**TASK=1**

**Code generated by Gemini colab:**

def analyze\_numbers(numbers):

  """

  Calculates the mean, minimum, and maximum values from a list of numbers.

  Args:

    numbers: A list of numbers.

  Returns:

    A tuple containing the mean, minimum, and maximum values.

    Returns None if the list is empty.

  """

  if not numbers:

    return None

  minimum = min(numbers)

  maximum = max(numbers)

  mean = sum(numbers) / len(numbers)

  return mean, minimum, maximum

# Example usage:

my\_list = [10, 20, 5, 30, 15]

result = analyze\_numbers(my\_list)

if result:

  mean\_value, min\_value, max\_value = result

  print(f"List: {my\_list}")

  print(f"Mean: {mean\_value}")

  print(f"Minimum: {min\_value}")

  print(f"Maximum: {max\_value}")

else:

  print("The list is empty.")

**OUTPUT:**

List: [10, 20, 5, 30, 15]

Mean: 16.0

Minimum: 5

Maximum: 30

**Code generated by Copilot:**

def calculate\_stats(numbers):

    """

    Reads a list of numbers and calculates the mean, minimum, and maximum value.

    Parameters:

    - numbers (list of float/int): The list of numbers.

    Returns:

    - mean (float): The average value.

    - minimum (float/int): The smallest value.

    - maximum (float/int): The largest value.

    """

    if not numbers:

        print("The list is empty.")

        return None, None, None

    mean = sum(numbers) / len(numbers)

    minimum = min(numbers)

    maximum = max(numbers)

    return mean, minimum, maximum

# Example usage:

num\_list = [float(x) for x in input("Enter numbers separated by space: ").split()]

mean, minimum, maximum = calculate\_stats(num\_list)

print(f"Mean: {mean}")

print(f"Minimum: {minimum}")

print(f"Maximum: {maximum}")

**OUTPUT:**

Enter numbers separated by space:

10 20 5 30 15

Mean: 16.0

Minimum: 5.0

Maximum: 30.0

**Difference:**

The code generated by the colab is static-way whereas the code generated by copilot is in dynamic-way.

**TASK=3**

**Code snippet by Gemini Colab:**

def find\_prime\_numbers\_in\_range(start, end):

  """

  Finds all prime numbers within a given range (inclusive).

  Args:

    start: The starting integer of the range.

    end: The ending integer of the range.

  Returns:

    A list of prime numbers within the specified range.

  """

  prime\_numbers = []

  for num in range(start, end + 1):

    # Prime numbers are greater than 1

    if num > 1:

      for i in range(2, int(num\*\*0.5) + 1):

        if (num % i) == 0:

          break  # Not a prime number

      else:

        prime\_numbers.append(num)  # It is a prime number

  return prime\_numbers

# Example usage:

start\_range = 10

end\_range = 50

prime\_list = find\_prime\_numbers\_in\_range(start\_range, end\_range)

print(f"Prime numbers between {start\_range} and {end\_range}: {prime\_list}")  
  
**OUTPUT:**

Prime numbers between 10 and 50:[11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]  
  
  
**CODE EXPLAINATION GIVEN BY GEMINI:**

Here's a breakdown of the code:

* **def find\_prime\_numbers\_in\_range(start, end):**: This line defines the function named find\_prime\_numbers\_in\_range that accepts two arguments: start and end, representing the beginning and end of the range to check.
* **prime\_numbers = []**: This initializes an empty list called prime\_numbers which will store the prime numbers found within the range.
* **for num in range(start, end + 1):**: This loop iterates through each number from start to end (inclusive).
* **if num > 1:**: This condition checks if the current number num is greater than 1, because prime numbers must be greater than 1.
* **`for i in range(2, int(num**0.5) + 1):\*\*: This nested loop iterates from 2 up to the square root ofnum`. We only need to check for divisors up to the square root of a number to determine if it's prime.
* **if (num % i) == 0:**: This condition checks if num is divisible by i. If it is, it means num has a divisor other than 1 and itself, so it's not a prime number.
* **break**: If a divisor is found, the inner loop is exited using break because we already know the number is not prime.
* **else:**: This else block is associated with the inner for loop. It is executed only if the inner loop completes without finding any divisors (i.e., the break statement was not executed). This indicates that num is a prime number.
* **prime\_numbers.append(num)**: If num is determined to be prime, it is added to the prime\_numbers list.
* **return prime\_numbers**: After checking all numbers in the range, the function returns the prime\_numbers list containing all the prime numbers found.
* **Example Usage**: The code then demonstrates how to use the function by setting a start\_range and end\_range, calling the function, and printing the resulting list of prime numbers.