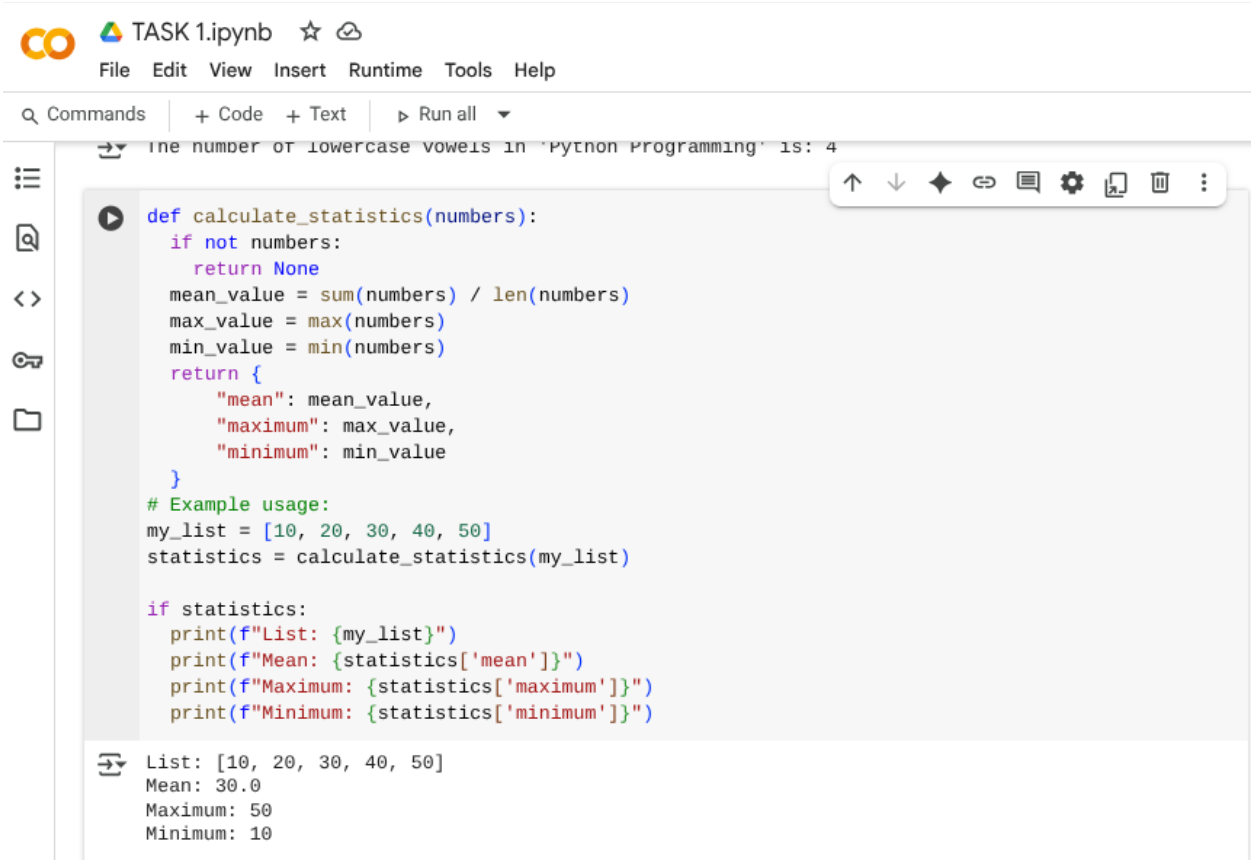
ASSIGNMENT-2

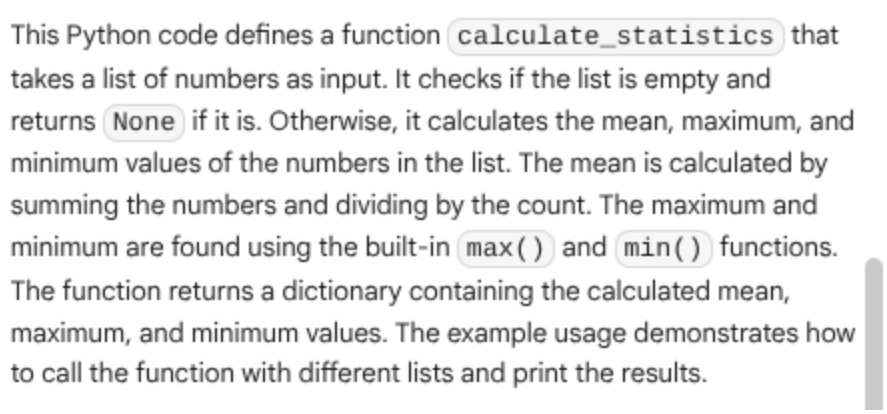
Task-1:

write a python program that reads a list of numbers and calculates mean,maximum and minimum values.

Code and Output:



Explanation:



Task-2:

write a python function that checks the given number is an Armstrong or not.

Comparision between gemini and copilot-

Gemini:

Code and Output:

```python
def is_armstrong_number(number):
  # Convert the number to a string to easily get the number of digits and individual
  num_str = str(number)
  num_digits = len(num_str)
  sum_of_powers = 0

  # Calculate the sum of each digit raised to the power of the number of digits
  for digit_char in num_str:
    digit = int(digit_char)
    sum_of_powers += digit ** num_digits

  # Check if the sum of powers equals the original number
  return sum_of_powers == number

# Example usage:
num1 = 153
if is_armstrong_number(num1):
  print(f"{num1} is an Armstrong number.")
else:
  print(f"{num1} is not an Armstrong number.")
```

```
153 is an Armstrong number.
123 is not an Armstrong number.
9 is an Armstrong number.
```

Explanation:

The provided Python code defines a function called `is_armstrong_number` that determines if a given integer is an Armstrong number.

An Armstrong number is a number that equals the sum of its own digits, each raised to the power of the total number of digits in the number.

The function begins by converting the input `number` into a string to easily access its individual digits and determine the total count of digits (`num_digits`).

It then initializes a variable `sum_of_powers` to zero.

The code iterates through each character (digit) in the number's string representation. For each digit character, it converts it back to an integer.

This integer digit is then raised to the power of `num_digits`, and the result is added to `sum_of_powers`.

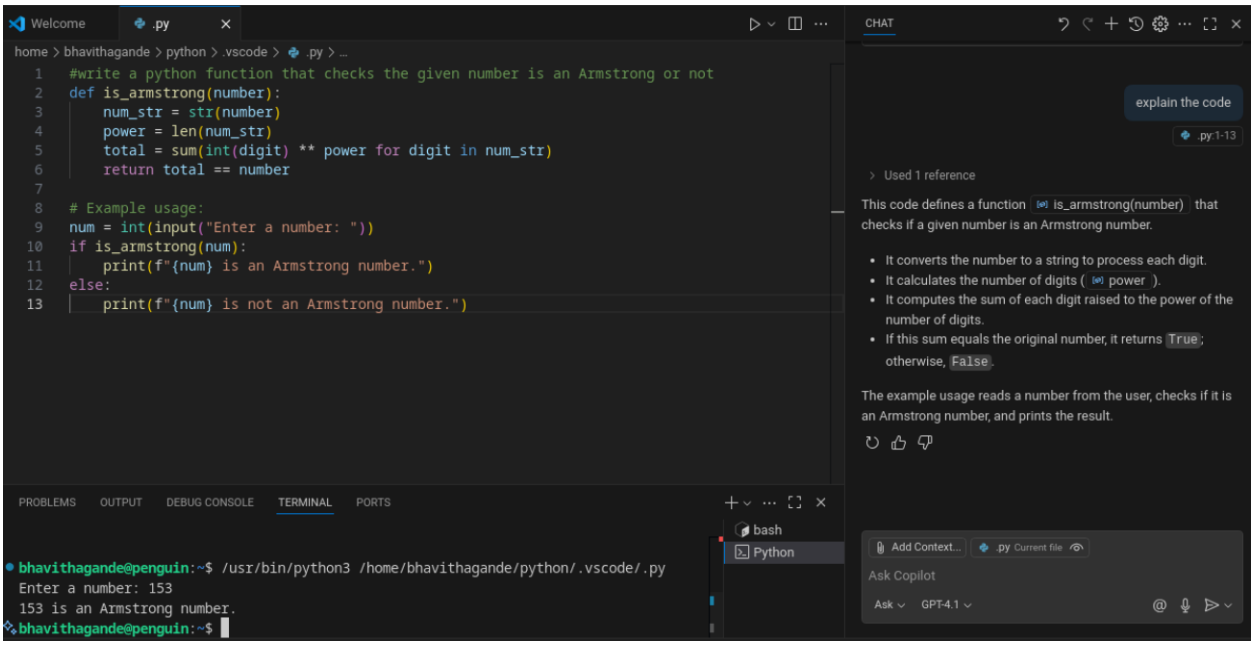
After processing all the digits, the function compares the calculated `sum_of_powers` with the original `number`.

If they are equal, the function returns `True`, indicating that the number is an Armstrong number.

If they are not equal, the function returns `False`.

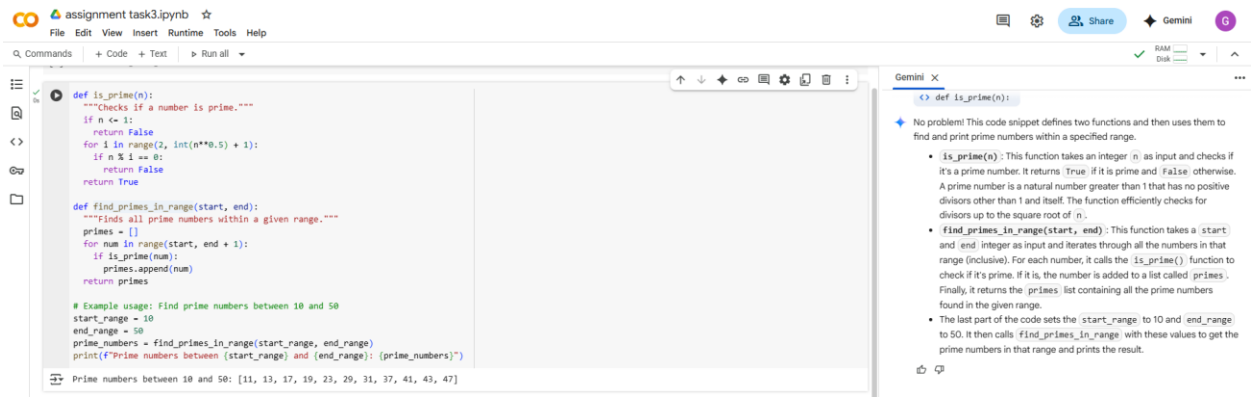
The example usage demonstrates how to call the `is_armstrong_number` function with different numbers (153, 123, and 9) and prints whether each number is an Armstrong number or not based on the function's return value.

Copilot:



Task-3:

Code,Output and Explanation:



Task-4:

Code and Output:

```python
#write a python function that calculates the sum of the first N Natural Numbers

def sum_natural_numbers_iterative(n):
    """
    Calculate the sum of first N natural numbers using iteration.

    Args:
        n (int): A positive integer

    Returns:
        int: Sum of first N natural numbers (1 + 2 + 3 + ... + n)
    """
    if n <= 0:
        return 0

    total = 0
    for i in range(1, n + 1):
        total += i
    return total


def sum_natural_numbers_formula(n):
    """
    Calculate the sum of first N natural numbers using mathematical formula.
    Formula: sum = n * (n + 1) / 2

    Args:
        n (int): A positive integer

    Returns:
        int: Sum of first N natural numbers
    """
    if n <= 0:
        return 0

    return n * (n + 1) // 2
```

```python
def sum_natural_numbers_recursive(n):
    """
    Calculate the sum of first N natural numbers using recursion.

    Args:
        n (int): A positive integer

    Returns:
        int: Sum of first N natural numbers
    """
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return n + sum_natural_numbers_recursive(n - 1)


# Example usage and testing
if __name__ == "__main__":
    # Test cases
    test_numbers = [5, 10, 100]

    for num in test_numbers:
        print(f"Sum of first {num} natural numbers:")
        print(f"  Iterative: {sum_natural_numbers_iterative(num)}")
        print(f"  Formula: {sum_natural_numbers_formula(num)}")
        print(f"  Recursive: {sum_natural_numbers_recursive(num)}")
        print()
```

Explanation:

Functions Implemented:

1. `sum_natural_numbers_iterative(n)` - Uses a simple loop to add numbers from 1 to n
2. `sum_natural_numbers_formula(n)` - Uses the mathematical formula: `n * (n + 1) / 2` (most efficient)
3. `sum_natural_numbers_recursive(n)` - Uses recursion to calculate the sum

Key Features:

- **Input validation:** All functions handle non-positive numbers by returning 0
- **Documentation:** Each function has clear docstrings explaining parameters and return values
- **Test cases:** The code includes example usage with test cases for N = 5, 10, and 100
- **Expected results:** Comments show what the results should be

Performance Comparison:

- **Formula method** is the most efficient (O(1) time complexity)
- **Iterative method** has O(n) time complexity
- **Recursive method** has O(n) time complexity but uses more memory due to call stack

You can run the file to see all three methods in action with the test cases. The formula method is generally recommended for production use due to its efficiency, especially for large values of N.

Task-5:

Code,Output and Explanation: