Assignment 2.1

2303A510A1 - Bhanu

Batch - 14

Task 1: Statistical Summary for Survey Data

Prompt:

Write a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values.

Code & Output:

```
def calculate_stats(numbers):
    if not numbers:
        return None, None, None

    mean_val = sum(numbers) / len(numbers)
    min_val = min(numbers)
    max_val = max(numbers)

    return mean_val, min_val, max_val

# Example usage:
my_list = list(map(int, input("Enter a list of numbers: ").split()))
mean, minimum, maximum = calculate_stats(my_list)

print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")
```

```
Enter a list of numbers: 2 7 3 8 4 9
Mean: 5.5
Minimum: 2
Maximum: 9
```

Explanation:

- The code is a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values.
- The function first checks if the list is empty and returns None, None, None if it is.
- Then it calculates the mean by summing up all the numbers and dividing by the number of numbers.
- Then it calculates the minimum by finding the smallest number in the list.

- Then it calculates the maximum by finding the largest number in the list.
- Finally it returns the mean, minimum, and maximum values.

Justification:

Survey data analysis is a common real-world task in data analytics. Calculating mean, minimum, and maximum values helps summarize user responses and identify trends. Using Google Gemini in Colab demonstrates how AI can quickly generate accurate statistical functions.

Task 2: Armstrong Number – AI Comparison

Prompt:

Write a Python program to check whether a given number is an Armstrong number. Explain the logic clearly and keep the code simple.

Code & Output by Gemini:

```python
def is_armstrong_number(number):
    # Convert the number to a string to easily count digits and iterate through them
    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

n=int(input("Enter the number: "))
print(is_armstrong_number(n))
```

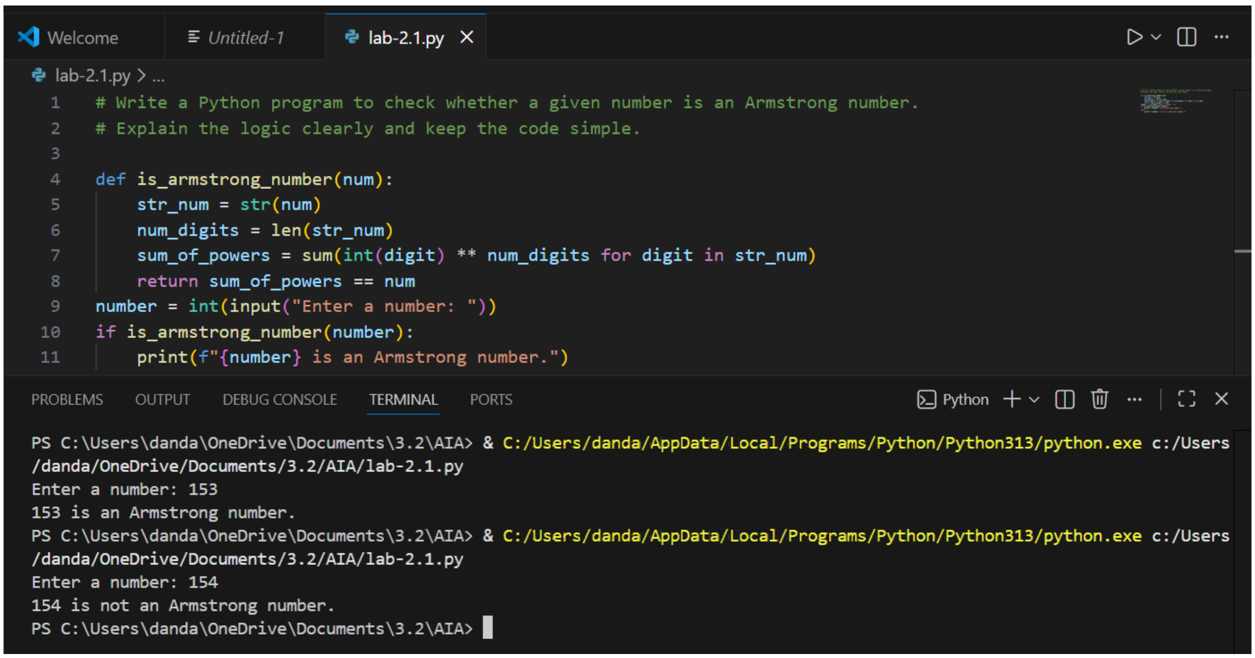
```
Enter the number: 153
True
```

Explanation:

- **Count the number of digits:** Determine how many digits are in the given number.
- **Initialize sum:** Create a variable to store the sum of the powered digits, starting at zero.

- **Extract digits and calculate power:** Iterate through each digit of the number. For each digit, raise it to the power of the total number of digits and add it to the sum.
- **Compare:** After processing all digits, compare the calculated sum with the original number. If they are equal, the number is an Armstrong number.

Code & Output by Github:



Explanation:

- Convert the number to a string to easily iterate over each digit.
- Calculate the number of digits in the number.
- Use a generator expression to compute the sum of each digit raised to the power of the number of digits.
- Compare the computed sum with the original number to determine if it is an Armstrong number.
- Print the result accordingly.

Comparison between Gemini & Github:

- Google Gemini produces a concise and Pythonic solution using string conversion and built-in functions.

- GitHub Copilot uses a loop-based, step-by-step approach that clearly shows the Armstrong number logic.
- Both implementations are correct, but they differ in readability and coding style.

Justification:

This comparison highlights how different AI tools solve the same problem using varied logic and coding styles. It helps evaluate code readability, clarity, and learning value across AI platforms.

Task 3: Leap Year Validation Using Cursor AI

Prompt:

Write a Python program to check if a year is a leap year

Code & Output:

```python
# Write a Python program to check if a year is a leap year
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year")
else:
    print(f"{year} is not a leap year")
```

Problems Output Debug Console Terminal Ports powershell

```
PS C:\Users\danda\OneDrive\Documents\3.2\AIA> & C:\Users\danda\AppData\Local\Programs\Python\Python313\python.exe c:/Users/danda/OneDrive/Documents/3.2/AIA/lab-2.1.py
Enter a year: 2000
2000 is a leap year
PS C:\Users\danda\OneDrive\Documents\3.2\AIA> & C:\Users\danda\AppData\Local\Programs\Python\Python313\python.exe c:/Users/danda/OneDrive/Documents/3.2/AIA/lab-2.1.py
Enter a year: 2012
2012 is a leap year
PS C:\Users\danda\OneDrive\Documents\3.2\AIA> & C:\Users\danda\AppData\Local\Programs\Python\Python313\python.exe c:/Users/danda/OneDrive/Documents/3.2/AIA/lab-2.1.py
Enter a year: 2018
2018 is not a leap year
PS C:\Users\danda\OneDrive\Documents\3.2\AIA>
```

Explanation:

- The code is a Python program that checks if a year is a leap year.
- The function is_leap_year takes a year as input and returns True if it is a leap year, False otherwise.
- The function first checks if the year is divisible by 4.
- If it is, then it checks if the year is divisible by 100.
- If it is, then it checks if the year is divisible by 400.
- If it is, then it returns True.
- Otherwise it returns False.
- The code is a Python program that checks if a year is a leap year.
- The function is_leap_year takes a year as input and returns True if it is a leap year, False otherwise.
- The function first checks if the year is divisible by 4.
- If it is, then it checks if the year is divisible by 100.

Justification:

This task demonstrates how AI tools like Cursor AI generate correct conditional logic for real-world calendar validation problems. Using multiple prompts helps analyze how prompt variation affects code structure, clarity, and reusability.

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

Prompt:

Write a Python program to calculate the sum of odd and even numbers in a given tuple. After writing the basic logic, refactor the code to make it more efficient, readable, and Pythonic without changing the output.

Code & Output:

lab-2.1.py > ...

```python
# Write a Python program to calculate the sum of odd and even numbers in a given tuple.
# After writing the basic logic, refactor the code to make it more efficient, readable,
# and Pythonic without changing the output.
tuple = tuple[int, ...](map[int](int, input("Enter a tuple of numbers: ").split()))
odd_sum = 0
even_sum = 0
for i in tuple:
    if i % 2 == 0:
        even_sum += i
    else:
        odd_sum += i
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

Problems Output Debug Console Terminal Ports powershell

```
PS C:\Users\danda\OneDrive\Documents\3.2\AIA> & C:\Users\danda\AppData\Local\Programs\Python\Python313\python.exe c:/Users/danda/OneDrive/Documents/3.2/AIA/lab-2.1.py
Enter a tuple of numbers: 1 2 3 4 5 6 7 8 9
Sum of odd numbers: 25
Sum of even numbers: 20
PS C:\Users\danda\OneDrive\Documents\3.2\AIA>
```

Explanation:

- The code is a Python program that calculates the sum of odd and even numbers in a given tuple.
- The program first initializes two variables, odd_sum and even_sum, to 0.

- Then it iterates through the tuple and adds the odd numbers to odd_sum and the even numbers to even_sum.
- Finally it prints the sum of odd numbers and the sum of even numbers.
- The code is a Python program that calculates the sum of odd and even numbers in a given tuple.
- The program first initializes two variables, odd_sum and even_sum, to 0.
- Then it iterates through the tuple and adds the odd numbers to odd_sum and the even numbers to even_sum.
- Finally it prints the sum of odd numbers and the sum of even numbers.
- The code is a Python program that calculates the sum of odd and even numbers in a given tuple.
- The program first initializes two variables, odd_sum and even_sum, to 0.
- Then it iterates through the tuple and adds the odd numbers to odd_sum and the even numbers to even_sum.
- Finally it prints the sum of odd numbers and the sum of even numbers.

Justification:

This task ensures that the student first applies their own logic to solve the problem before using AI assistance. AI refactoring is then used to improve code readability and efficiency while maintaining the same output.