

## LAB ASSIGNMENT – 2

**Name:**Kappaganti Srimukhi

**Hall Ticket No:** 2303A54060

**Batch :** 47\_B

### Task 1: Statistical Summary for Survey Data


#### Prompt:

generate a python function that takes a list of numbers and calculates the mean, minimum and maximum values. Display the output

+

Gemini 2.5 Flash ▾ ▶

#### Code & Output:


```
[2]
✓ Os  # Example usage:
my_list = [10, 20, 5, 30, 15, 25]
mean, minimum, maximum = analyze_numbers(my_list)

print(f"Original List: {my_list}")
print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")

# Example with an empty list
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nOriginal List (empty): {empty_list}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum: {max_empty}")

... Original List: [10, 20, 5, 30, 15, 25]
Mean: 17.5
Minimum: 5
Maximum: 30

Original List (empty): []
Mean: None
Minimum: None
Maximum: None
```

```
[2]
✓ Os  # Example usage:
my_list = [10, 20, 5, 30, 15, 25]
mean, minimum, maximum = analyze_numbers(my_list)

print(f"Original List: {my_list}")
print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")

# Example with an empty list
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nOriginal List (empty): {empty_list}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum: {max_empty}")
```

---

```
... Original List: [10, 20, 5, 30, 15, 25]
Mean: 17.5
Minimum: 5
Maximum: 30

Original List (empty): []
Mean: None
Minimum: None
Maximum: None
```

### **Explanation:**

- Gemini produced readable and clear code.
- made effective use of built-in Python functions
- The output appeared correctly in Google Colab.

## **Task 2: Armstrong Number – AI Comparison**

### **Prompt:**

Write a python program to check whether a number is Armstrong number



Gemini 2.5 Flash ▾ 

### **Code:**

[3]  
✓ Os

```
def is_armstrong_number(number):  
    """  
    Checks if a number is an Armstrong number.  
  
    An Armstrong number is a number that is equal to the sum of its own digits  
    each raised to the power of the number of digits.  
  
    Args:  
        number (int): The integer to check.  
  
    Returns:  
        bool: True if the number is an Armstrong number, False otherwise.  
    """  
    if not isinstance(number, int) or number < 0:  
        print("Please enter a non-negative integer.")  
        return False  
  
    num_str = str(number)  
    num_digits = len(num_str)  
    sum_of_powers = 0  
  
    for digit_char in num_str:  
        digit = int(digit_char)  
        sum_of_powers += digit ** num_digits  
  
    return sum_of_powers == number
```

[4]  
✓ Os


```
# Example Usage:  
  
# Test with an Armstrong number  
num1 = 153  
if is_armstrong_number(num1):  
    print(f"{num1} is an Armstrong number.")  
else:  
    print(f"{num1} is not an Armstrong number.")  
  
# Test with a non-Armstrong number  
num2 = 123  
if is_armstrong_number(num2):  
    print(f"{num2} is an Armstrong number.")  
else:  
    print(f"{num2} is not an Armstrong number.")  
  
# Test with another Armstrong number  
num3 = 407  
if is_armstrong_number(num3):  
    print(f"{num3} is an Armstrong number.")  
else:  
    print(f"{num3} is not an Armstrong number.")
```

### Output:

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

### Using GitHub Copilot in VS Code:

Check whether a number is Armstrong number

 Add Context...

#### **Code:**

```
1 def is_armstrong(n):
2     """Check if a number is an Armstrong number."""
3     digits = str(abs(n))
4     num_digits = len(digits)
5     sum_of_powers = sum(int(digit) ** num_digits for digit in digits)
6     return sum_of_powers == abs(n)
7
8
9 # Test cases
10 print(is_armstrong(153))    # True (1³ + 5³ + 3³ = 153)
11 print(is_armstrong(9474))  # True (9⁴ + 4⁴ + 7⁴ + 4⁴ = 9474)
12 print(is_armstrong(123))   # False
```

#### **Output:**

```
PS C:\Users\HP\OneDrive\Documents\Website-1> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/Users/HP/OneDrive/Documents/Website-1/Armstrong.py
True
True
False
```

#### **Explanation:**

- Gemini is better at learning the basics.
- Copilot generates code that is more polished and optimised.

### **Task 3: Leap Year Validation Using Cursor AI**

#### **Prompt 1:**

Write a python program to check whether a given year is leap year

Auto ▾

Edit Selection ▾



#### **Code:**

```

year = int(input("Enter a year: "))

if (year % 4 == 0):
    if (year % 100 == 0):
        if (year % 400 == 0):
            print(f"{year} is a leap year.")
        else:
            print(f"{year} is not a leap year.")
    else:
        print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")

```

### Output:

```

PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python313
Enter a year: 2024
2024 is a leap year.

```

### Prompt 2: Optimised version

```

def is_leap_year(year):
    """
    Returns True if the given year is a leap year, else False.
    """
    return (year % 4 == 0 and (year % 100 != 0 or year % 400 == 0))

# Example usage
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")

```

### Explanation:

- Version 2 is reusable and modular.
- Cursor AI enhanced structure and documentation
- The second prompt produced code that was more polished and tidy.

### Task 4: Student Logic + AI Refactoring (Odd/Even Sum)

#### **Own Code:**

```

1 numbers = (1, 2, 3, 4, 5, 6)
2 even_sum = 0
3 odd_sum = 0
4 for num in numbers:
5     if num % 2 == 0:
6         even_sum += num
7     else:
8         odd_sum += num
9 print("Even Sum:", even_sum)
10 print("Odd Sum:", odd_sum)

```

#### Output:

```
>>> %Run -c $EDITOR_CONTENT
```

```
Even Sum: 12
```

```
Odd Sum: 9
```

#### Refactored Code:

- B** numbers = (1, 2, 3, 4, 5, 6) even\_sum = 0 odd\_sum = 0 for num in numbers: if num % 2 == 0: even\_sum += num else: odd\_sum += num print("Even Sum:", even\_sum) print("Odd Sum:", odd\_sum) Refactor this code to make it more optimized and readable.

#### Code & Output:

◆ Gemini

```

▶ numbers = (1, 2, 3, 4, 5, 6)

# Refactored code using sum() with generator expressions
even_sum = sum(num for num in numbers if num % 2 == 0)
odd_sum = sum(num for num in numbers if num % 2 != 0)

print("Even Sum:", even_sum)
print("Odd Sum:", odd_sum)

```

```

... Even Sum: 12
    Odd Sum: 9

```

#### Explanation:

- In refactored code, the number of lines are reduced

- And such codes are very useful for large data sets.