

Assignment-2.1

p. nandhini

2303A51695

Batch-28

Task 1: Statistical Summary for Survey Data

Scenario:

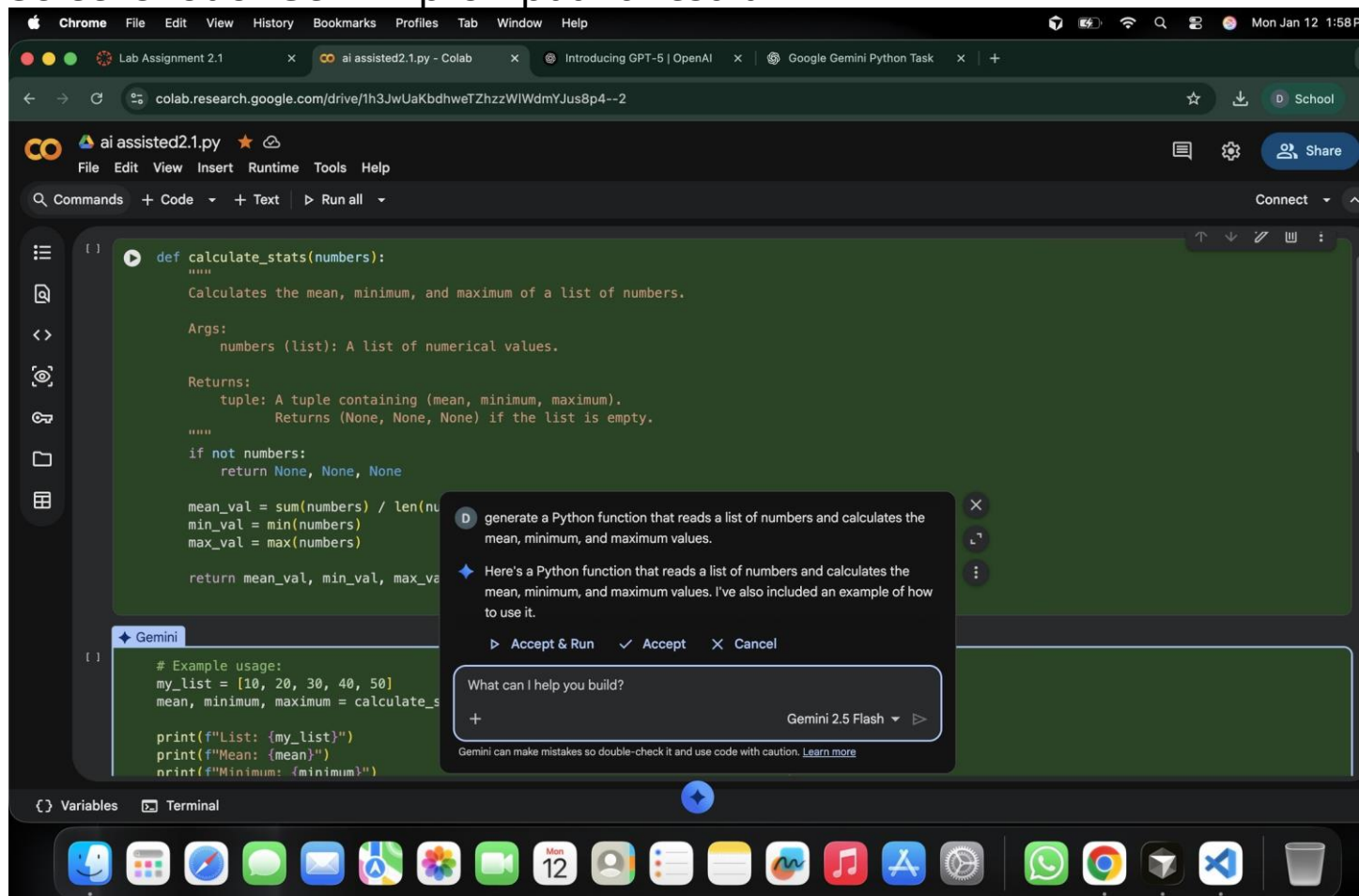
You are a data analyst intern working with survey responses stored as numerical lists.

❖ 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.

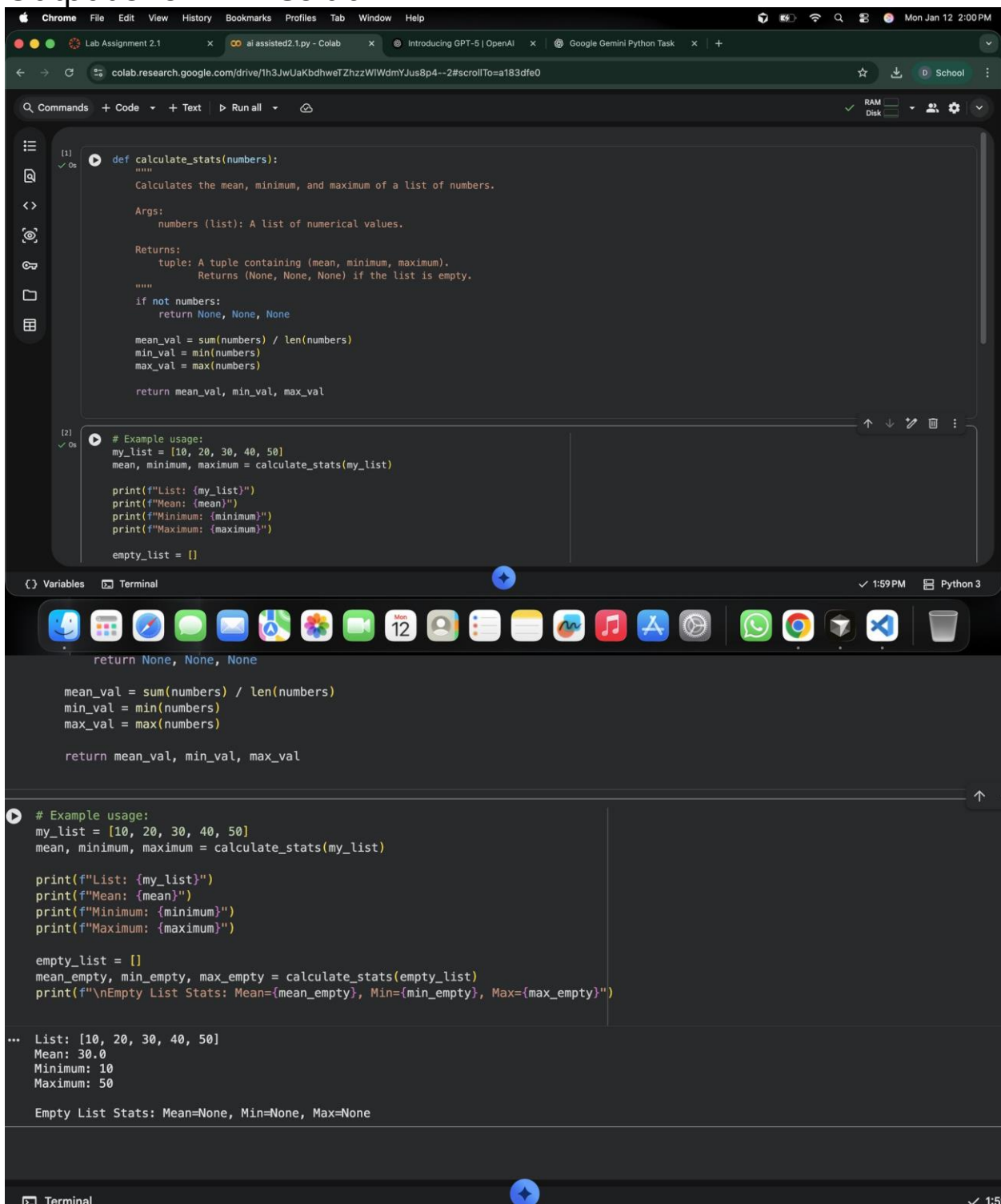
Output:

Screenshot of Gemini prompt and result



Correct Python function

Output shown in Colab



The screenshot shows a Google Colab notebook interface. The top bar includes the Chrome browser tabs and the Colab toolbar. The notebook contains two code cells. The first cell defines a function `calculate_stats` that calculates the mean, minimum, and maximum of a list of numbers. The second cell demonstrates the function's usage with a non-empty list and an empty list. The output of the second cell shows the results for both cases.

```
[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  
  
[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 = []  
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

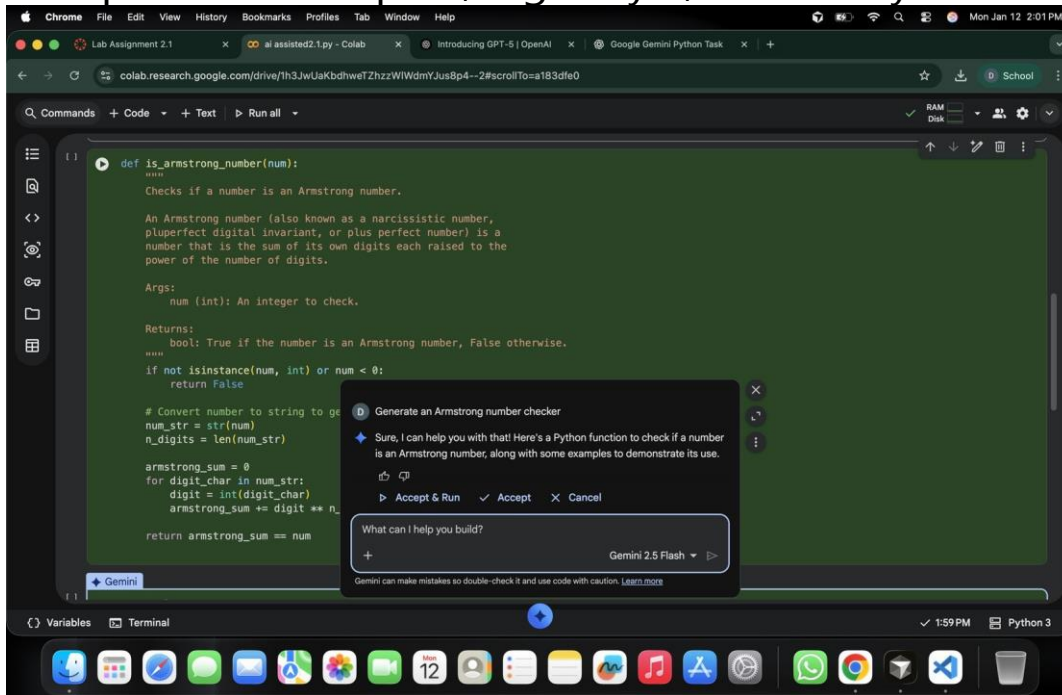
❖ Scenario:

You are evaluating AI tools for numeric validation logic.

❖ Task:

Generate an Armstrong number checker using Gemini and GitHub Copilot.

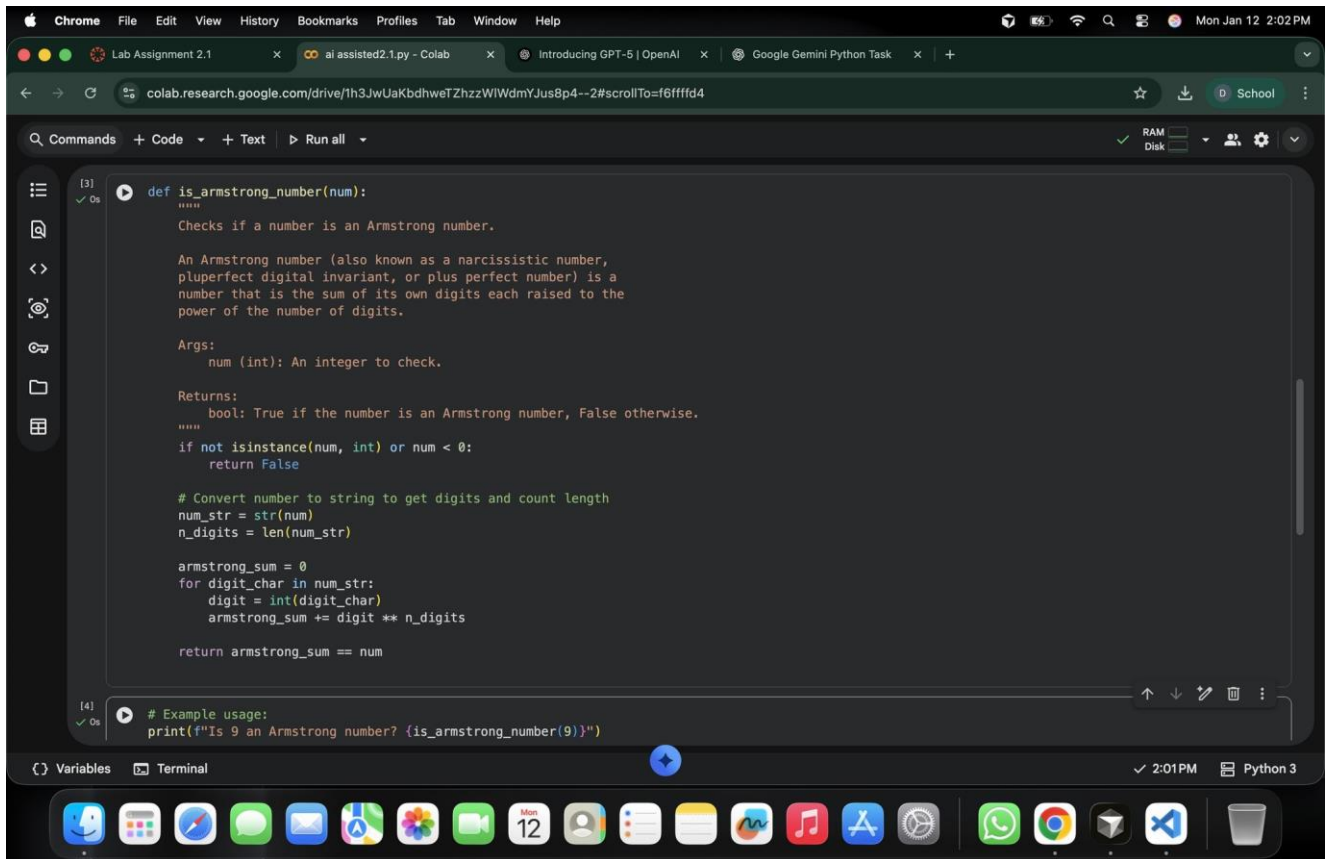
Compare their outputs, logic style, and clarity.



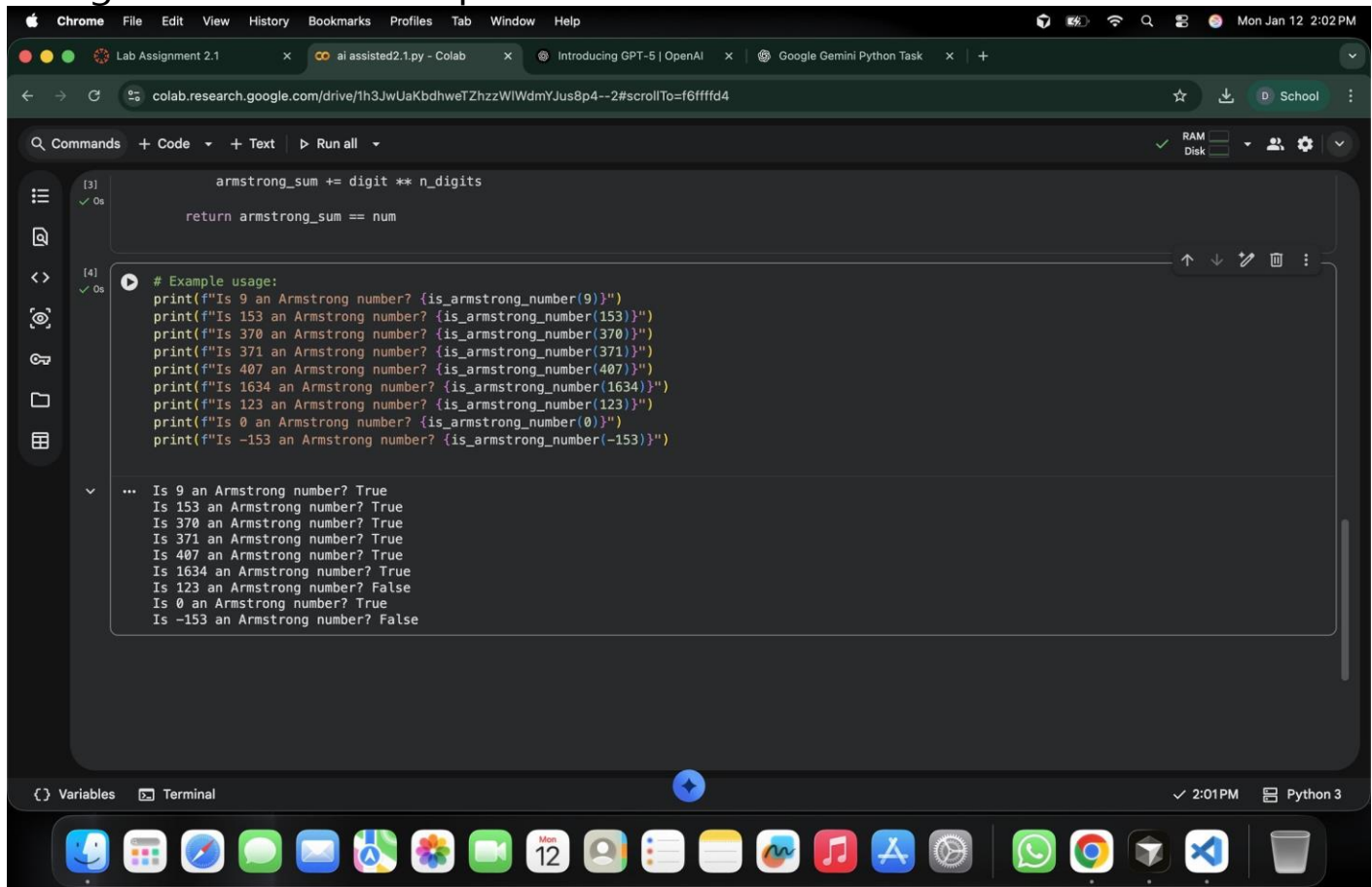
The screenshot shows a Google Colab notebook interface. The main code cell contains a Python function `def is_armstrong_number(num):` with docstrings and logic to check if a number is an Armstrong number. The function converts the number to a string, iterates over its digits, calculates the sum of each digit raised to the power of the number of digits, and returns `True` if the sum equals the original number, otherwise `False`.

Overlaid on the code is a Gemini chat window titled "Generate an Armstrong number checker". The chat history shows a user prompt and a Gemini response: "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." The chat window includes buttons for "Accept & Run", "Accept", and "Cancel". Below the chat window is a text input field with the placeholder "What can I help you build?" and a "Gemini 2.5 Flash" model selector.

```
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
```



Using vs and GitHub copilot



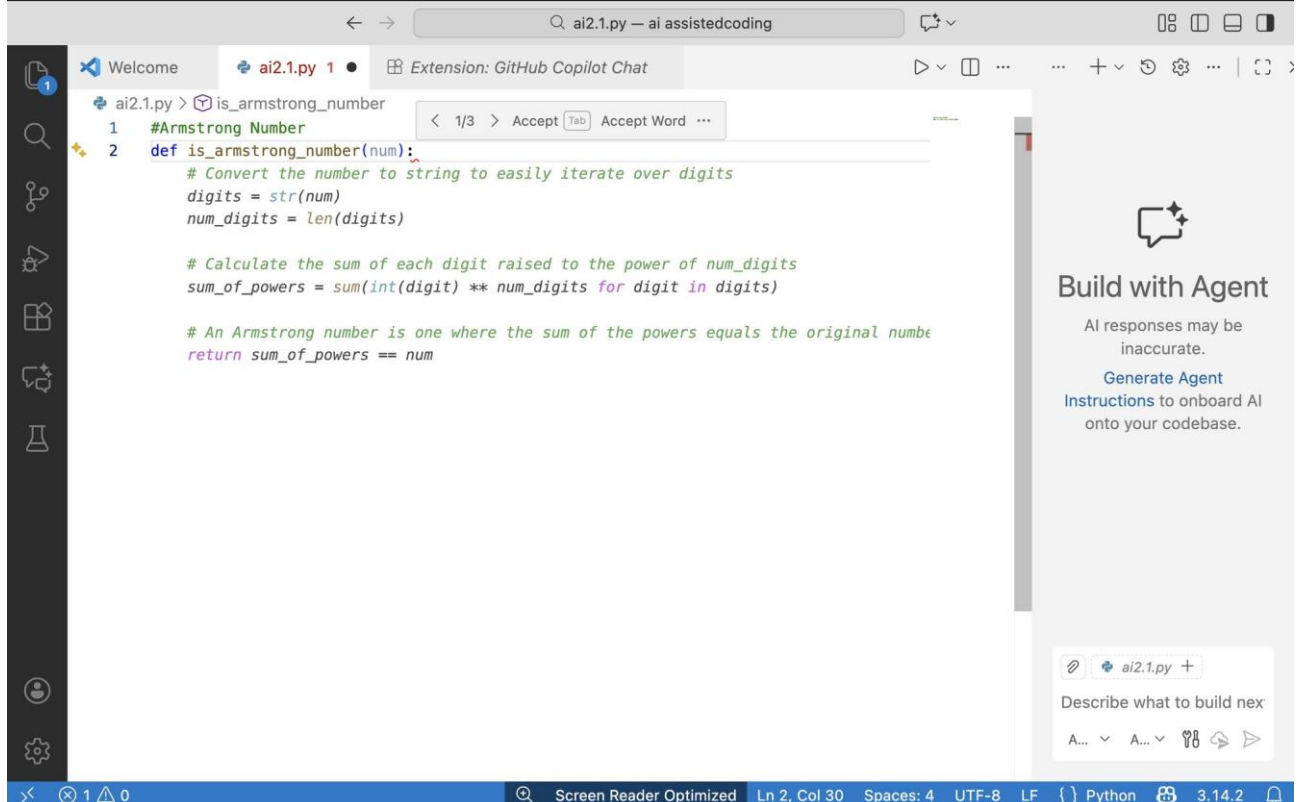
The screenshot shows the Google Colab interface. The top bar includes the Chrome menu and several open tabs: 'Lab Assignment 2.1', 'ai assisted2.1.py - Colab', 'Introducing GPT-5 | OpenAI', and 'Google Gemini Python Task'. The address bar shows the Colab URL. The main workspace has a dark theme and contains a Python script. The script defines a function `is_armstrong_number` and includes example usage calls. The output of the script is displayed below the code, showing the results of the function calls for various numbers.

```
[3]
armstrong_sum += digit ** n_digits

return armstrong_sum == num

[4]
# 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
```



The screenshot shows the VS Code interface. The top bar includes the search bar and several open tabs: 'Welcome', 'ai2.1.py 1', and 'Extension: GitHub Copilot Chat'. The main workspace shows the same Python script as the Colab screenshot, but with GitHub Copilot suggestions visible. The sidebar on the right contains a message from the 'Build with Agent' extension, stating that AI responses may be inaccurate and providing instructions to onboard AI onto the codebase.

```
ai2.1.py > is_armstrong_number
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
```

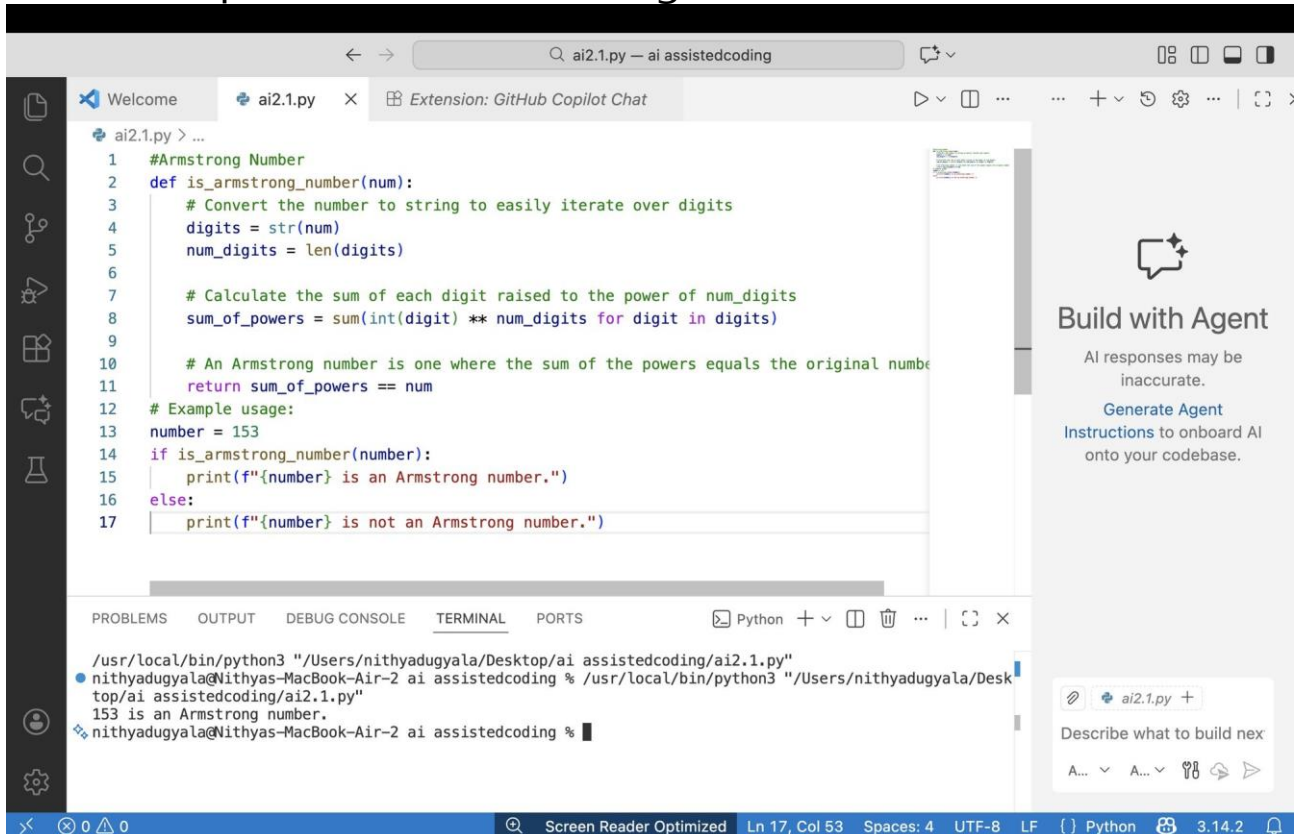
Build with Agent

AI responses may be inaccurate.

Generate Agent

Instructions to onboard AI onto your codebase.

Task 3: Leap Year Validation Using Cursor AI



The screenshot displays the Cursor AI IDE interface. The main editor window shows a Python file named `ai2.1.py` with the following code:

```
1 #Armstrong Number
2 def is_armstrong_number(num):
3     # Convert the number to string to easily iterate over digits
4     digits = str(num)
5     num_digits = len(digits)
6
7     # Calculate the sum of each digit raised to the power of num_digits
8     sum_of_powers = sum(int(digit) ** num_digits for digit in digits)
9
10    # An Armstrong number is one where the sum of the powers equals the original number
11    return sum_of_powers == num
12
13    # Example usage:
14    number = 153
15    if is_armstrong_number(number):
16        print(f"{number} is an Armstrong number.")
17    else:
18        print(f"{number} is not an Armstrong number.")
```

The terminal window at the bottom shows the execution of the script:

```
/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 %
```

On the right side of the IDE, there is a sidebar with the text "Build with Agent" and a button "Generate Agent". Below this, there is a section titled "Describe what to build next" with a text input field and a "Generate" button.

❖ 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 image shows the Cursor IDE interface with the 'ai assistedcoding' extension. The left sidebar contains a chat window with the following content:

Leap year validation: **Sent to Chat**

Now - Auto

Leap Year Validation code in python

Thought for 1s

Here's a concise Python function to validate leap years:

```
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
```

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.

Plan, @ for context, / for commands

Agent Auto

The main editor shows the file `ai2.1.py` with the following code:

```
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
8
9 # Examples:
10 # print(is_leap_year(2000)) # True
11 # print(is_leap_year(1900)) # False
12 # print(is_leap_year(2024)) # True
```

The right sidebar shows the 'AI ASSISTEDCODING' panel with a list of files: `ai2.1.py`, `Assignment1.5.pdf`, and `assignment-1.py`. The bottom panel shows the terminal output of the code execution:

```
zsh
/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 %
```

Other prompt:

The screenshot displays the Cursor IDE interface with the following components:

- Top Bar:** Shows the file name `ai2.1.py` and the editor title `ai assistedcoding`.
- Left Panel (Chat/Context):**
 - Buttons: `Leap year validation`, `Sent to Chat`, `Python leap year p`, `+`, and `...`.
 - Text: `Now - Auto`
 - Prompt: `generate a Python program that checks whether a given year is a leap year`
 - Status: `Explored 1 file`
 - Feedback: `Already in place. The program at ai2.1.py defines is_leap_year using the standard leap-year rules and prints a few examples:`
 - Code Preview (Lines 1-7):

```
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
```
 - Instructions: `To use it interactively, run python ai2.1.py and adjust or add print lines for other years.`
 - Input Field: `Plan, @ for context, / for commands`
 - Agent: `Agent - Auto`
- Editor (Center):**
 - File: `ai2.1.py`
 - Function: `is_leap_year`
 - Code (Lines 1-7):

```
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
```
- Right Panel (Outline):**
 - Section: `AI ASSISTEDCODING`
 - Files: `ai2.1.py`, `Assignment1.5.pdf`, and `assignment-1.py`
 - Buttons: `> OUTLINE` and `> TIMELINE`
- Bottom Bar:**
 - 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 displays the Cursor IDE interface. On the left, a chat window shows a prompt: "generate a Python program that checks whether a given year is a leap year". Below the prompt, it says "Already in place. The program at ai2.1.py defines is_leap_year using the standard leap-year rules and prints a few examples:". A code snippet for `ai2.1.py` is shown, defining a function `is_leap_year` and testing it with years 2000, 1900, and 2024. The main editor shows the full code for `ai2.1.py`, which includes docstrings and test cases. The bottom terminal window shows the command `python ai2.1.py` being executed, resulting in the output: `True`, `False`, and `True` for the respective years.

```
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

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

```
/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 %
```

A screenshot of the Visual Studio Code editor interface. The top bar shows the file name "ai2.1.py" and the extension "GitHub Copilot Chat". The main editor area displays a Python script generated by AI, which defines a function to calculate the sum of odd and even numbers in a tuple. The terminal at the bottom shows the command to run the script and its output. On the right side, there is a sidebar with the text "Build with Agent" and a warning about AI response accuracy.

[illegible]