AI assisted coding

2303A52490

ADITHYA

ASSIGNMENT 2

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

and Cursor AI

Task 1: Cleaning Sensor Data

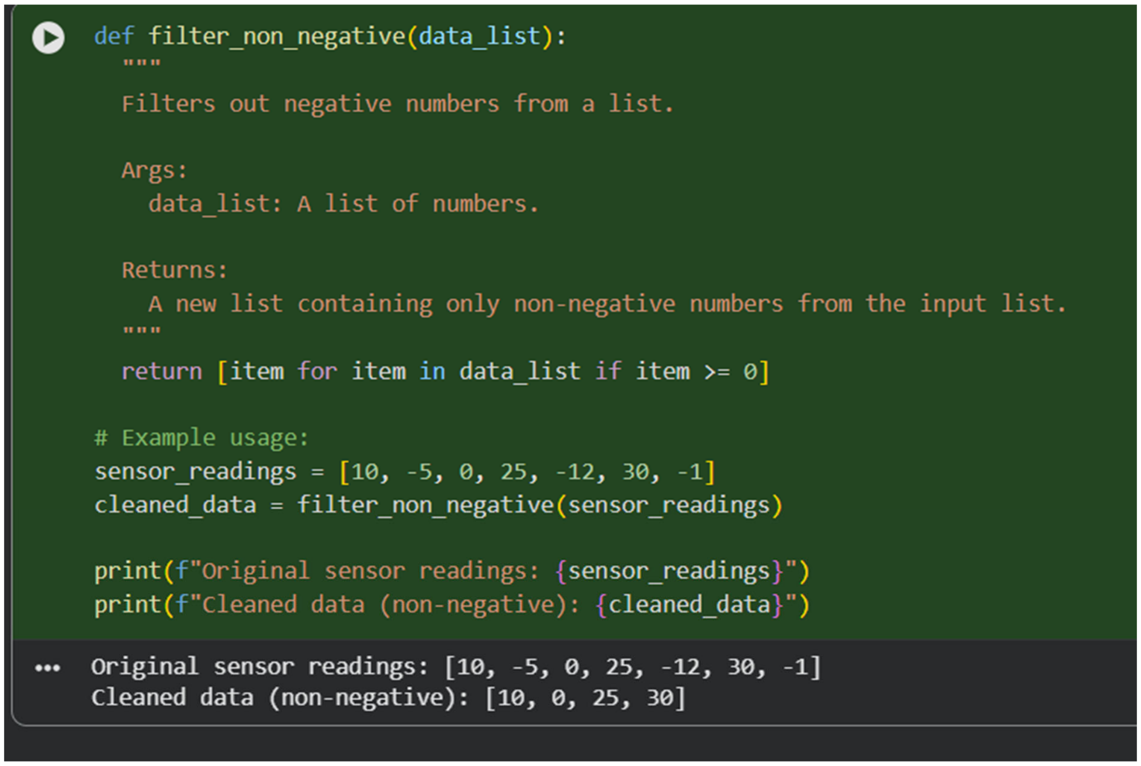
❖ Scenario:

❖ You are cleaning IoT sensor data where negative values are invalid.

❖ Task:

Use Gemini in Colab to generate a function that filters out all negative

numbers from a list.



```python
def filter_non_negative(data_list):
  """
  Filters out negative numbers from a list.

  Args:
    data_list: A list of numbers.

  Returns:
    A new list containing only non-negative numbers from the input list.
  """
  return [item for item in data_list if item >= 0]

# Example usage:
sensor_readings = [10, -5, 0, 25, -12, 30, -1]
cleaned_data = filter_non_negative(sensor_readings)

print(f"Original sensor readings: {sensor_readings}")
print(f"Cleaned data (non-negative): {cleaned_data}")
```

```
Original sensor readings: [10, -5, 0, 25, -12, 30, -1]
Cleaned data (non-negative): [10, 0, 25, 30]
```

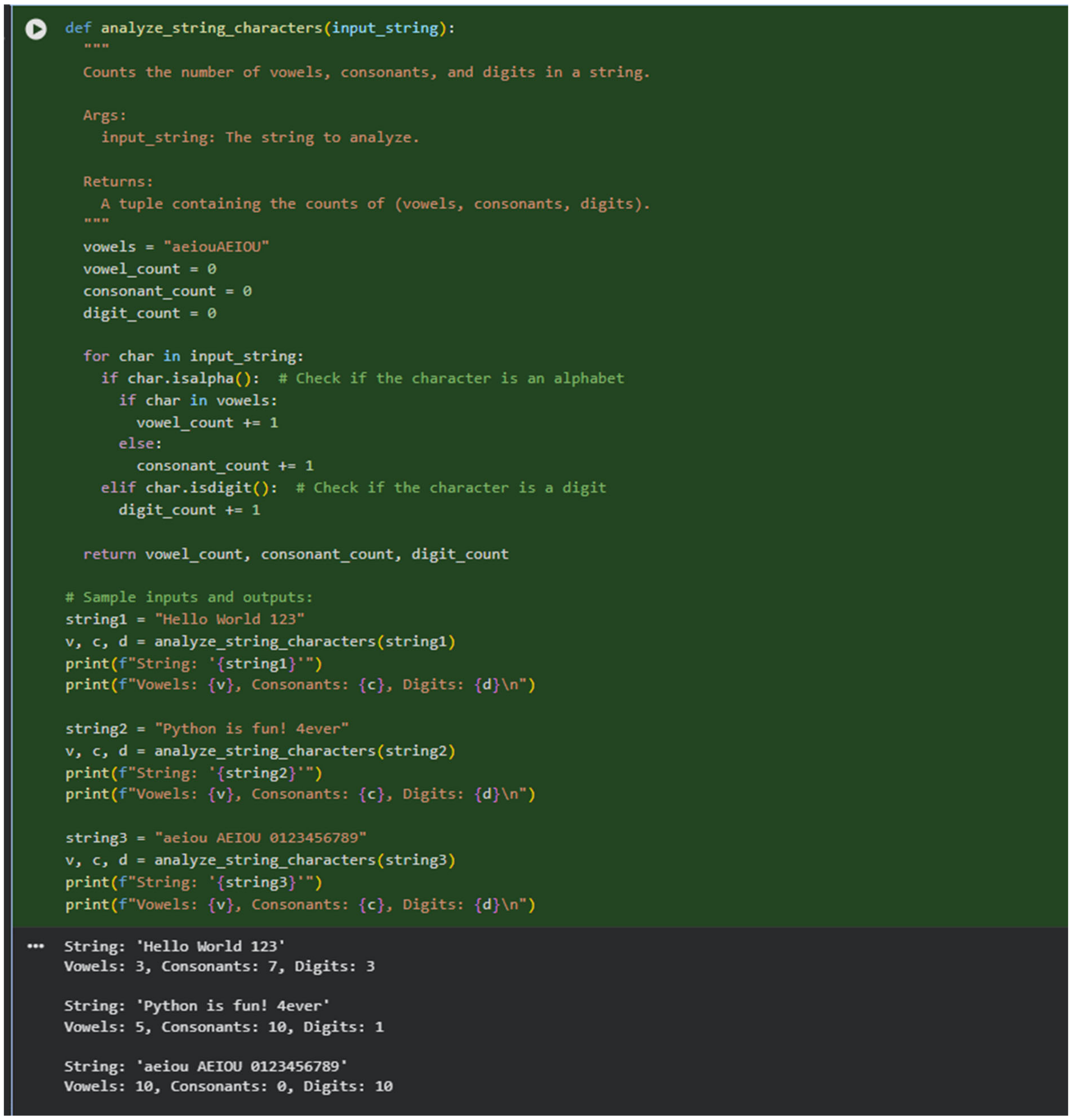
Task 2: String Character Analysis

❖ Scenario:

You are building a text-analysis feature.

❖ Task:

Use Gemini to generate a Python function that counts vowels, consonants, and digits in a string.



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

  Args:
    input_string: The string to analyze.

  Returns:
    A tuple containing the counts of (vowels, consonants, digits).
  """
  vowels = "aeiouAEIOU"
  vowel_count = 0
  consonant_count = 0
  digit_count = 0

  for char in input_string:
    if char.isalpha():  # Check if the character is an alphabet
      if char in vowels:
        vowel_count += 1
      else:
        consonant_count += 1
    elif char.isdigit():  # Check if the character is a digit
      digit_count += 1

  return vowel_count, consonant_count, digit_count

# Sample inputs and outputs:
string1 = "Hello World 123"
v, c, d = analyze_string_characters(string1)
print(f"String: '{string1}'")
print(f"Vowels: {v}, Consonants: {c}, Digits: {d}\n")

string2 = "Python is fun! 4ever"
v, c, d = analyze_string_characters(string2)
print(f"String: '{string2}'")
print(f"Vowels: {v}, Consonants: {c}, Digits: {d}\n")

string3 = "aeiou AEIOU 0123456789"
v, c, d = analyze_string_characters(string3)
print(f"String: '{string3}'")
print(f"Vowels: {v}, Consonants: {c}, Digits: {d}\n")
```

```
String: 'Hello World 123'
Vowels: 3, Consonants: 7, Digits: 3

String: 'Python is fun! 4ever'
Vowels: 5, Consonants: 10, Digits: 1

String: 'aeiou AEIOU 0123456789'
Vowels: 10, Consonants: 0, Digits: 10
```

Task 3

❖ Scenario:

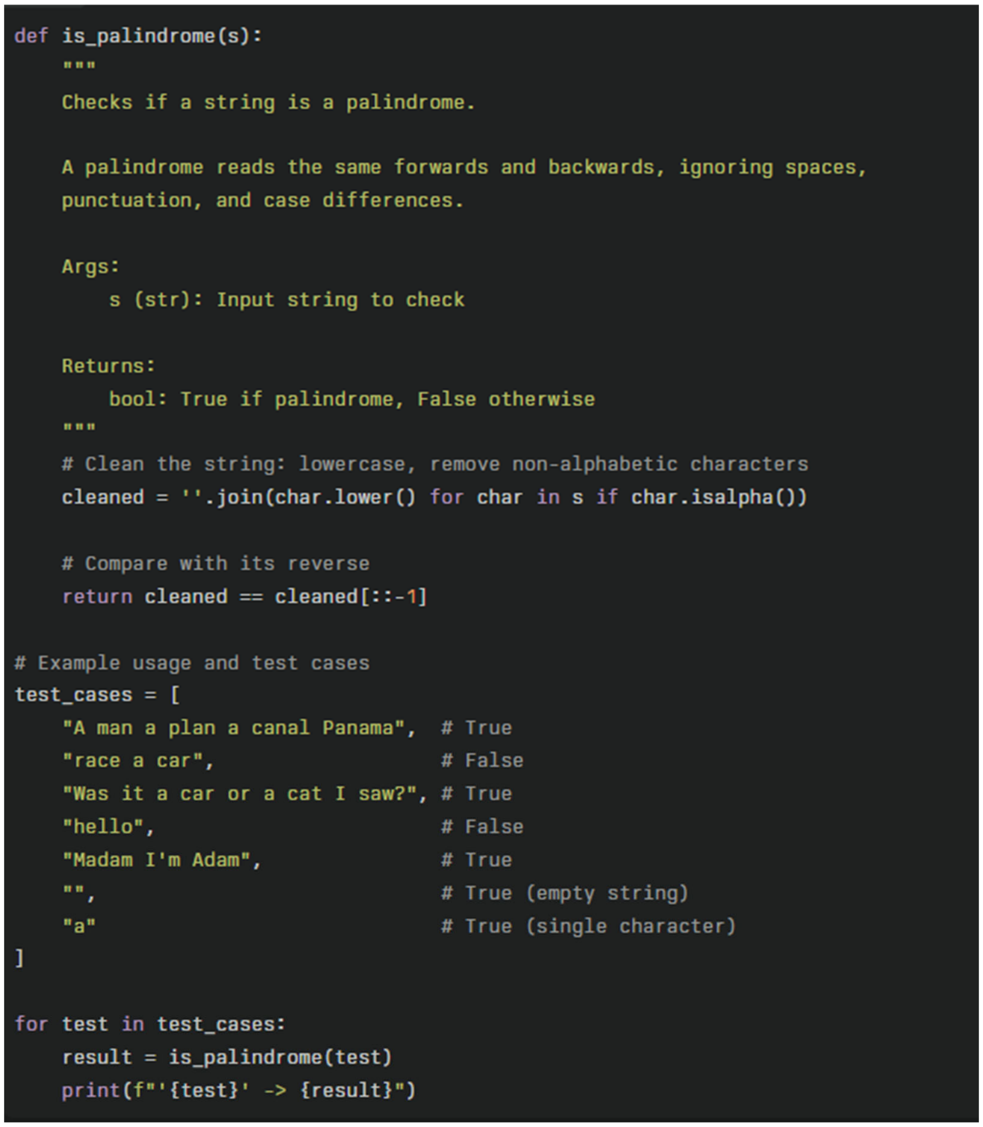
You must decide which AI tool is clearer for string logic.

❖ Task:

Generate a palindrome-checking function using Gemini and Copilot, then

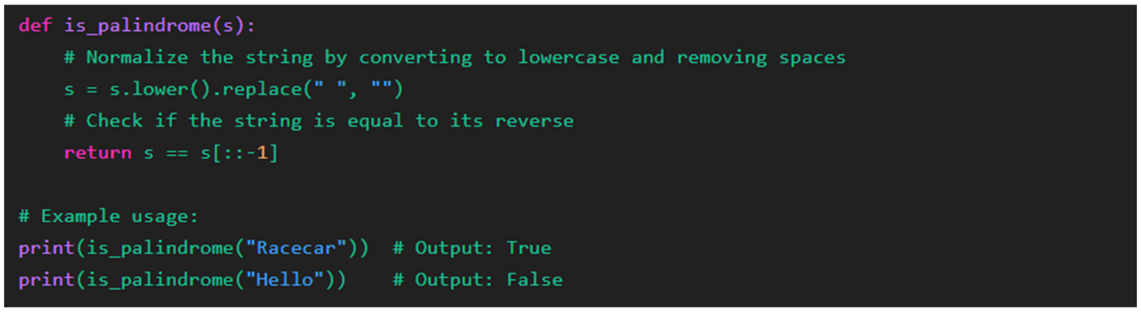
compare the results.

GEMINI



```python
def is_palindrome(s):
    """
    Checks if a string is a palindrome.

    A palindrome reads the same forwards and backwards, ignoring spaces,
    punctuation, and case differences.

    Args:
        s (str): Input string to check

    Returns:
        bool: True if palindrome, False otherwise
    """
    # Clean the string: lowercase, remove non-alphabetic characters
    cleaned = ''.join(char.lower() for char in s if char.isalpha())

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

# Example usage and test cases
test_cases = [
    "A man a plan a canal Panama",  # True
    "race a car",                   # False
    "Was it a car or a cat I saw?", # True
    "hello",                        # False
    "Madam I'm Adam",               # True
    "",                             # True (empty string)
    "a"                             # True (single character)
]

for test in test_cases:
    result = is_palindrome(test)
    print(f"'{test}' -> {result}")
```

COPILOT



```
def is_palindrome(s):
    # Normalize the string by converting to lowercase and removing spaces
    s = s.lower().replace(" ", "")
    # Check if the string is equal to its reverse
    return s == s[::-1]

# Example usage:
print(is_palindrome("Racecar"))  # Output: True
print(is_palindrome("Hello"))    # Output: False
```

OBSERVATIONS

3 Key Differences:

- **Character filtering**: First code removes *only spaces*, second removes *all nonalphabetic characters*
- **Code style**: First is minimal (no docstring, modifies input), second is production-ready (docstring, doesn't modify input, list comprehension)
- **Test coverage**: First tests 2 simple cases, second tests 7 diverse cases including punctuation, empty string, and edge cases

Task 4: Code Explanation Using AI

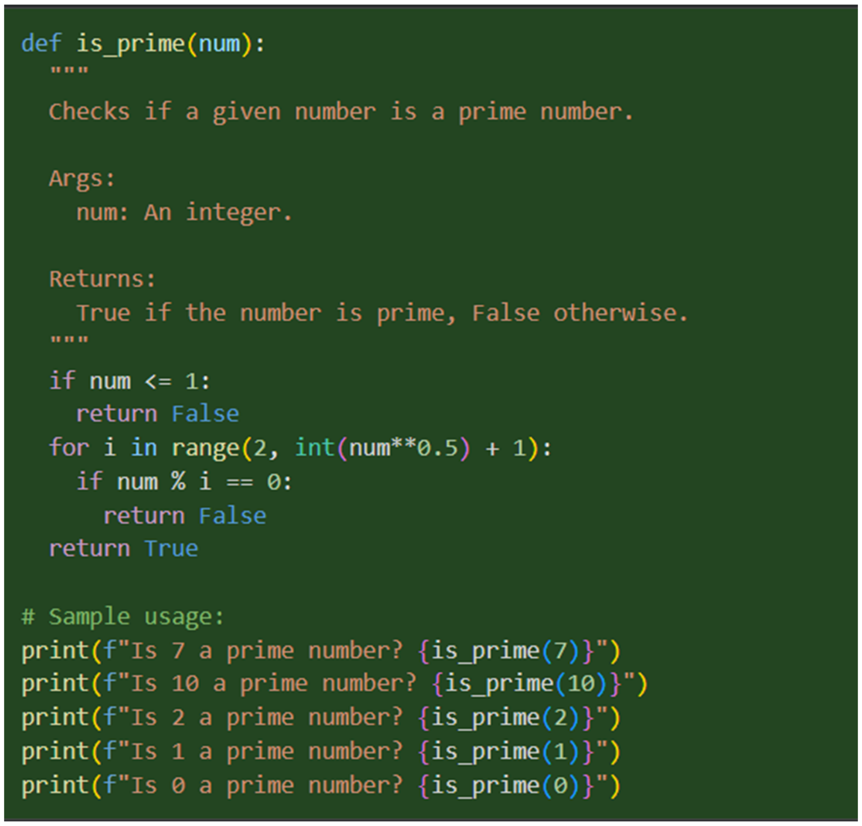
❖ Scenario:

You are reviewing unfamiliar code written by another developer.

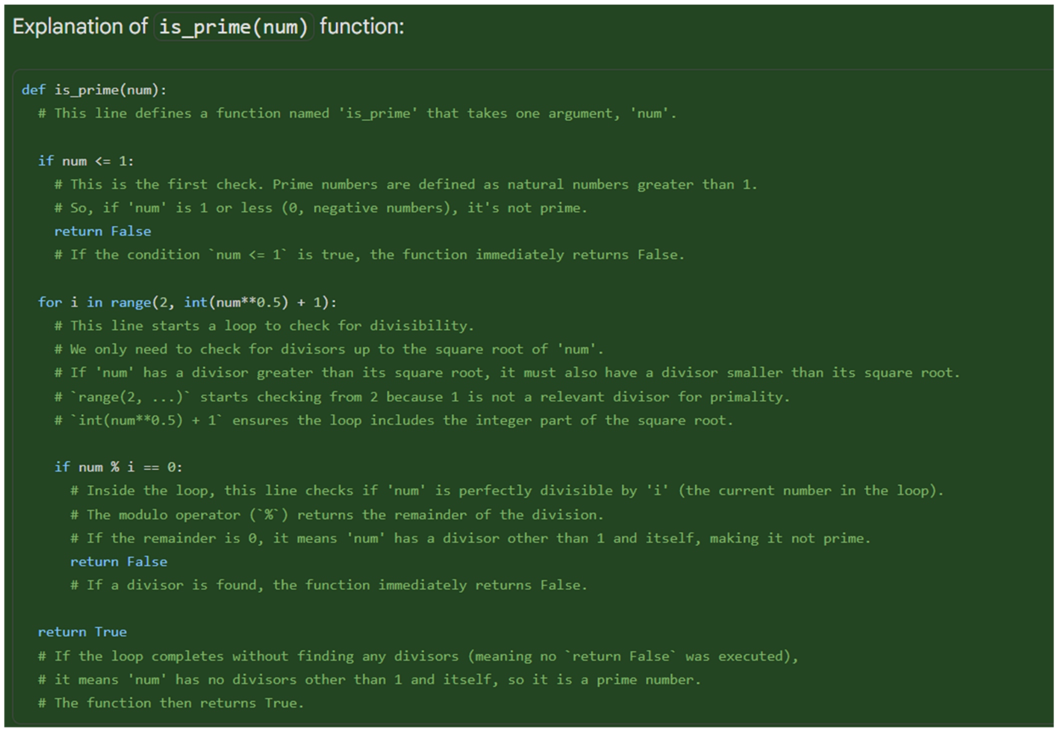
❖ Task:

Ask Gemini to explain a Python function (prime check OR palindrome

check) line by line.



```python
def is_prime(num):
  """
  Checks if a given number is a prime number.

  Args:
    num: An integer.

  Returns:
    True if the number is prime, False otherwise.
  """
  if num <= 1:
    return False
  for i in range(2, int(num**0.5) + 1):
    if num % i == 0:
      return False
  return True

# Sample usage:
print(f"Is 7 a prime number? {is_prime(7)}")
print(f"Is 10 a prime number? {is_prime(10)}")
print(f"Is 2 a prime number? {is_prime(2)}")
print(f"Is 1 a prime number? {is_prime(1)}")
print(f"Is 0 a prime number? {is_prime(0)}")
```



Explanation of `is_prime(num)` function:

```python
def is_prime(num):
  # This line defines a function named 'is_prime' that takes one argument, 'num'.

  if num <= 1:
    # This is the first check. Prime numbers are defined as natural numbers greater than 1.
    # So, if 'num' is 1 or less (0, negative numbers), it's not prime.
    return False
    # If the condition `num <= 1` is true, the function immediately returns False.

  for i in range(2, int(num**0.5) + 1):
    # This line starts a loop to check for divisibility.
    # We only need to check for divisors up to the square root of 'num'.
    # If 'num' has a divisor greater than its square root, it must also have a divisor smaller than its square root.
    # `range(2, ...)` starts checking from 2 because 1 is not a relevant divisor for primality.
    # `int(num**0.5) + 1` ensures the loop includes the integer part of the square root.

    if num % i == 0:
      # Inside the loop, this line checks if 'num' is perfectly divisible by 'i' (the current number in the loop).
      # The modulo operator (`%`) returns the remainder of the division.
      # If the remainder is 0, it means 'num' has a divisor other than 1 and itself, making it not prime.
      return False
      # If a divisor is found, the function immediately returns False.

  return True
  # If the loop completes without finding any divisors (meaning no `return False` was executed),
  # it means 'num' has no divisors other than 1 and itself, so it is a prime number.
  # The function then returns True.
```

MY COMMENTS ON THIS

I understand the prime function now. The square root check make sense because factors come in pairs, so only need to test up to √n. Loop starts from 2 skip 1 since 1 not count. Early return when find divisor save time. If no divisor after loop, it is prime.