

Assignment-2.1

p. nandhini

2303A51695

Batch-28

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

The screenshot shows a Google Colab notebook titled "ai assisted2.1.py". The code cell contains a Python function definition:

```
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
```

A Gemini AI sidebar is open, displaying a prompt and a generated response:

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

The screenshot shows the Google Colab interface with a dark theme. At the top, there are tabs for 'Lab Assignment 2.1', 'ai assisted2.1.py - Colab', 'Introducing GPT-5 | OpenAI', and 'Google Gemini Python Task'. The main area contains two code cells:

Cell 1:

```
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
```

Cell 2:

```
# 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
```

The output section shows the execution results for Cell 2:

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

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

The bottom of the screen shows a toolbar with various icons and a status bar indicating '1:59 PM' and 'Python 3'.

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 assisted2.ipynb - Colab", "Introducing GPT-5 | OpenAI", and "Google Gemini Python Task". The main content area displays a Jupyter Notebook cell with Python code for checking if a number is Armstrong. The code defines a function `is_armstrong_number` that converts an integer to a string, iterates over its digits, and calculates the sum of each digit raised to the power of its position. A docstring explains that an Armstrong number is one where the sum of its digits raised to the power of the number of digits equals the number itself. The code includes type annotations and handles edge cases like non-integer inputs and negative numbers. A Gemini AI interface is overlaid on the screen, showing a conversation where Gemini is asked to generate an Armstrong number checker and provides the same or similar code. 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
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Lab Assignment 2.1 | ai assisted2.1.py - Colab | Introducing GPT-5 | OpenAI | Google Gemini Python Task | Mon Jan 12 2:02PM

colab.research.google.com/drive/1h3JwUaKbdhweTZhzzWIWdmYJus8p4--2#scrollTo=f6ffffd4

Commands + Code + Text | Run all

RAM Disk School

```
[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
```

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

Variables Terminal 2:01PM Python 3



Using vs and GitHub copilot

The screenshot shows a Google Colab notebook interface. The top navigation bar includes 'Chrome', 'Edit', 'View', 'History', 'Bookmarks', 'Profiles', 'Tab', 'Window', and 'Help'. The tabs are 'Lab Assignment 2.1', 'ai assisted 2.1.py - Colab', 'Introducing GPT-5 | OpenAI', and 'Google Gemini Python Task'. The status bar at the bottom right shows 'Mon Jan 12 2:02PM'.

The main workspace contains the following code:

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

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

Below the code, the status bar shows 'RAM Disk' and other icons. The bottom navigation bar includes 'Variables', 'Terminal', and 'Python 3'.

The screenshot shows a code editor interface with the following details:

- File Bar:** ai2.1.py — ai assistedcoding
- Left Sidebar:** Welcome, ai2.1.py 1, Extension: GitHub Copilot Chat, and various icons for file operations, search, and AI integration.
- Code Editor Area:** The file ai2.1.py contains the following Python code:

```
1 #Armstrong Number
2 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
```
- Right Sidebar:** A "Build with Agent" section with a speech bubble icon, instructions, and a status bar at the bottom.
- Status Bar:** Screen Reader Optimized, Ln 2, Col 30, Spaces: 4, UTF-8, LF, Python, and a file navigation bar.

Task 3: Leap Year Validation Using Cursor AI

The screenshot shows a code editor interface with a Python file named `ai2.1.py` open. The code defines a function `is_armstrong_number` that checks if a given number is an Armstrong number. The terminal below shows the execution of the script and its output, confirming that 153 is an Armstrong number.

```
#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

# Example usage:
number = 153
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/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"
153 is an Armstrong number.
◆ nithyadugyala@Nithyas-MacBook-Air-2 ai assistedcoding %
```

Screen Reader Optimized Ln 17, Col 53 Spaces: 4 UTF-8 LF {} Python 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.

A screenshot of a code editor window titled "ai assistedcoding". The main pane displays a Python file named "ai2.1.py" with the following 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
```

The code follows the Gregorian rules: divisible by 4 is leap, except centuries unless divisible by 400. A note at the bottom says "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."

A screenshot of a code editor window titled "ai assistedcoding". The main pane displays a Python file named "ai2.1.py" with the following 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
```

The code is identical to the one in the previous screenshot. A note at the bottom says "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."

A screenshot of a terminal window titled "zsh". The command entered is:

```
/usr/local/bin/python3 "/Users/nithyadugala/Desktop/ai assistedcoding/ai2.1.py"
```

The output shows the results of running the script with three different years:

```
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 software interface for AI-assisted coding. At the top, there's a navigation bar with icons for back, forward, search, and refresh. The title bar says "ai assistedcoding".

The main area is a code editor with a dark theme. On the left, a sidebar displays the project structure under "AI ASSISTED CODING": "ai2.1.py" (selected), "Assignment1.5.pdf", and "assignment-1.py".

The code editor window contains the following Python code:

```
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 editor, a message says: "To use it interactively, run `python ai2.1.py` and adjust or add `print` lines for other years."

At the bottom left, a command palette shows "Plan, @ for context, / for commands" and "Agent: Auto". It also includes icons for file operations like open, save, and delete.

At the bottom right, there are status indicators: "Cursor Tab", "Ln 7, Col 25", "Spaces: 4", "UTF-8", "LF", "Python", "Python 3.14.2", and a magnifying glass icon.

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 user input box containing a prompt to generate a Python program for leap year validation. Below this is a code editor window titled 'ai2.1.py' showing the original code. The right pane shows the AI interface with a sidebar labeled 'AI ASSISTED CODING' containing files like 'ai2.1.py', 'Assignment1.5.pdf', and 'assignment-1.py'. The bottom pane is a terminal window showing the execution of the script and its output.

```
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
```

To use it interactively, run `python ai2.1.py` and adjust or add `print` lines for other years.

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

The screenshot shows two instances of the Visual Studio Code (VS Code) interface, both running on a Mac OS X system. The title bar for both instances reads "ai2.1.py — ai assistedcoding".

Top Instance (Line 11):

```
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
19     return odd_sum, even_sum
20
21 # Example usage:
22 numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)
23 odd_total, even_total = sum_odd_even(numbers)
24 print(f"Sum of odd numbers: {odd_total}")
25 print(f"Sum of even numbers: {even_total}")
```

Bottom Instance (Line 25):

```
25
26 #refactor the code of sum of odd or even numbers in a tuple
27 def sum_odd_even(numbers):
28     odd_sum = 0
29     even_sum = 0
30     for x in numbers:
31         if x % 2 == 0:
32             even_sum += x
33         else:
34             odd_sum += x
35
36     return odd_sum, even_sum
37
38 # Example usage:
39 numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)
40 odd_total, even_total = sum_odd_even(numbers)
41 print(f"Sum of odd numbers: {odd_total}")
42 print(f"Sum of even numbers: {even_total}")
```

Common UI Elements:

- Sidebar:** Includes icons for File, Find, Go, Open, Save, and others.
- Code Editor:** Displays the Python code.
- Terminal:** Shows the output of the code execution in the terminal tab.
- Output Tab:** Shows the output of the code execution in the output tab.
- Build with Agent Panel:** A sidebar panel on the right for interacting with AI features, including a "Build with Agent" button, AI response accuracy information, and generate agent instructions.
- Status Bar:** Shows the file path ("ai2.1.py"), screen reader optimization status, line and column numbers (e.g., "Ln 13, Col 12" or "Ln 35, Col 29"), spaces and tabs settings, encoding ("UTF-8"), line endings ("LF"), Python version ("Python 3.14.2"), and a notification icon.