"'racecar' is a palindrome: {is_palindrome('racecar')}")
print(f"'No lemon, no melon' is a palindrome: {is_palindrome('No lemon, no melon')}")
```

```
'Madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'Hello World' is a palindrome: False
'racecar' is a palindrome: True
'No lemon, no melon' is a palindrome: True
```

Co-pilot :



```
#Generate a palindrome-checking function
def is_palindrome(s):
    """
    Check if the given string is a palindrome.

    Parameters:
    s (str): The string to check.

    Returns:
    bool: True if the string is a palindrome, False otherwise.
    """
    cleaned_string = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned_string == cleaned_string[::-1]
# Example usage:
test_string = "A man, a plan, a canal: Panama"
result = is_palindrome(test_string)
print(f'Is the string "{test_string}" a palindrome? {result}')
```

```
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\
.0-win32-x64\bundled\libs\debugpy\launcher' '61560' '--' 'D:\AI\AI-02.py'
Is the string "A man, a plan, a canal: Panama" a palindrome? True
PS D:\AI>
```

Justification :

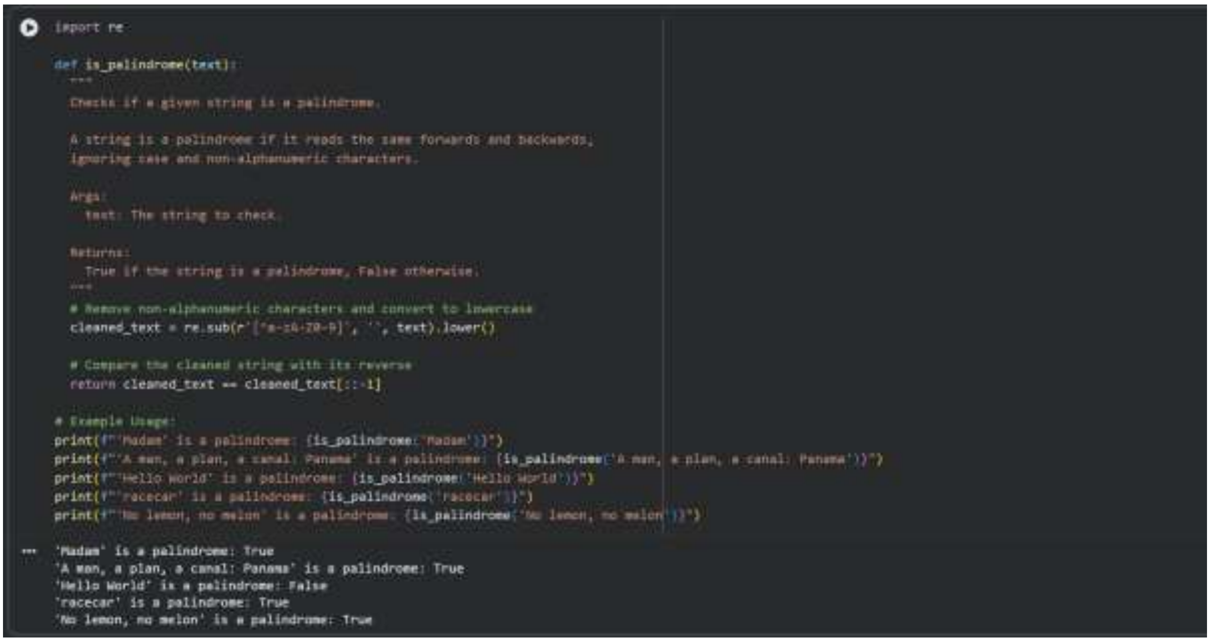
This task involved generating palindrome-checking functions using both Gemini and Copilot. Gemini delivered a concise and optimized solution, focusing on brevity and efficiency, while Copilot presented a more elaborate, step-by-step implementation that clearly explained the underlying logic. This comparison highlights that different AI tools excel in different areas: Gemini is well suited for writing clean, professional-grade code, whereas Copilot is particularly useful for gaining a deeper understanding of the logic. Overall, the task underscored the value of evaluating multiple AI tools to select the one that best fits a specific development or learning need.

Task 4: Code Explanation Using AI

PROMPT :

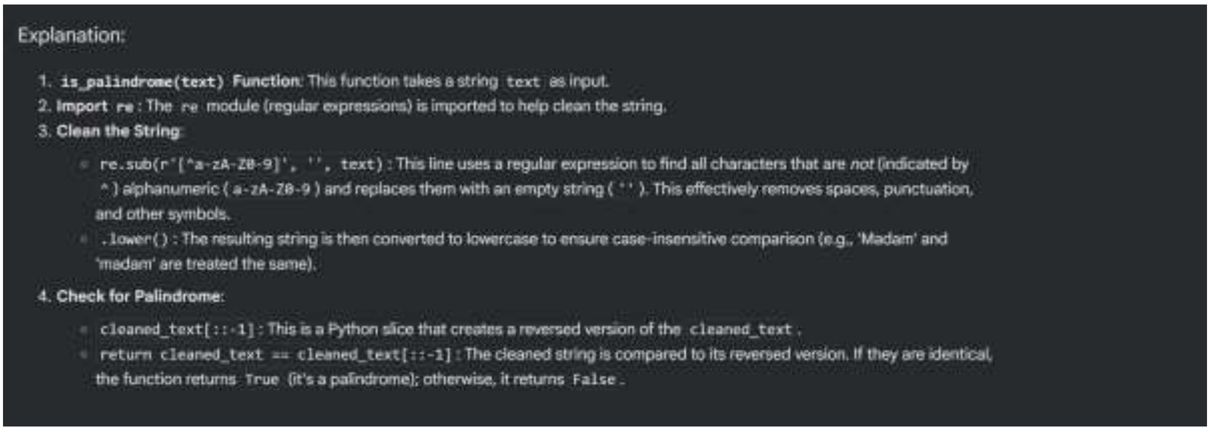
Explain a Python function (prime check OR palindrome check) line by line.

CODE and OUTPUT :



```
import re

def is_palindrome(text):
  """
  Checks if a given string is a palindrome.

  A string is a palindrome if it reads the same forwards and backwards,
  ignoring case and non-alphanumeric characters.

  Args:
    text: The string to check.

  Returns:
    True if the string is a palindrome, False otherwise.
  """
  # Remove non-alphanumeric characters and convert to lowercase
  cleaned_text = re.sub(r'[^a-zA-Z0-9]', '', text).lower()

  # Compare the cleaned string with its reverse
  return cleaned_text == cleaned_text[::-1]

# Example Usage:
print(f"'Madam' is a palindrome: {is_palindrome('Madam')}")
print(f"'A man, a plan, a canal: Panama' is a palindrome: {is_palindrome('A man, a plan, a canal: Panama')}")
print(f"'Hello World' is a palindrome: {is_palindrome('Hello World')}")
print(f"'racecar' is a palindrome: {is_palindrome('racecar')}")
print(f"'No lemon, no melon' is a palindrome: {is_palindrome('No lemon, no melon')}")
```

```
'Madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'Hello World' is a palindrome: False
'racecar' is a palindrome: True
'No lemon, no melon' is a palindrome: True
```



Explanation:

1. **`is_palindrome(text)` Function**: This function takes a string `text` as input.
2. **Import `re`**: The `re` module (regular expressions) is imported to help clean the string.
3. **Clean the String**:
    - `re.sub(r'[^a-zA-Z0-9]', '', text)` : This line uses a regular expression to find all characters that are *not* (indicated by `^`) alphanumeric (`a-zA-Z0-9`) and replaces them with an empty string (`''`). This effectively removes spaces, punctuation, and other symbols.
    - `.lower()` : The resulting string is then converted to lowercase to ensure case-insensitive comparison (e.g., 'Madam' and 'madam' are treated the same).
4. **Check for Palindrome**:
    - `cleaned_text[::-1]` : This is a Python slice that creates a reversed version of the `cleaned_text` .
    - `return cleaned_text == cleaned_text[::-1]` : The cleaned string is compared to its reversed version. If they are identical, the function returns `True` (it's a palindrome); otherwise, it returns `False` .

Justification:

This task highlighted Gemini's strength in clearly explaining unfamiliar code on a line-by-line basis. The detailed breakdown of the palindrome-checking function made it easier to understand both the underlying logic and the applied optimizations. This demonstrates that AI tools are valuable not just for generating code, but also for acting as effective learning aids that help users grasp concepts and improve their understanding of efficient programming practices.