

AI ASSISTED CODING:

ASSIGNMENT-2.3:

M.KEERTHANA

2403A51259

BATCH-11

CSE-GEN

Task-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.

Prompt: Write a Python function in Google Colab that reads a CSV file into a pandas Data Frame and calculates the mean, minimum, and maximum of numeric columns. Include example CSV input and display the output

Code:

```
import csv

# Step 1: Create a sample CSV file inside Colab
with open("data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["Name", "Marks"])
    writer.writerow(["Alice", 85])
    writer.writerow(["Bob", 90])
    writer.writerow(["Charlie", 78])
    writer.writerow(["David", 92])
    writer.writerow(["Eva", 88])

# Step 2: Function to calculate mean, min, max
def calculate_stats(file_path, column_name):
    values = []
    with open(file_path, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            try:
                values.append(float(row[column_name]))
            except ValueError:
                pass

    if not values:
        return "No numeric data found in the given column."
    mean_val = sum(values) / len(values)
    min_val = min(values)
    max_val = max(values)
    return {"mean": mean_val, "min": min_val, "max": max_val}

# Step 3: Run and print results
file_path = "data.csv"
column_name = "Marks"
stats = calculate_stats(file_path, column_name)
print("Mean:", stats["mean"])
print("Min:", stats["min"])
print("Max:", stats["max"])
```

Output:



Mean: 86.6
Min: 78.0
Max: 92.0

Task-2:

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

Expected Output-2:

- Side-by-side comparison and observations.

Prompt: Write a Python function that checks whether a given string is a palindrome. Include comments and example usage with input and output.

CODE:

```
# --- Gemini's palindrome check ---
def is_palindrome(s):
    # Normalize: lowercase + remove non-alphanumeric
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]

# --- Copilot's palindrome check ---
def check_palindrome(word):
    # Simple direct check without preprocessing
    return word == word[::-1]

# --- Test strings ---
test_strings = [
    "racecar",           # Perfect palindrome
    "RaceCar",          # Different case
    "A man, a plan, a canal: Panama", # Spaces & punctuation
    "hello"              # Not a palindrome
]

# --- Side-by-side console output ---
print(f"{'Input':40} | {'Gemini Output':15} | {'Copilot Output'}")
```

```
# --- Side-by-side console output ---
print(f"{'Input':40} | {'Gemini Output':15} | {'Copilot Output'}")
print("-" * 75)
for text in test_strings:
    gemini_result = is_palindrome(text)
    copilot_result = check_palindrome(text)
    print(f"{text:40} | {gemini_result!s:15} | {copilot_result}")

# --- Optional: Pretty table with Pandas ---
import pandas as pd

data = []
for text in test_strings:
    data.append({
        "Input": text,
        "Gemini Output": is_palindrome(text),
        "Copilot Output": check_palindrome(text)
    })

df = pd.DataFrame(data)
df
```

Output:

| | Input | Gemini Output | Copilot Output |
|--|--------------------------------|---------------|----------------|
| | racecar | True | True |
| | RaceCar | True | False |
| | A man, a plan, a canal: Panama | True | False |
| | hello | False | False |

| | Input | Gemini Output | Copilot Output |
|---|--------------------------------|---------------|----------------|
| 0 | racecar | True | True |
| 1 | RaceCar | True | False |
| 2 | A man, a plan, a canal: Panama | True | False |
| 3 | hello | False | False |

Side-by-side comparison:

Gemini Output

```
python
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]
```

Copilot Output

```
python
def is_palindrome(s):
    s = s.replace(" ", "").lower()
    return s == s[::-1]
```

Obsesrvation:

Both Gemini and Copilot produced identical solutions in terms of logic, structure, and output. The only minor difference is the order of

.lower() and .replace(), but it doesn't change the result. This shows that for simple problems like palindrome checks, AI tools tend to converge on the same optimal approach.

Task-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

Prompt: Explain the following Python function line by line. The function calculates the area of various shapes (circle, rectangle, triangle). Provide a detailed explanation with the code snippet.

```
def calculate_area(shape, **kwargs):
    # shape: string specifying the type of shape ("circle", "rectangle", "triangle")
    # **kwargs: extra keyword arguments for the required dimensions

    if shape == "circle":
        # Get the radius from kwargs, default to 0 if not provided
        radius = kwargs.get("radius", 0)
        # Formula for area of a circle:  $\pi \times \text{radius}^2$ 
        return 3.1416 * radius ** 2

    elif shape == "rectangle":
        # Get the length and width, defaulting to 0 if missing
        length = kwargs.get("length", 0)
        width = kwargs.get("width", 0)
        # Formula for area of a rectangle: length  $\times$  width
        return length * width

    elif shape == "triangle":
        # Get the base and height, defaulting to 0 if missing
        base = kwargs.get("base", 0)
        height = kwargs.get("height", 0)
        # Formula for area of a triangle:  $0.5 \times \text{base} \times \text{height}$ 
        return 0.5 * base * height

    else:
        # If shape is not recognized, return an error message
        return "Shape not supported"

# Example usage:
print(calculate_area("circle", radius=5))      # Circle with radius 5
print(calculate_area("rectangle", length=4, width=6))  # Rectangle 4x6
print(calculate_area("triangle", base=3, height=8))   # Triangle base 3, height 8
print(calculate_area("hexagon", side=5))           # Unsupported shape
```

Output:

```
➡ 78.53999999999999  
24  
12.0  
Shape not supported
```

Explanation:

1. How it works:

The function takes the name of the shape (circle, rectangle, or triangle). It also takes the measurements needed for that shape (like radius, length, width, base, height) using ****kwargs**. Kwargs allows the function to accept many extra values without fixing them in advance.

2. For a Circle:

Gets the radius.

Formula: $\text{Area} = 3.1416 \times \text{radius}^2$.

3. For a Rectangle:

Gets length and width.

Formula: $\text{Area} = \text{length} \times \text{width}$.

4. For a Triangle:

Gets base and height.

Formula: $\text{Area} = 0.5 \times \text{base} \times \text{height}$.

5. If shape is not found:

It returns "Shape not supported".

6. Example:

```
print (calculate_area ("circle", radius=5))  
print (calculate_area ("rectangle", length=4, width=6))  
print (calculate_area ("triangle", base=3, height=8))  
print (calculate_area ("hexagon", side=5))
```

7. Output:

Circle (radius 5) → 78.54

Rectangle (4×6) → 24

Triangle (base 3, height 8) → 12.0

Hexagon → Shape not supported

Task-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

Prompt:

Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares). Provide screenshots of working environment, prompts, and generated Python code.

Code:

```
from typing import Iterable, Union

Number = Union[int, float]

def sum_of_squares(numbers: Iterable[Number]) -> float:
    """Return the sum of squares of the provided numbers.

    Args:
        numbers: Any iterable of numeric values (ints or floats).

    Returns:
        The sum of each number squared, as a float.
    """
    return float(sum(value * value for value in numbers))


if __name__ == "__main__":
    example_numbers: list[Number] = [1, 2, 3, 4]
    result = sum_of_squares(example_numbers)
    print(f"Input: {example_numbers}")
    print(f"Sum of squares: {result}")
```

Output:

```
1.exe c:/Users/sonti/sum.py
Input: [1, 2, 3, 4]
Sum of squares: 30.0
PS C:\Users\sonti>
```


Task-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.

Prompt: Write a Python function that takes a list of integers and calculates the sum of odd numbers and even numbers separately. Return both results.


Before refactored code:

```
 # Initial code (before refactoring)
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

sum_even = 0
sum_odd = 0

for num in numbers:
    if num % 2 == 0:
        sum_even += num
    else:
        sum_odd += num

print("Sum of even numbers:", sum_even)
print("Sum of odd numbers:", sum_odd)
```

```
 Sum of even numbers: 30
Sum of odd numbers: 25
```

Refactored code:



Refactored code

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum_even = sum(n for n in numbers if n % 2 == 0)
sum_odd = sum(n for n in numbers if n % 2 != 0)
print(f"Sum of even numbers: {sum_even}")
print(f"Sum of odd numbers: {sum_odd}")
```



```
Sum of even numbers: 30
Sum of odd numbers: 25
```