AI Assisted Coding -2

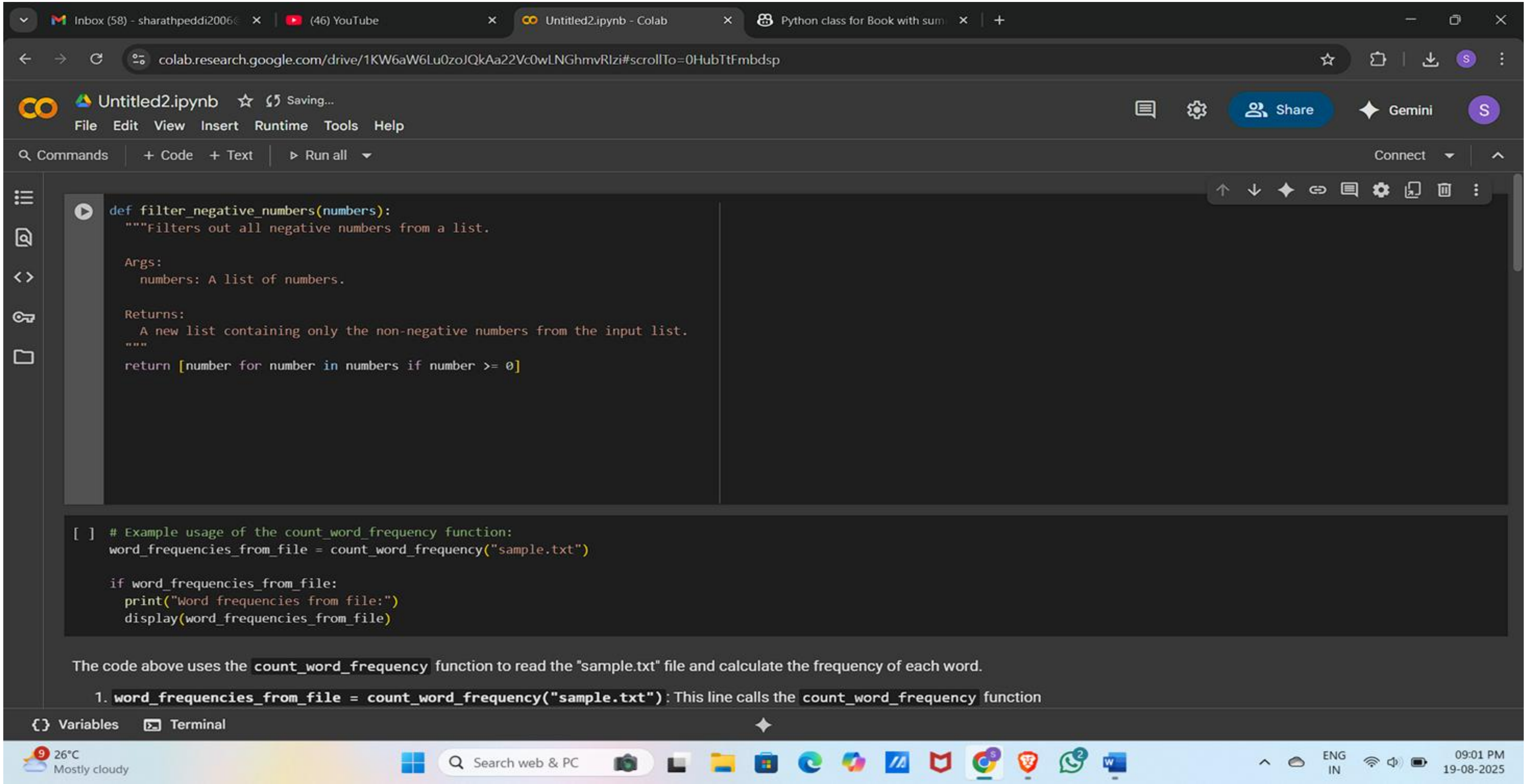
Name : Laxman

H.No : 2403A51283

Batch : 01

Subject : AI Assisted Coding

1)Use Gemini in Colab to write a function that filters out all negative numbers froma given list.



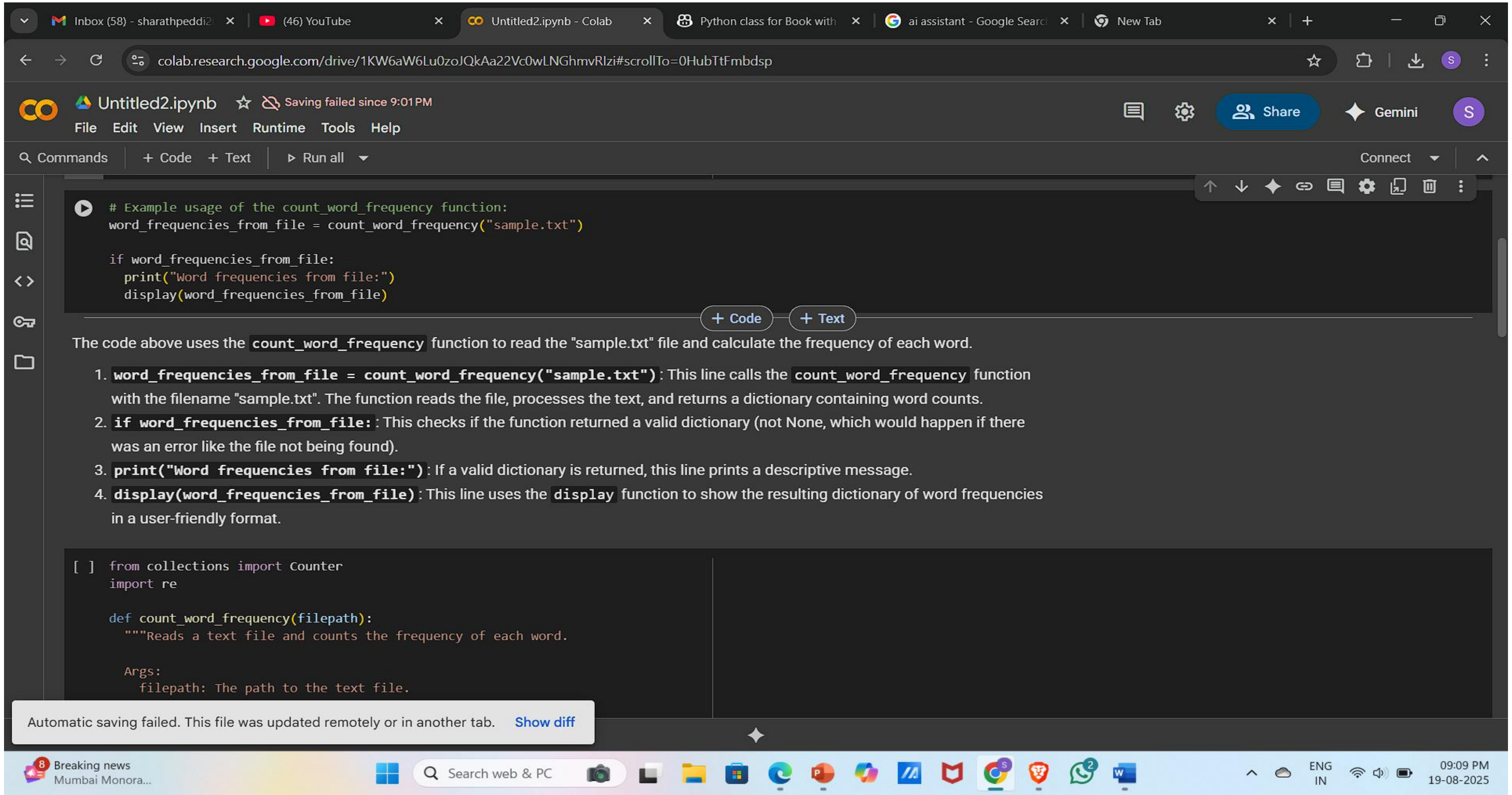
```python
def filter_negative_numbers(numbers):
  """Filters out all negative numbers from a list.

  Args:
    numbers: A list of numbers.

  Returns:
    A new list containing only the non-negative numbers from the input list.
  """
  return [number for number in numbers if number >= 0]
```

```python
# Example usage of the count_word_frequency function:
word_frequencies_from_file = count_word_frequency("sample.txt")

if word_frequencies_from_file:
  print("Word frequencies from file:")
  display(word_frequencies_from_file)
```

The code above uses the `count_word_frequency` function to read the "sample.txt" file and calculate the frequency of each word.

1. `word_frequencies_from_file = count_word_frequency("sample.txt")`: This line calls the `count_word_frequency` function

2)Ask Gemini to generate code that reads a text file and counts the frequency of each word. Then ask it to explain the code


Untitled2.ipynb
Saving failed since 9:01 PM
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
Connect

```
# Example usage of the count_word_frequency function:
word_frequencies_from_file = count_word_frequency("sample.txt")

if word_frequencies_from_file:
  print("Word frequencies from file:")
  display(word_frequencies_from_file)
```

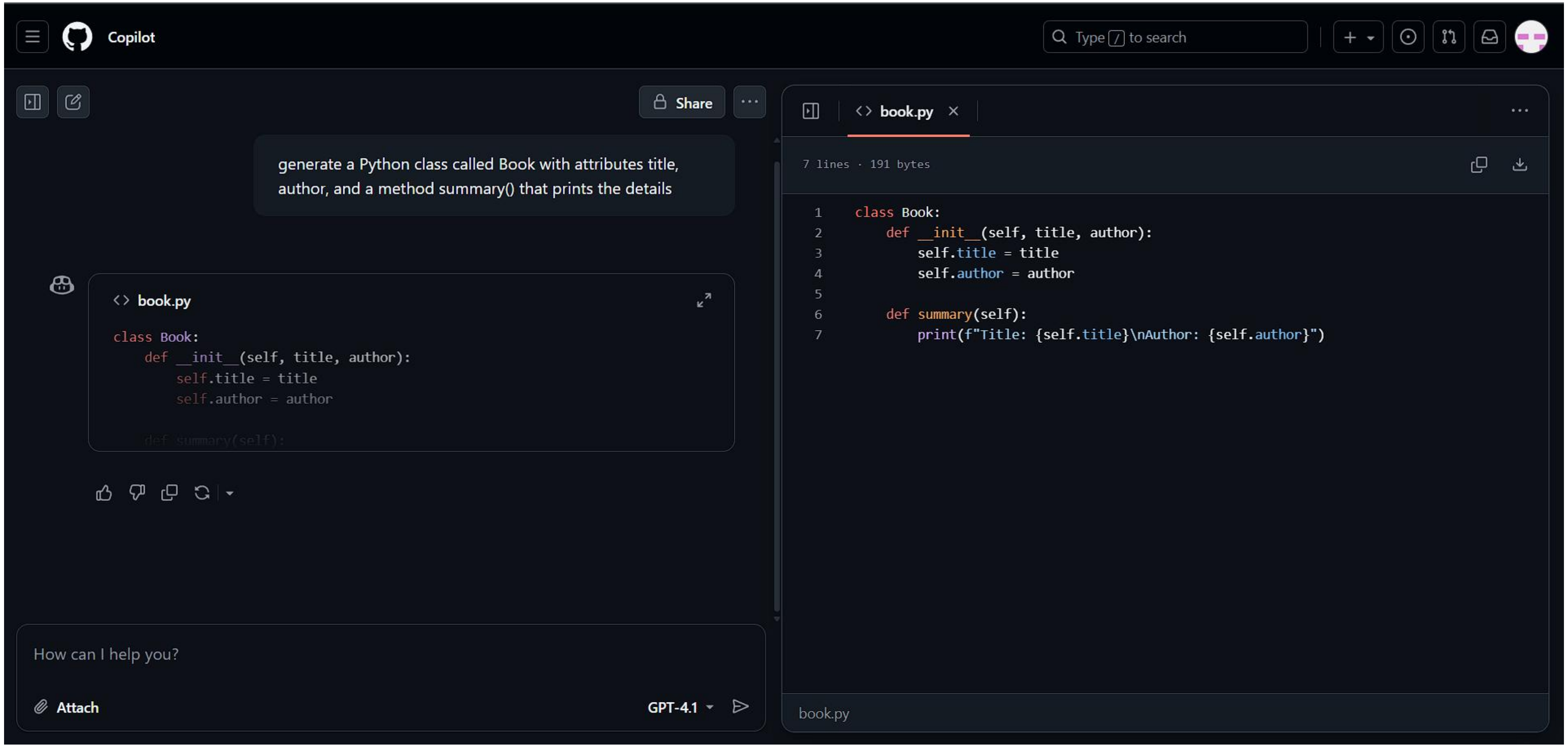
The code above uses the `count_word_frequency` function to read the "sample.txt" file and calculate the frequency of each word.

1. **`word_frequencies_from_file = count_word_frequency("sample.txt")`**: This line calls the `count_word_frequency` function with the filename "sample.txt". The function reads the file, processes the text, and returns a dictionary containing word counts.
2. **`if word_frequencies_from_file:`**: This checks if the function returned a valid dictionary (not None, which would happen if there was an error like the file not being found).
3. **`print("Word frequencies from file:")`**: If a valid dictionary is returned, this line prints a descriptive message.
4. **`display(word_frequencies_from_file)`**: This line uses the `display` function to show the resulting dictionary of word frequencies in a user-friendly format.

```
from collections import Counter
import re

def count_word_frequency(filepath):
  """Reads a text file and counts the frequency of each word.

  Args:
    filepath: The path to the text file.
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

3)Use Cursor AI to generate a Python class called Book with attributes title, author, and a method summary() that prints the details



generate a Python class called Book with attributes title, author, and a method summary() that prints the details

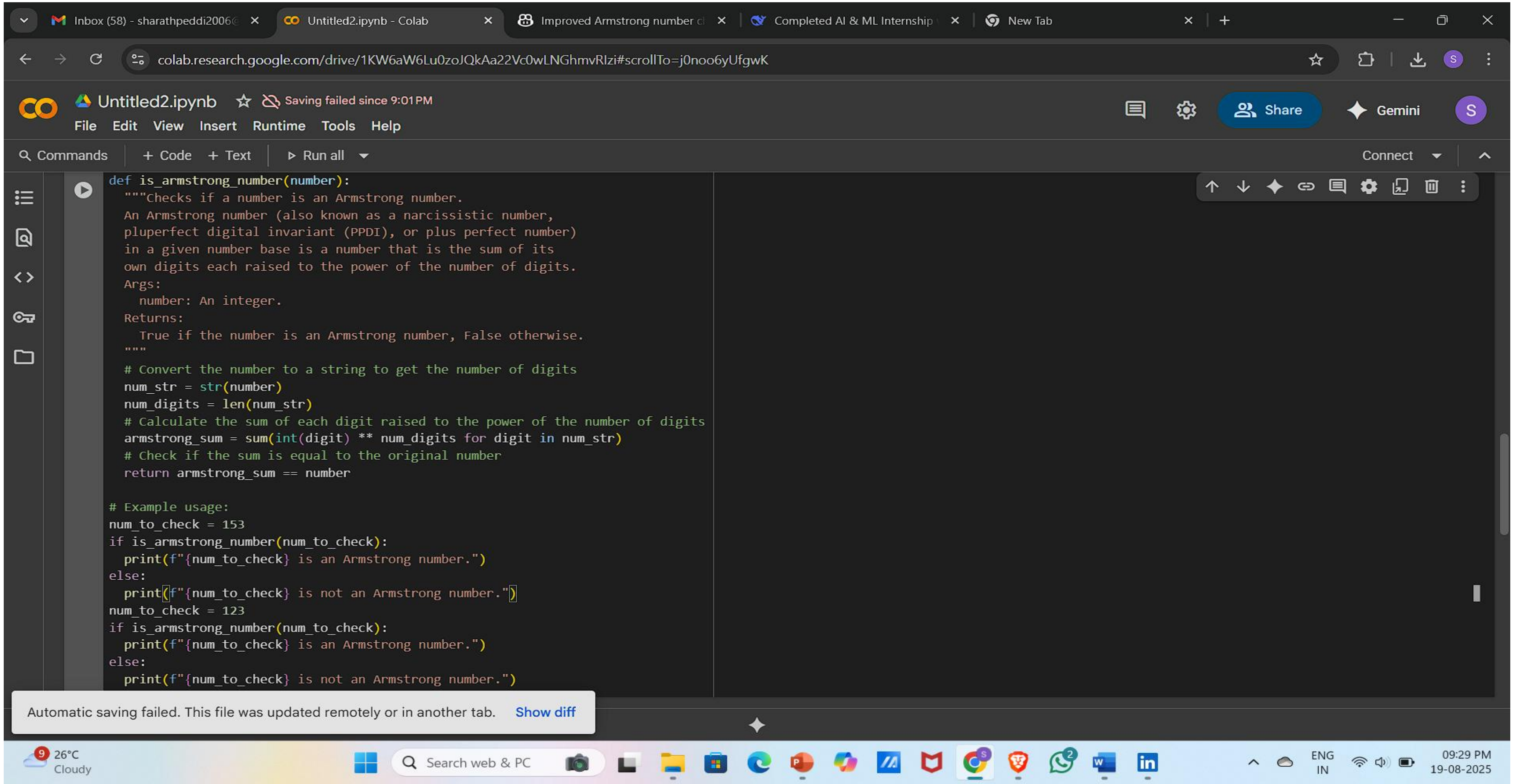
book.py

```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
```

book.py

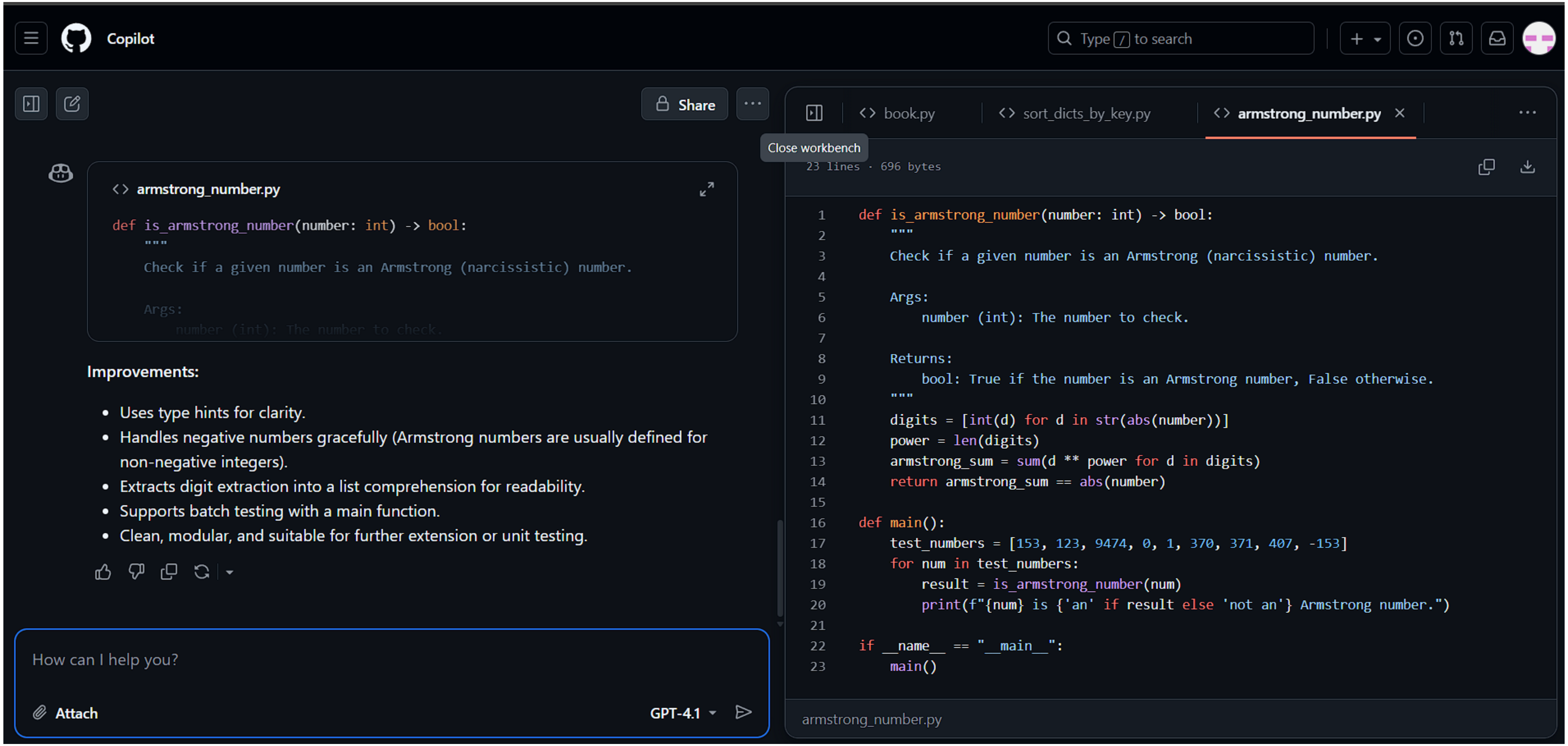
```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        print(f"Title: {self.title}\nAuthor: {self.author}")
```

4) Ask Gemini to write a program that checks whether a number is an Armstrong number, and then modify it using Cursor AI to improve performance or structure


Inbox (58) - sharathpeddi2006
Untitled2.ipynb - Colab
Improved Armstrong number c
Completed AI & ML Internship
New Tab
colab.research.google.com/drive/1KW6aW6Lu0zoJQkAa22Vc0wLNGhmvRlzi#scrollTo=j0noo6yUfgwK
Untitled2.ipynb
Saving failed since 9:01 PM
File
Edit
View
Insert
Runtime
Tools
Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect
26°C
Cloudy
Search web & PC
ENG
IN
09:29 PM
19-08-2025

```python
def is_armstrong_number(number):
  """Checks if a number is an Armstrong number.
  An Armstrong number (also known as a narcissistic number,
  pluperfect digital invariant (PPDI), or plus perfect number)
  in a given number base is a number that is the sum of its
  own digits each raised to the power of the number of digits.
  Args:
    number: An integer.
  Returns:
    True if the number is an Armstrong number, False otherwise.
  """
  # Convert the number to a string to get the number of digits
  num_str = str(number)
  num_digits = len(num_str)
  # Calculate the sum of each digit raised to the power of the number of digits
  armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
  # Check if the sum is equal to the original number
  return armstrong_sum == number

# Example usage:
num_to_check = 153
if is_armstrong_number(num_to_check):
  print(f"{num_to_check} is an Armstrong number.")
else:
  print(f"{num_to_check} is not an Armstrong number.")
num_to_check = 123
if is_armstrong_number(num_to_check):
  print(f"{num_to_check} is an Armstrong number.")
else:
  print(f"{num_to_check} is not an Armstrong number.")
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff



armstrong_number.py

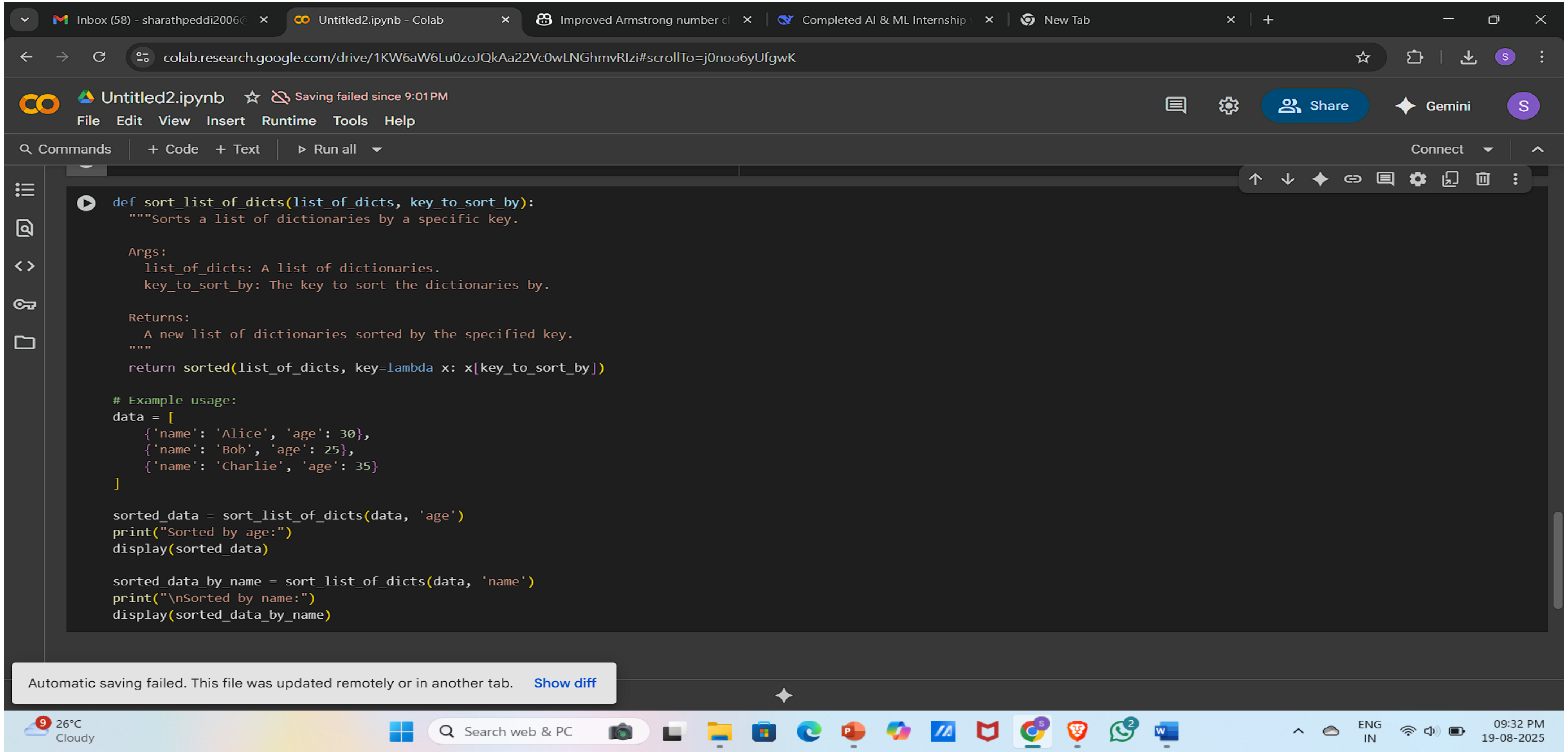
```python
def is_armstrong_number(number: int) -> bool:
    """
    Check if a given number is an Armstrong (narcissistic) number.

    Args:
        number (int): The number to check.
```

Improvements:

- Uses type hints for clarity.
- Handles negative numbers gracefully (Armstrong numbers are usually defined for non-negative integers).
- Extracts digit extraction into a list comprehension for readability.
- Supports batch testing with a main function.
- Clean, modular, and suitable for further extension or unit testing.

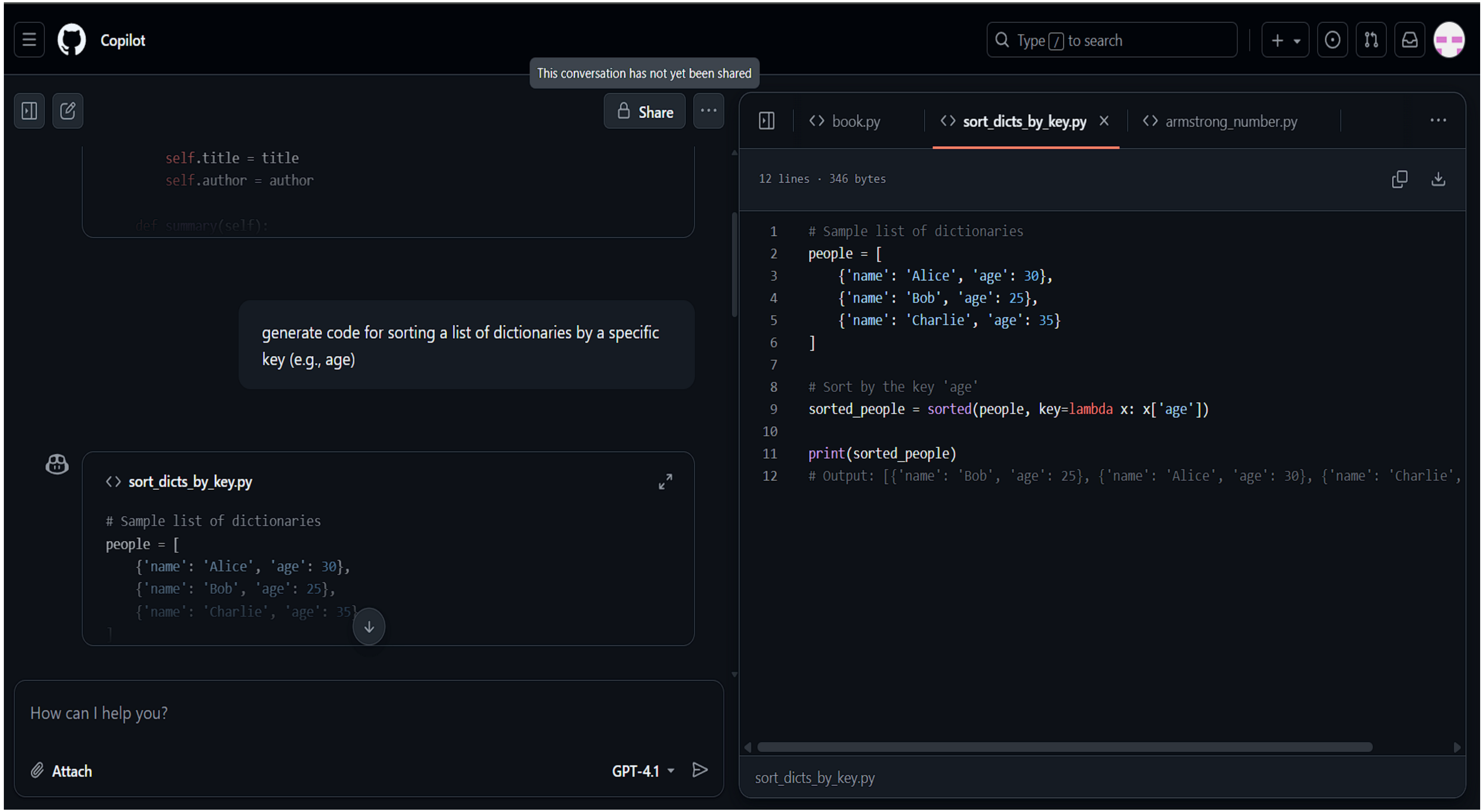
```python
def is_armstrong_number(number: int) -> bool:
    """
    Check if a given number is an Armstrong (narcissistic) number.

    Args:
        number (int): The number to check.

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    digits = [int(d) for d in str(abs(number))]
    power = len(digits)
    armstrong_sum = sum(d ** power for d in digits)
    return armstrong_sum == abs(number)

def main():
    test_numbers = [153, 123, 9474, 0, 1, 370, 371, 407, -153]
    for num in test_numbers:
        result = is_armstrong_number(num)
        print(f"{num} is {'an' if result else 'not an'} Armstrong number.")

if __name__ == "__main__":
    main()
```

5) Use both Gemini and Cursor AI to generate code for sorting a list of dictionaries by a specific key (e.g., age).


Inbox (58) - sharathpeddi2006
Untitled2.ipynb - Colab
Improved Armstrong number c
Completed AI & ML Internship
New Tab
colab.research.google.com/drive/1KW6aW6Lu0zoJQkAa22Vc0wLNGhmvRIzi#scrollTo=j0noo6yUfgwK
Untitled2.ipynb
Saving failed since 9:01 PM
File
Edit
View
Insert
Runtime
Tools
Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect
26°C
Cloudy
Search web & PC
ENG
IN
09:32 PM
19-08-2025

```python
def sort_list_of_dicts(list_of_dicts, key_to_sort_by):
  """Sorts a list of dictionaries by a specific key.

  Args:
    list_of_dicts: A list of dictionaries.
    key_to_sort_by: The key to sort the dictionaries by.

  Returns:
    A new list of dictionaries sorted by the specified key.
  """
  return sorted(list_of_dicts, key=lambda x: x[key_to_sort_by])

# Example usage:
data = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]

sorted_data = sort_list_of_dicts(data, 'age')
print("Sorted by age:")
display(sorted_data)

sorted_data_by_name = sort_list_of_dicts(data, 'name')
print("\nSorted by name:")
display(sorted_data_by_name)
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff



```
        self.title = title
        self.author = author

    def summary(self):
```

generate code for sorting a list of dictionaries by a specific key (e.g., age)

sort_dicts_by_key.py

```
# Sample list of dictionaries
people = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]
```

```
# Sample list of dictionaries
people = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]

# Sort by the key 'age'
sorted_people = sorted(people, key=lambda x: x['age'])

print(sorted_people)
# Output: [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie',
```