

Assignment-2.1

S.Rajesh 2303A52301 B-35

Task 1: Statistical Summary for Survey Data Scenario:

You are a data analyst intern working with survey responses stored as numerical lists.

Task:

Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

Output:

Screenshot of Gemini prompt and result

```
def calculate_stats(numbers):
    """
    Calculates the mean, minimum, and maximum of a list of numbers.

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

    Returns:
        tuple: A tuple containing (mean, minimum, maximum).
        Returns (None, None, None) if the list is empty.
    """
    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 = [10, 20, 30, 40, 50]
mean, minimum, maximum = calculate_stats(my_list)

print("List: {}".format(my_list))
print("Mean: {}".format(mean))
print("Minimum: {}".format(minimum))

D generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.
◆ Here's a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values. I've also included an example of how to use it.

Accept & Run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

Correct Python function Output
shown in Colab

```
[1] ✓ Os def calculate_stats(numbers):
    """
    Calculates the mean, minimum, and maximum of a list of numbers.

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

    Returns:
        tuple: A tuple containing (mean, minimum, maximum).
        Returns (None, None, None) if the list is empty.
    """
    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

[2] ✓ Os # Example usage:
my_list = [10, 20, 30, 40, 50]
mean, minimum, maximum = calculate_stats(my_list)

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

empty_list = []

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 = [10, 20, 30, 40, 50]
mean, minimum, maximum = calculate_stats(my_list)

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

empty_list = []
mean_empty, min_empty, max_empty = calculate_stats(empty_list)
print(f"\nEmpty List Stats: Mean={mean_empty}, Min={min_empty}, Max={max_empty}")

... List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50

Empty List Stats: Mean=None, Min=None, Max=None
```

Task 2: Armstrong Number – AI Comparison Scenario:

You are evaluating AI tools for numeric validation logic.

?

Task:

Generate an Armstrong number checker using Gemini and GitHub Copilot.

Compare their outputs, logic style, and clarity.

The screenshot shows a Mac desktop with a Chrome browser window open. The browser has several tabs: 'Lab Assignment 2.1', 'ai assisted21.py - Colab', 'Introducing GPT-5 | OpenAI', and 'Google Gemini Python Task'. The main content area displays a Python code snippet for checking Armstrong numbers. A Gemini AI interface is overlaid on the browser window, showing a prompt and a response. The prompt asks for an Armstrong number checker. The response from Gemini provides the code and some explanatory text. The desktop dock at the bottom shows various application icons.

```
def is_armstrong_number(num):
    """
    Checks if a number is an Armstrong number.

    An Armstrong number (also known as a narcissistic number, pluperfect digital invariant, or plus perfect number) is a number that is the sum of its own digits each raised to the power of the number of digits.

    Args:
        num (int): An integer to check.

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(num, int) or num < 0:
        return False

    # Convert number to string to get digits
    num_str = str(num)
    n_digits = len(num_str)

    armstrong_sum = 0
    for digit_char in num_str:
        digit = int(digit_char)
        armstrong_sum += digit ** n_digits

    return armstrong_sum == num
```

D Generate an Armstrong number checker
Sure, I can help you with that! Here's a Python function to check if a number is an Armstrong number, along with some examples to demonstrate its use.
Accept & Run Accept Cancel

What can I help you build?
Gemini 2.5 Flash

The screenshot shows a Mac desktop with a Chrome browser window open. The browser has several tabs:

- Lab Assignment 2.1
- ai assisted2.1.py - Colab
- Introducing GPT-5 | OpenAI
- Google Gemini Python Task

The Colab tab contains the following Python code:

```
[3] ✓ 0s def is_armstrong_number(num):  
    """  
    Checks if a number is an Armstrong number.  
    An Armstrong number (also known as a narcissistic number, pluperfect digital invariant, or plus perfect number) is a number that is the sum of its own digits each raised to the power of the number of digits.  
    Args:  
        num (int): An integer to check.  
    Returns:  
        bool: True if the number is an Armstrong number, False otherwise.  
    """  
    if not isinstance(num, int) or num < 0:  
        return False  
  
    # Convert number to string to get digits and count length  
    num_str = str(num)  
    n_digits = len(num_str)  
  
    armstrong_sum = 0  
    for digit_char in num_str:  
        digit = int(digit_char)  
        armstrong_sum += digit ** n_digits  
  
    return armstrong_sum == num
```

Below the code, there is a comment block:

```
[4] ✓ 0s # Example usage:  
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}")
```

The status bar at the bottom of the screen shows the time as 2:01PM and the Python version as Python 3.

Using vs and GitHub copilot

A screenshot of a Mac desktop environment. At the top, there's a menu bar with Apple, Chrome, File, Edit, View, History, Bookmarks, Profiles, Tab, Window, Help. Below the menu bar, there are several tabs in a browser window: "Lab Assignment 2.1", "ai assisted2.1.py - Colab", "Introducing GPT-5 | OpenAI", and "Google Gemini Python Task". The "ai assisted2.1.py - Colab" tab is active, showing code in a Jupyter notebook cell:

```
armstrong_sum += digit ** n_digits
return armstrong_sum == num
```

[3] ✓ 0s [4] ✓ 0s

Example usage:
print("Is 9 an Armstrong number? {is_armstrong_number(9)}")
print("Is 153 an Armstrong number? {is_armstrong_number(153)}")
print("Is 370 an Armstrong number? {is_armstrong_number(370)}")
print("Is 371 an Armstrong number? {is_armstrong_number(371)}")
print("Is 407 an Armstrong number? {is_armstrong_number(407)}")
print("Is 1634 an Armstrong number? {is_armstrong_number(1634)}")
print("Is 123 an Armstrong number? {is_armstrong_number(123)}")
print("Is 0 an Armstrong number? {is_armstrong_number(0)}")
print("Is -153 an Armstrong number? {is_armstrong_number(-153)}")

... Is 9 an Armstrong number? True
Is 153 an Armstrong number? True
Is 370 an Armstrong number? True
Is 371 an Armstrong number? True
Is 407 an Armstrong number? True
Is 1634 an Armstrong number? True
Is 123 an Armstrong number? False
Is 0 an Armstrong number? True
Is -153 an Armstrong number? False

The terminal window below shows the system tray and some application icons.

A screenshot of the GitHub Copilot Chat interface. On the left is a sidebar with icons for Welcome, ai2.1.py (1 file), and Extension: GitHub Copilot Chat. The main area shows a code editor for "ai2.1.py" with the following content:

```
#Armstrong Number
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    digits = str(num)
    num_digits = len(digits)

    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in digits)

    # An Armstrong number is one where the sum of the powers equals the original number
    return sum_of_powers == num
```

The right side of the interface has a "Build with Agent" section with the following text:

AI responses may be inaccurate.
Generate Agent
Instructions to onboard AI onto your codebase.

Below this is a "Describe what to build next" input field with placeholder text "Describe what to build next" and a "Next" button.

At the bottom, there are status indicators: Screen Reader Optimized, Ln 2, Col 30, Spaces: 4, UTF-8, LF, Python 3.14.2, and a bell icon.

Task 3: Leap Year Validation Using Cursor AI

The screenshot shows a VS Code interface with the following details:

- Editor:** The main editor window displays a Python script named `ai2.1.py` containing code to check if a number is an Armstrong number. The code uses a function `is_armstrong_number` to calculate the sum of digits raised to the power of their count and compare it to the original number.
- Terminal:** The terminal window shows the output of running the script with the input `153`, which correctly identifies it as an Armstrong number.
- Extension:** A "Extension: GitHub Copilot Chat" panel is open on the right, showing a message from the AI agent.
- Bottom Bar:** The status bar at the bottom indicates the file is a Python script, the line and column numbers (Ln 17, Col 53), and the file size (3.14.2).

Scenario:

You are validating a calendar module for a backend system.

Task:

Use Cursor AI to generate a Python program that checks whether a given year is a leap year.

Use at least two different prompts and observe changes in code.

The screenshot shows a code editor interface with two tabs: "ai assistedcoding" and "ai2.1.py". The "ai2.1.py" tab contains the following Python code:

```
def is_leap_year(year: int) -> bool:
    """Return True if 'year' is a leap year, else False."""
    if year % 400 == 0:
        return True
    if year % 100 == 0:
        return False
    return year % 4 == 0

# Examples:
# print(is_leap_year(2000)) # True
# print(is_leap_year(1900)) # False
# print(is_leap_year(2024)) # True
```

Below the code, a note states: "This follows the Gregorian rules: divisible by 4 is leap, except centuries unless divisible by 400. Let me know if you want it saved to [ai2.1.py](#) or wrapped with CLI input."

On the left side of the editor, there is a sidebar titled "Plan, @ for context, / for commands" which includes a "Agent" dropdown and some icons.

At the bottom of the screen, there is a terminal window titled "ai assistedcoding" showing the command line and its output. The terminal output includes:

```
/usr/local/bin/python3 "/Users/nithyadugala/Desktop/ai assistedcoding/ai2.1.py"
● nithyadugala@Nithyas-MacBook-Air-2 ai assistedcoding % /usr/local/bin/python3 "/Users/nithyadugala/Desktop/ai assistedcoding/ai2.1.py"
True
False
True
○ nithyadugala@Nithyas-MacBook-Air-2 ai assistedcoding %
```

Other prompt:

The screenshot shows a code editor window titled "ai assistedcoding". The main area displays a Python file named "ai2.1.py" with the following code:

```
ai2.1.py •  
ai2.1.py > is_leap_year  
1 def is_leap_year(year: int) -> bool:  
2     """Return True if 'year' is a leap year, else False."""  
3     if year % 400 == 0:  
4         return True  
5     if year % 100 == 0:  
6         return False  
7     return year % 4 == 0
```

Below the code, a note says: "To use it interactively, run `python ai2.1.py` and adjust or add `print` lines for other years." To the right, a sidebar titled "AI ASSISTEDCODING" lists "ai2.1.py", "Assignment1.5.pdf", and "assignment-1.py". At the bottom, status bars show "Plan, @ for context, / for commands", "Agent Auto", and "Cursor Tab Ln 7, Col 25 Spaces: 4 UTF-8 LF () Python Python 3.14.2".

task 4: Student Logic + AI Refactoring (Odd/Even Sum) ☰ Scenario:
Company policy requires developers to write logic before using AI.

☐ Task:

Write a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it using any AI tool. Expected Output:

- ☐ Original code
- ☐ Refactored code
- ☐ Explanation of improvements

The screenshot shows a code editor interface with two main panes. The left pane displays a Python script named `ai2.1.py` containing a function `is_leap_year` that checks if a given year is a leap year according to standard rules. The right pane shows a terminal window where the script is run, demonstrating its functionality with examples like 2000, 1900, and 2024.

```
def is_leap_year(year: int) -> bool:
    """Return True if 'year' is a leap year, else False."""
    if year % 400 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 4 == 0:
        return True
    return False

print(is_leap_year(2000)) # True
print(is_leap_year(1900)) # False
print(is_leap_year(2024)) # True
```

Terminal output:

```
/usr/local/bin/python3 "/Users/nithyadugyala/Desktop/ai assistedcoding/ai2.1.py"
nithyadugyala@Nithyas-MacBook-Air-2: ~ ai assistedcoding % /usr/local/bin/python3 "/Users/nithyadugyala/Desktop/ai assistedcoding/ai2.1.py"
nithyadugyala@Nithyas-MacBook-Air-2: ~ ai assistedcoding % /usr/local/bin/python3 "/Users/nithyadugyala/Desktop/ai assistedcoding/ai2.1.py"
True
False
True
nithyadugyala@Nithyas-MacBook-Air-2: ~ ai assistedcoding %
```

The screenshot shows two instances of the Visual Studio Code (VS Code) interface, both displaying the same Python file, `ai2.1.py`. The file contains a function to calculate the sum of odd and even numbers from a tuple.

```
11
12 #Write a Python program that calculates the sum of odd and even numbers
13 #in a tuple
14
15 def sum_odd_even(numbers):
16     odd_sum = sum(x for x in numbers if x % 2 != 0)
17     even_sum = sum(x for x in numbers if x % 2 == 0)
18     return odd_sum, even_sum
19
20 # Example usage:
21 numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)
22 odd_total, even_total = sum_odd_even(numbers)
23 print(f"Sum of odd numbers: {odd_total}")
24 print(f"Sum of even numbers: {even_total}")
```

The interface includes a sidebar with various icons for file operations, a terminal window at the bottom showing the execution of the script and its output, and a status bar at the bottom right indicating the Python extension version (3.14.2).

Build with Agent

AI responses may be inaccurate.
Generate Agent
Instructions to onboard AI onto your codebase.

Build with Agent

AI responses may be inaccurate.
Generate Agent
Instructions to onboard AI onto your codebase.