AI-ASSISTED CODING

LAB ASSISGNMENT-2.3

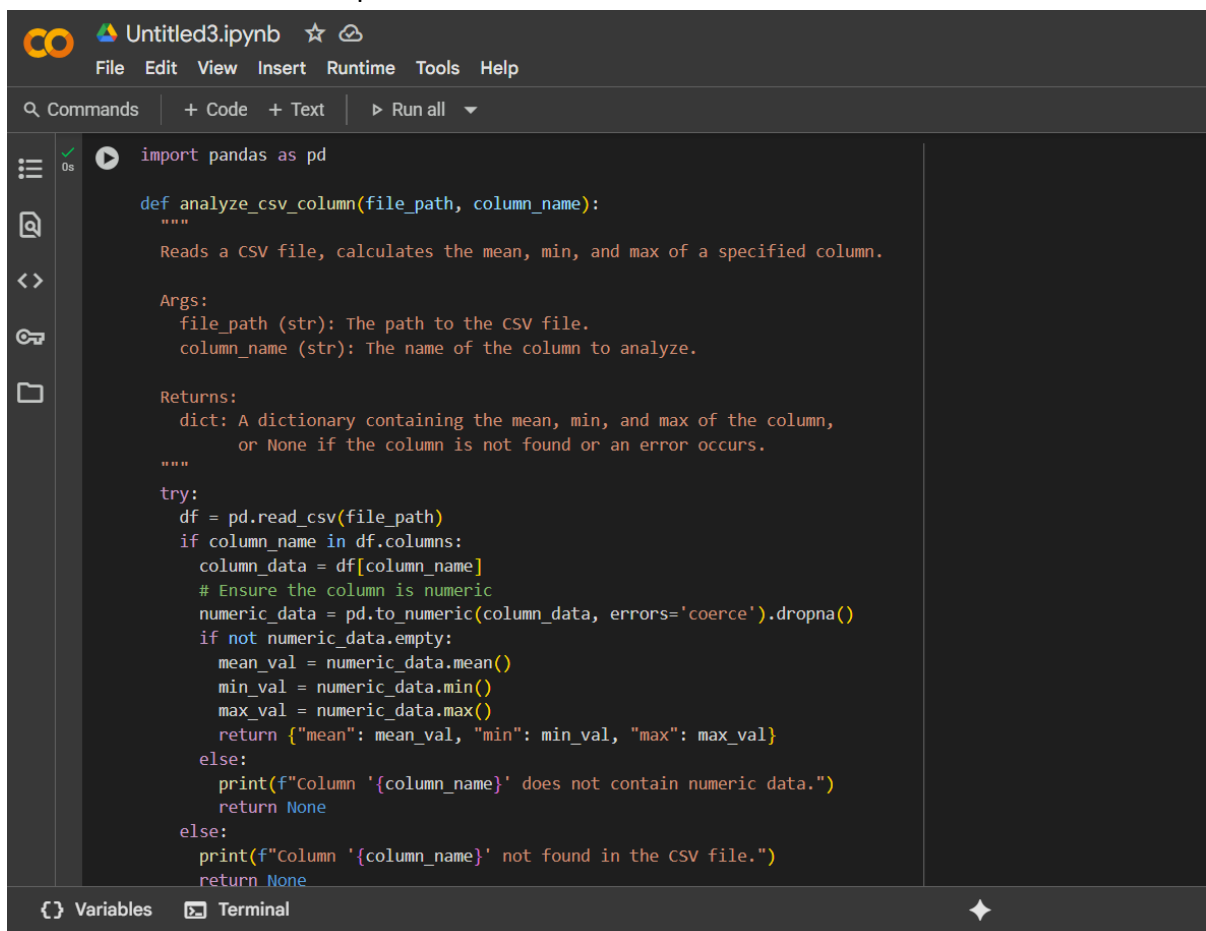Roll no:2403A510B4

Name: PANAGANTI.AJAY

Batch: 05

CSE 2nd year

Task Description#1

● Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.

Expected Output#1
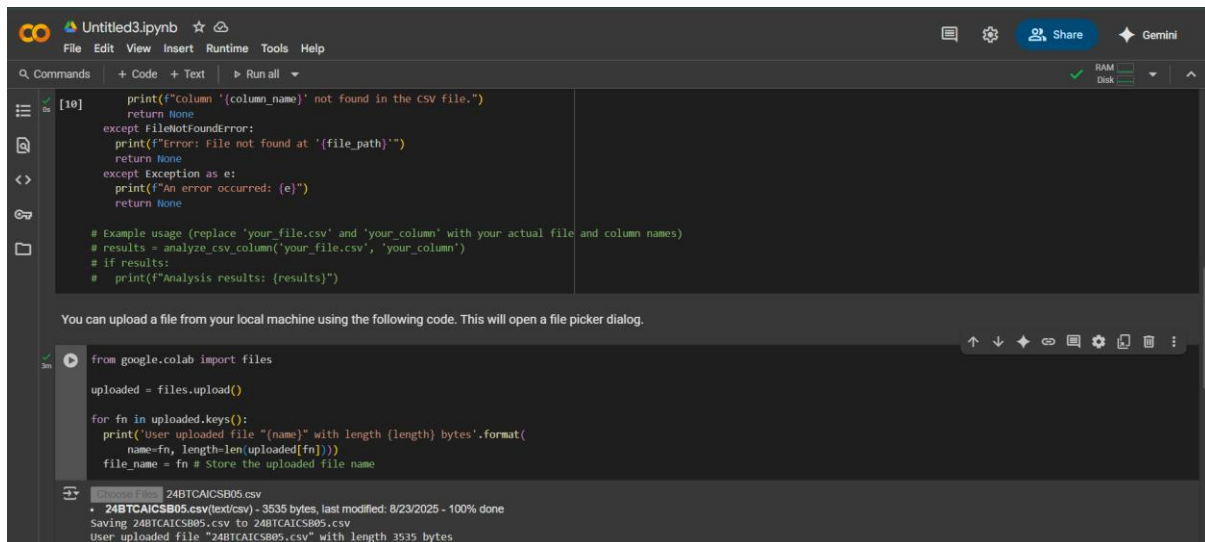
● Functional code with output and screenshot

Explanation:

Here's a breakdown of what it does:

- **import pandas as pd**: This line imports the pandas library, which is used for data manipulation and analysis, and gives it the alias pd.

- **def analyze_csv_column(file_path, column_name):**: This defines the function named analyze_csv_column that takes two arguments: file_path (the path to the CSV file) and column_name (the name of the column to analyze).
- **Docstring**: The text within triple quotes explains what the function does, its arguments, and what it returns.
- **try...except block**: This block is used for error handling. It attempts to execute the code within the try block and catches potential errors (like FileNotFoundError if the file doesn't exist or other exceptions) in the except blocks.
- **df = pd.read_csv(file_path)**: This line reads the CSV file specified by file_path into a pandas DataFrame called df.
- **if column_name in df.columns:**: This checks if the column_name provided by the user exists in the DataFrame's columns.
- **column_data = df[column_name]**: If the column exists, this line selects the data from that column.
- **numeric_data = pd.to_numeric(column_data, errors='coerce').dropna()**: This is a crucial step. It attempts to convert the data in the selected column to numeric type. errors='coerce' will turn any values that cannot be converted into numbers into NaN (Not a Number), and .dropna() removes these NaN values. This ensures that calculations are only performed on valid numbers.
- **if not numeric_data.empty:**: This checks if there is any valid numeric data left after the conversion and dropping of non-numeric values.
- **mean_val = numeric_data.mean()**, **min_val = numeric_data.min()**, **max_val = numeric_data.max()**: These lines calculate the mean, minimum, and maximum of the numeric data in the column using built-in pandas methods.
- **return {"mean": mean_val, "min": min_val, "max": max_val}**: If calculations are successful, the function returns a dictionary containing the calculated mean, min, and max values.
- **else blocks and print statements**: These handle cases where the column is not found, the column does not contain numeric data, or an error occurs during file processing, printing informative messages to the user and returning None.
- **Example Usage (commented out)**: The lines at the end show how to call the function and print the results.

In summary, this function provides a robust way to read a CSV, specifically target a column, handle non-numeric data within that column, and calculate basic descriptive statistics (mean, min, max) if valid numeric data is present.

0 / 2000

Explanation:

Here's a breakdown:

- **from google.colab import files**: This line imports the files object from the google.colab library, which provides utilities for working with files in Colab.
- **uploaded = files.upload()**: This is the main command that triggers the file upload process. When you run this line, a file picker dialog will appear in your browser, allowing you to select one or more files from your local machine. Once you've selected and confirmed the files, they are uploaded to the Colab runtime's temporary storage. The files.upload() function returns a dictionary where the keys are the filenames and the values are the file contents as bytes.
- **for fn in uploaded.keys():**: This loop iterates through the keys of the uploaded dictionary, which are the names of the files that were uploaded.
- **print('User uploaded file "{name}" with length {length} bytes'.format(name=fn, length=len(uploaded[fn])))**: Inside the loop, this line prints a confirmation message for each uploaded file, showing its name and size in bytes.
- **file_name = fn # Store the uploaded file name**: This line assigns the name of the *last* uploaded file to the variable file_name. This is useful if you only expect to upload one file and want to easily reference its name later in your code.

In short, this code provides a simple way to get files from your computer into your Colab notebook so you can work with them.
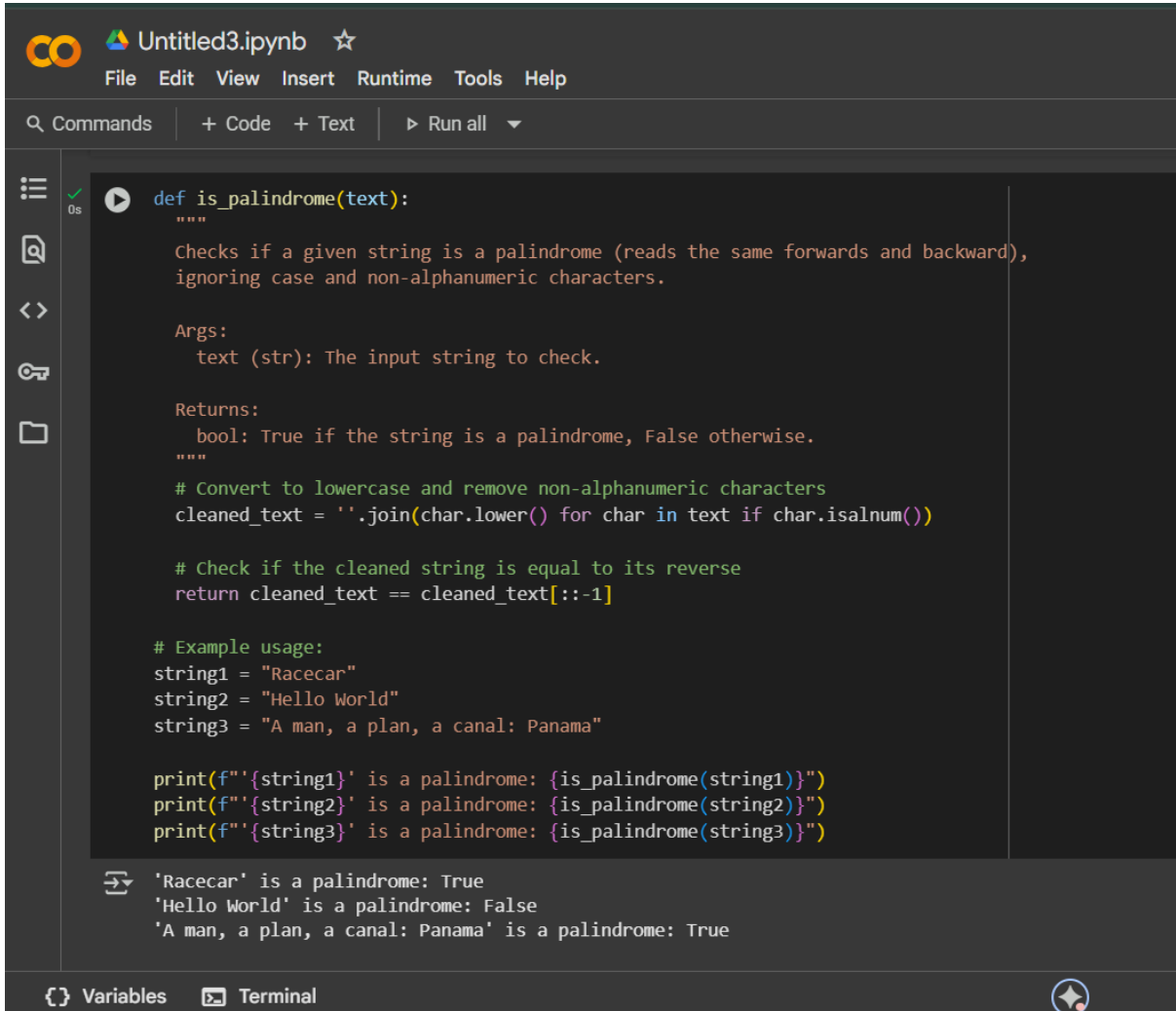
TASK DESCRIPTION#2

● Compare Gemini and Copilot outputs for a palindrome check function.

Expected Output#2

● Side-by-side comparison and observations

Gemini Code and Output:


```python
def is_palindrome(text):
    """
    Checks if a given string is a palindrome (reads the same forwards and backward),
    ignoring case and non-alphanumeric characters.

    Args:
        text (str): The input string to check.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    # Convert to lowercase and remove non-alphanumeric characters
    cleaned_text = ''.join(char.lower() for char in text if char.isalnum())

    # Check if the cleaned string is equal to its reverse
    return cleaned_text == cleaned_text[::-1]

# Example usage:
string1 = "Racecar"
string2 = "Hello World"
string3 = "A man, a plan, a canal: Panama"

print(f"'{string1}' is a palindrome: {is_palindrome(string1)}")
print(f"'{string2}' is a palindrome: {is_palindrome(string2)}")
print(f"'{string3}' is a palindrome: {is_palindrome(string3)}")
```

```
'Racecar' is a palindrome: True
'Hello World' is a palindrome: False
'A man, a plan, a canal: Panama' is a palindrome: True
```

Explanation:

This function checks if a given string is a palindrome, ignoring case and non-alphanumeric characters. Here's how it works:

- **def is_palindrome(text):**: This line defines the function named is_palindrome that takes one argument, text, which is the string you want to check.
- **Docstring**: The text within the triple quotes explains the purpose of the function, its arguments, and what it returns.
- **cleaned_text = ''.join(char.lower() for char in text if char.isalnum())**: This is the core of the cleaning process:
    - **for char in text**: It iterates through each character in the input text.
    - **if char.isalnum()**: It checks if the current character is alphanumeric (a letter or a number). If it's not alphanumeric (like spaces, punctuation, etc.), it's skipped.

- o **char.lower()**: If the character is alphanumeric, it's converted to lowercase. This makes the check case-insensitive (so "Racecar" and "racecar" are treated the same).
      - o **''.join(...)**: Finally, all the processed characters are joined back together into a new string called cleaned_text.
- **return cleaned_text == cleaned_text[::-1]**: This line checks if the cleaned_text is equal to its reverse.
      - o **cleaned_text[::-1]**: This is a Python slicing trick that creates a reversed copy of the cleaned_text string.
      - o **==**: This compares the original cleaned_text with its reversed version.
      - o **return**: The function returns True if they are the same (meaning it's a palindrome) and False otherwise.
- **Example Usage**: The lines after the function definition show how to call the function with different strings and print the results to the console.

In essence, the function cleans the input string by removing irrelevant characters and making it lowercase, and then it simply checks if the cleaned string reads the same forwards and backward.

Copilot Code:

```python
def is_palindrome(s):
    # Remove spaces and convert to lowercase for uniformity
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned == cleaned[::-1]

# Example usage
test_strings = ["Madam", "Racecar", "Hello", "12321", "Was it a car or a cat

for text in test_strings:
    result = is_palindrome(text)
    print(f"'{text}' is a palindrome? → {result}")
```

```python
alindrome(s):
    nove spaces and convert to lowercase for uniformity
    ned = ''.join(c.lower() for c in s if c.isalnum())
    :n cleaned == cleaned[::-1]

: usage
ings = ["Madam", "Racecar", "Hello", "12321", "Was it a car or a cat I saw"]

 in test_strings:
    .t = is_palindrome(text)
    :(f"'{text}' is a palindrome? → {result}")
```

Copilot Output:

```
'Madam' is a palindrome? → True
'Racecar' is a palindrome? → True
'Hello' is a palindrome? → False
'12321' is a palindrome? → True
'Was it a car or a cat I saw' is a palindrome? → True
```

Explanation:

⬜ c.lower(): Converts each character to lowercase so that "Madam" and "madam" are treated the same.

⬜ if c.isalnum(): Keeps only letters and numbers, ignoring spaces, punctuation, etc.

⬜ ''.join(...): Combines the cleaned characters back into a single string.

TASK DESCRIPTION#3

● Ask Gemini to explain a Python function (to calculate area of various shapes) line by line..

Expected Output#3

● Detailed explanation with code snippet

Code and Output:

```python
import math

def calculate_area(shape, **kwargs):
    """
    Calculates the area of various shapes.

    Args:
      shape (str): The name of the shape (e.g., 'circle', 'rectangle', 'triangle').
      **kwargs: Keyword arguments for the dimensions of the shape.

    Returns:
      float: The area of the shape, or None if the shape is not supported or dimensions are missing.
    """
    if shape.lower() == 'circle':
      radius = kwargs.get('radius')
      if radius is not None:
        return math.pi * (radius ** 2)
      else:
        print("Error: Radius is required for a circle.")
        return None
    elif shape.lower() == 'rectangle':
      length = kwargs.get('length')
      width = kwargs.get('width')
      if length is not None and width is not None:
        return length * width
      else:
        print("Error: Length and width are required for a rectangle.")
        return None
    elif shape.lower() == 'triangle':
      base = kwargs.get('base')
```

```python
        print("Error: Length and width are required for a rectangle.")
        return None
    elif shape.lower() == 'triangle':
      base = kwargs.get('base')
      height = kwargs.get('height')
      if base is not None and height is not None:
        return 0.5 * base * height
      else:
        print("Error: Base and height are required for a triangle.")
        return None
    else:
      print(f"Error: Shape '{shape}' is not supported.")
      return None

# Example usage:
circle_area = calculate_area('circle', radius=5)
rectangle_area = calculate_area('rectangle', length=4, width=6)
triangle_area = calculate_area('triangle', base=3, height=7)

print(f"Area of circle with radius 5: {circle_area}")
print(f"Area of rectangle with length 4 and width 6: {rectangle_area}")
print(f"Area of triangle with base 3 and height 7: {triangle_area}")
```

```
Area of circle with radius 5: 78.53981633974483
Area of rectangle with length 4 and width 6: 24
Area of triangle with base 3 and height 7: 10.5
```

Explanation:

Here is a line-by-line explanation of the calculate_area function:

- **import math**: This line imports the math module, which is necessary to use math.pi for circle area calculations.
- **def calculate_area(shape, **kwargs):**: This defines the function named calculate_area.

- o   shape is a string argument specifying the type of shape (e.g., 'circle', 'rectangle').
- o   **kwargs allows you to pass variable keyword arguments (like radius=5, length=4, width=6, etc.) for the dimensions.
- **Docstring**: The text in triple quotes explains the function's purpose, arguments, and return value.
- **if shape.lower() == 'circle':**: This checks if the shape is 'circle' (case-insensitive).
- **radius = kwargs.get('radius')**: It safely retrieves the 'radius' value from the keyword arguments.
- **if radius is not None:**: Checks if a radius was provided.
- **return math.pi * (radius ** 2)**: Calculates and returns the circle's area.
- **else:**: If no radius was given for a circle:
- **print("Error: Radius is required for a circle.")**: Prints an error message.
- **return None**: Returns None.
- **elif shape.lower() == 'rectangle':**: Checks if the shape is 'rectangle'.
- **length = kwargs.get('length')** and **width = kwargs.get('width')**: Safely retrieve 'length' and 'width'.
- **if length is not None and width is not None:**: Checks if both length and width were provided.
- **return length * width**: Calculates and returns the rectangle's area.
- **else:**: If length or width is missing:
- **print("Error: Length and width are required for a rectangle.")**: Prints an error.
- **return None**: Returns None.
- **elif shape.lower() == 'triangle':**: Checks if the shape is 'triangle'.
- **base = kwargs.get('base')** and **height = kwargs.get('height')**: Safely retrieve 'base' and 'height'.
- **if base is not None and height is not None:**: Checks if both base and height were provided.
- **return 0.5 * base * height**: Calculates and returns the triangle's area.
- **else:**: If base or height is missing:
- **print("Error: Base and height are required for a triangle.")**: Prints an error.
- **return None**: Returns None.
- **else:**: If the shape is not supported:
- **print(f"Error: Shape '{shape}' is not supported.")**: Prints an error message with the unsupported shape name.
- **return None**: Returns None.
- **Example Usage**: The lines outside the function show how to call the function with different shapes and dimensions and print the results.

TASK DESCRIPTION#4

● Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares)

Expected Output#4

● Screenshots of working environments with few prompts to generate python code

Code:

TASK DESCRIPTION#4

● Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares)

RUN

Run and Debug

To customize Run and Debug create a launch.json file.

Show automatic Python configurations

sum_of_squares.py > ⊗ sum_of_squares_formula

```python
def sum_of_squares_builtin(numbers):
    """
    Calculate sum of squares using built-in sum() and list comprehension.

    Args:
        numbers (list): List of numbers

    Returns:
        float: Sum of squares of all numbers
    """
    return sum(x**2 for x in numbers)

def sum_of_squares_loop(numbers):
    """
    Calculate sum of squares using a traditional for loop.

    Args:
        numbers (list): List of numbers

    Returns:
        float: Sum of squares of all numbers
    """
    total = 0
    for num in numbers:
        total += num ** 2
    return total

def sum_of_squares_map(numbers):
    """
    Calculate sum of squares using map() function.

    Args:
        numbers (list): List of numbers
```

sum_of_squares.py > ⊗ sum_of_squares_range

```python
def sum_of_squares_map(numbers):
    """
    Calculate sum of squares using map() function.

    Args:
        numbers (list): List of numbers

    Returns:
        float: Sum of squares of all numbers
    """
    return sum(map(lambda x: x**2, numbers))

def sum_of_squares_range(n):
    """
    Calculate sum of squares from 1 to n.

    Args:
        n (int): Upper limit

    Returns:
        int: Sum of squares from 1² to n²
    """
    return sum(i**2 for i in range(1, n + 1))

def sum_of_squares_formula(n):
    """    Calculate sum of squares from 1 to n using mathematical formula. TAB to jump here
    Formula: n(n+1)(2n+1)/6

    Args:
        n (int): Upper limit

    Returns:
```
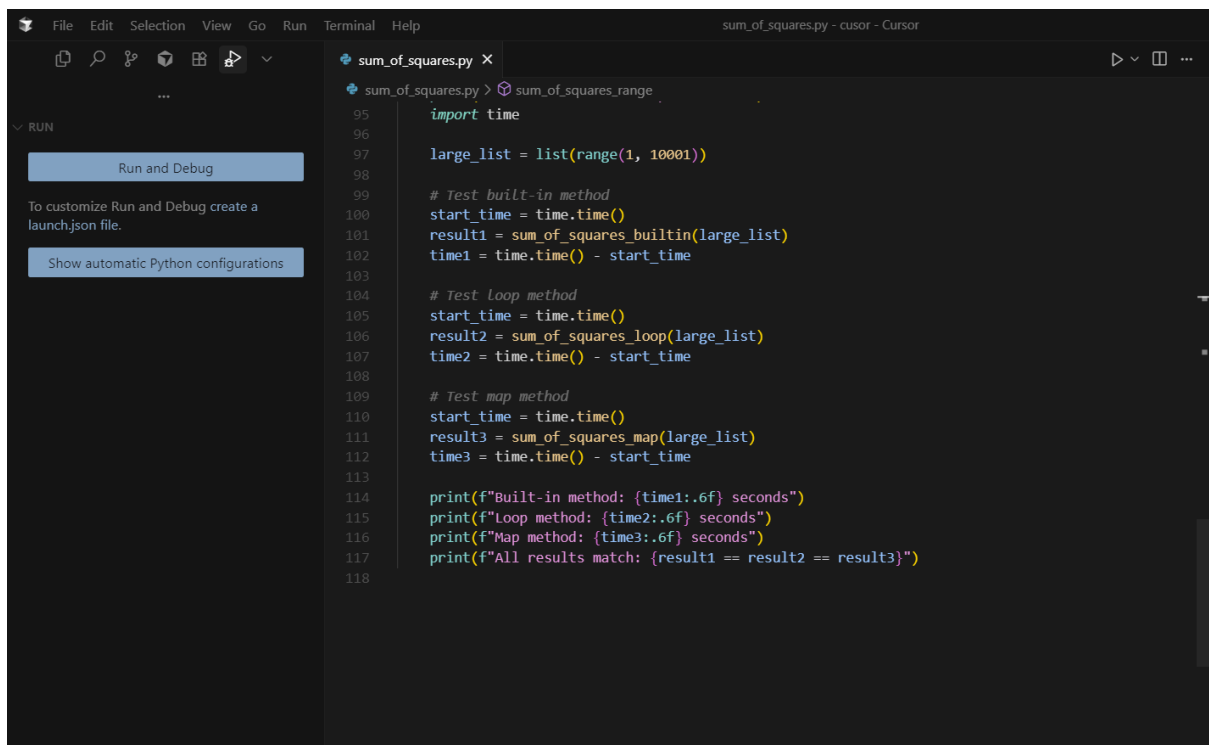
```python
        return sum(i**2 for i in range(1, n + 1))

def sum_of_squares_formula(n):
    """    Calculate sum of squares from 1 to n using mathematical formula.
    Formula: n(n+1)(2n+1)/6

    Args:
        n (int): Upper limit

    Returns:
        int: Sum of squares from 1² to n²
    """
    return n * (n + 1) * (2 * n + 1) // 6

# Example usage and testing
if __name__ == "__main__":
    # Test with a list of numbers
    test_numbers = [1, 2, 3, 4, 5]
    print("Test numbers:", test_numbers)

    print("\n--- Different Methods ---")
    print(f"Using built-in sum(): {sum_of_squares_builtin(test_numbers)}")
    print(f"Using for loop: {sum_of_squares_loop(test_numbers)}")
    print(f"Using map(): {sum_of_squares_map(test_numbers)}")

    # Test with range
    n = 5
    print(f"\n--- Sum of squares from 1 to {n} ---")
    print(f"Using range: {sum_of_squares_range(n)}")
    print(f"Using formula: {sum_of_squares_formula(n)}")

    # Interactive example
    print("\n--- Interactive Example ---")
```
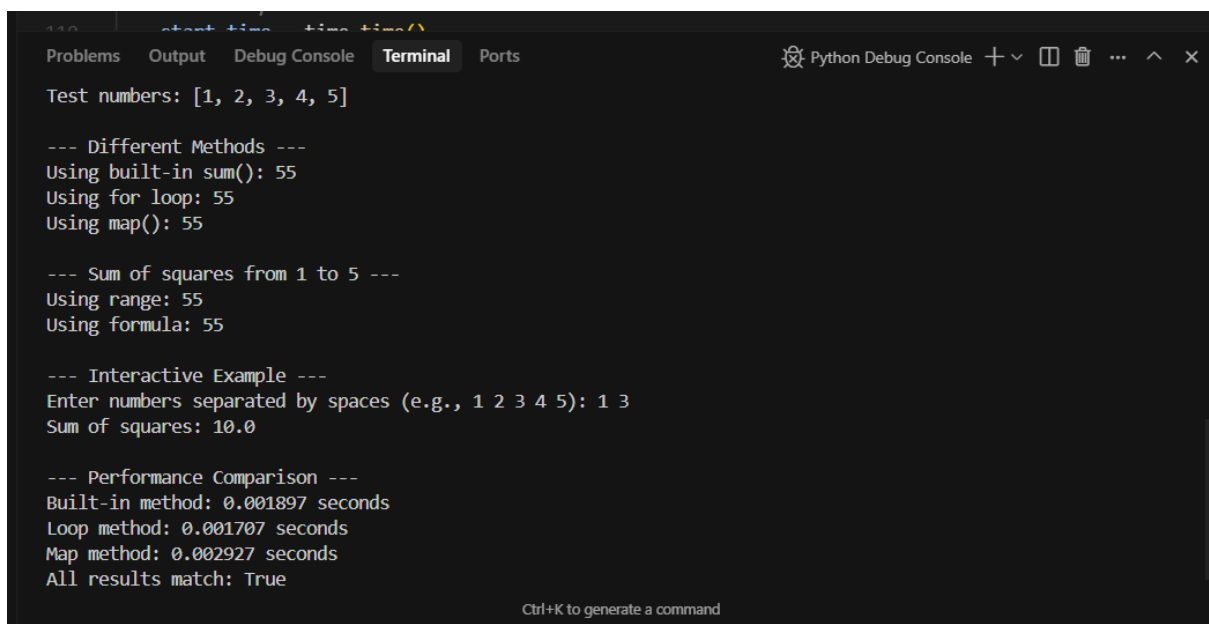
```python
    # Interactive example
    print("\n--- Interactive Example ---")
    try:
        user_input = input("Enter numbers separated by spaces (e.g., 1 2 3 4 5): ")
        user_numbers = [float(x) for x in user_input.split()]
        result = sum_of_squares_builtin(user_numbers)
        print(f"Sum of squares: {result}")
    except ValueError:
        print("Invalid input. Please enter valid numbers separated by spaces.")
    except KeyboardInterrupt:
        print("\nProgram interrupted by user.")

    # Performance comparison
    print("\n--- Performance Comparison ---")
    import time

    large_list = list(range(1, 10001))

    # Test built-in method
    start_time = time.time()
    result1 = sum_of_squares_builtin(large_list)
    time1 = time.time() - start_time

    # Test loop method
    start_time = time.time()
    result2 = sum_of_squares_loop(large_list)
    time2 = time.time() - start_time

    # Test map method
    start_time = time.time()
    result3 = sum_of_squares_map(large_list)
    time3 = time.time() - start_time
```
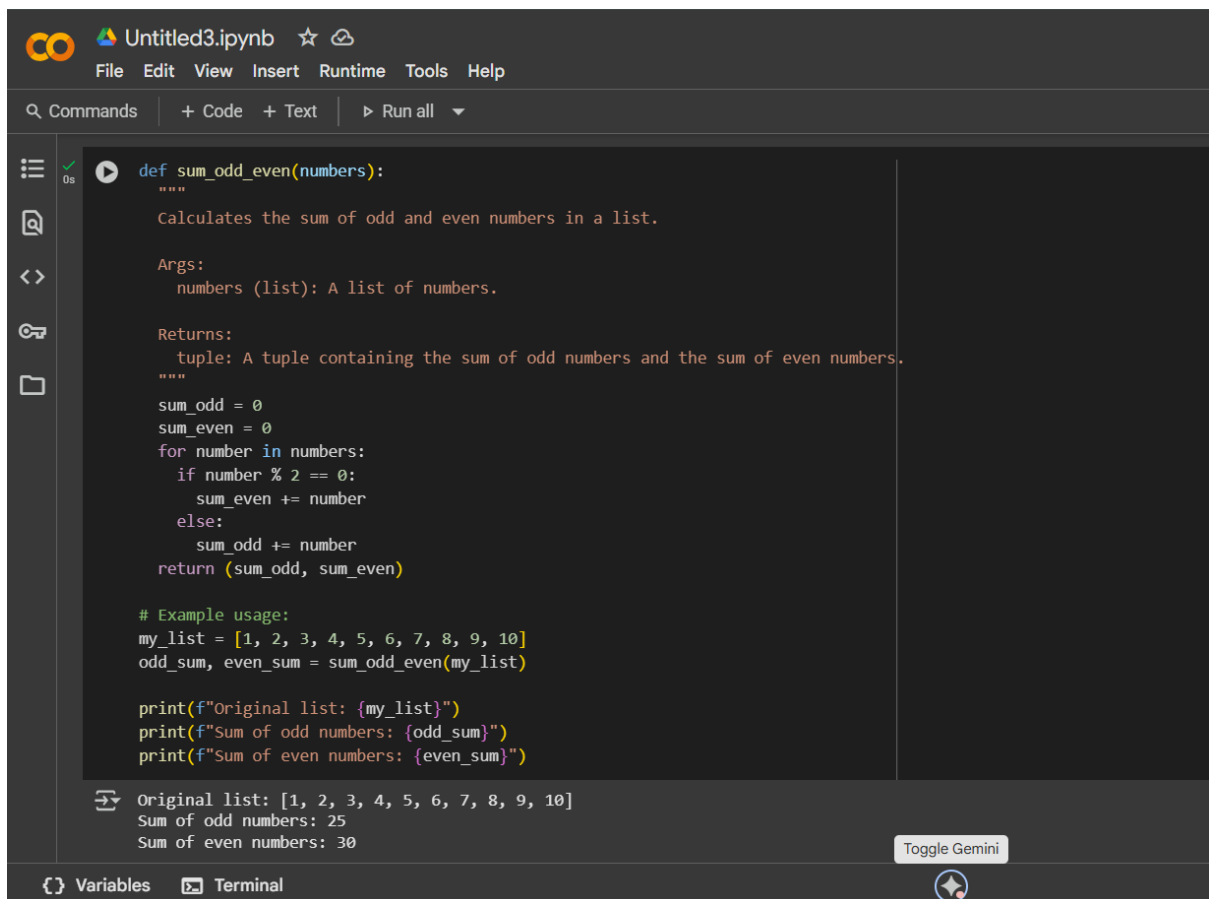
Output:

TASK DESCRIPTION#5

● Student need to write code to calculate sum of add number and even numbers in the list

Expected Output#5

● Refactored code written by student with improved logic.

Code and Output:

```python
def sum_odd_even(numbers):
    """
    Calculates the sum of odd and even numbers in a list.

    Args:
        numbers (list): A list of numbers.

    Returns:
        tuple: A tuple containing the sum of odd numbers and the sum of even numbers.
    """
    sum_odd = 0
    sum_even = 0
    for number in numbers:
        if number % 2 == 0:
            sum_even += number
        else:
            sum_odd += number
    return (sum_odd, sum_even)

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd_sum, even_sum = sum_odd_even(my_list)

print(f"Original list: {my_list}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
```

Explanation:

Here's a line-by-line explanation:

- **def sum_odd_even(numbers):**: This line defines the function named sum_odd_even that takes one argument, numbers, which is expected to be a list of numbers.

- **Docstring**: The text within triple quotes explains what the function does, its arguments, and what it returns.

- **sum_odd = 0**: This initializes a variable sum_odd to zero. This variable will store the cumulative sum of the odd numbers found in the list.

- **sum_even = 0**: This initializes a variable sum_even to zero. This variable will store the cumulative sum of the even numbers found in the list.

- **for number in numbers:**: This starts a loop that will iterate through each element in the input list numbers. In each iteration, the current element is assigned to the variable number.

- **if number % 2 == 0:**: Inside the loop, this line checks if the current number is even. The modulo operator (%) calculates the remainder when number is divided by 2. If the remainder is 0, the number is even.

- **sum_even += number**: If the condition in the if statement is true (the number is even), this line adds the current number to the sum_even variable. The += is a shorthand for sum_even = sum_even + number.
- **else:**: If the condition in the if statement is false (the number is not even, meaning it's odd):
- **sum_odd += number**: This line adds the current number to the sum_odd variable.
- **return (sum_odd, sum_even)**: After the loop has finished iterating through all the numbers in the list, this line returns a tuple containing two values: the final sum_odd and the final sum_even.
- **Example Usage**: The lines following the function definition demonstrate how to create a list (my_list), call the sum_odd_even function with this list, and then print the original list and the calculated sums of odd and even numbers.

In essence, the function goes through the list number by number, checks if each number is odd or even, and adds it to the corresponding sum variable. Finally, it provides the total sums for both odd and even numbers.