LAB ASSIGNMENT – 2.2

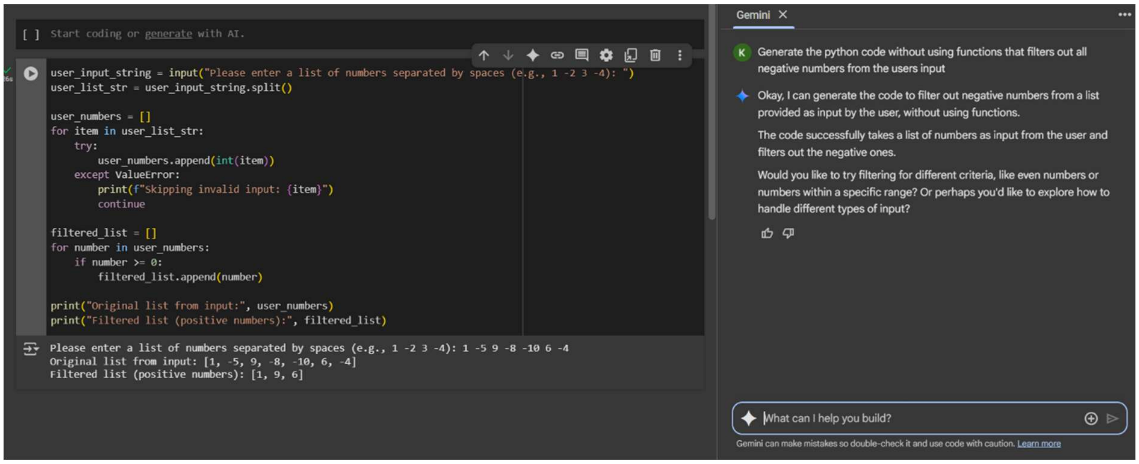
AI Assisted coding

Name: V.Vamshi

Htno: 2403A52016

Task -1:

- **Description:** Use Gemini in colab to write a function that filter out all negative numbers from a given list
- **Prompt:** Generate the python code without using functions that filters out all negative numbers from the users input.



```python
user_input_string = input("Please enter a list of numbers separated by spaces (e.g., 1 -2 3 -4): ")
user_list_str = user_input_string.split()

user_numbers = []
for item in user_list_str:
    try:
        user_numbers.append(int(item))
    except ValueError:
        print(f"Skipping invalid input: {item}")
        continue

filtered_list = []
for number in user_numbers:
    if number >= 0:
        filtered_list.append(number)

print("Original list from input:", user_numbers)
print("Filtered list (positive numbers):", filtered_list)
```

```
Please enter a list of numbers separated by spaces (e.g., 1 -2 3 -4): 1 -5 9 -8 -10 6 -4
Original list from input: [1, -5, 9, -8, -10, 6, -4]
Filtered list (positive numbers): [1, 9, 6]
```

K Generate the python code without using functions that filters out all negative numbers from the users input

Okay, I can generate the code to filter out negative numbers from a list provided as input by the user, without using functions.

The code successfully takes a list of numbers as input from the user and filters out the negative ones.

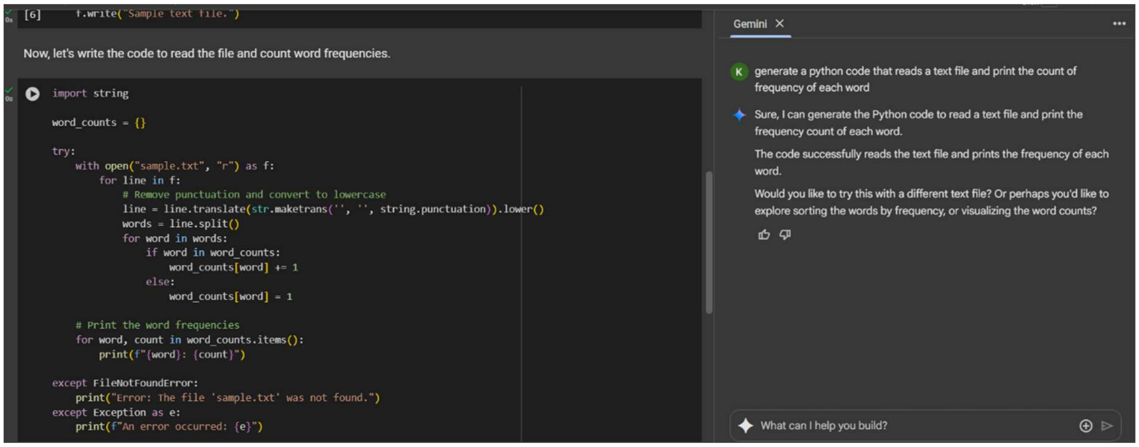
Would you like to try filtering for different criteria, like even numbers or numbers within a specific range? Or perhaps you'd like to explore how to handle different types of input?

Gemini can make mistakes so double-check it and use code with caution. Learn more

- **Observation:** Gemini can understand the task very clearly.It take very less time to complete the task and the output is accurate and matches the example provided.

Task-2:

- **Description:** Ask Gemini to generate code that reads a text file and counts the frequency of each word. Then ask it to explain the code.
- **Prompt:** Generate a python code that reads a text file and print the count of frequency of each word.



```
[6]     f.write("Sample text file.")
```

Now, let's write the code to read the file and count word frequencies.

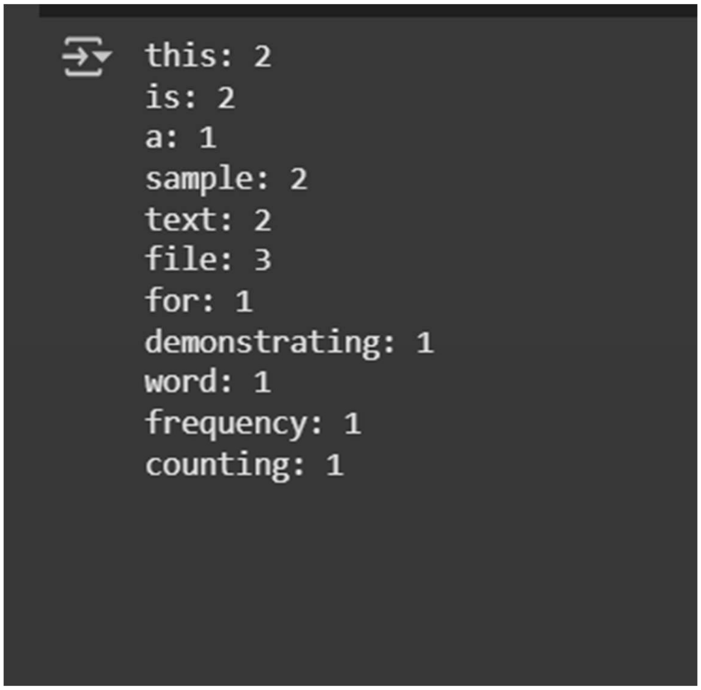
```
import string

word_counts = {}

try:
    with open("sample.txt", "r") as f:
        for line in f:
            # Remove punctuation and convert to lowercase
            line = line.translate(str.maketrans('', '', string.punctuation)).lower()
            words = line.split()
            for word in words:
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1

    # Print the word frequencies
    for word, count in word_counts.items():
        print(f"{word}: {count}")

except FileNotFoundError:
    print("Error: The file 'sample.txt' was not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

generate a python code that reads a text file and print the count of frequency of each word

Sure, I can generate the Python code to read a text file and print the frequency count of each word.

The code successfully reads the text file and prints the frequency of each word.

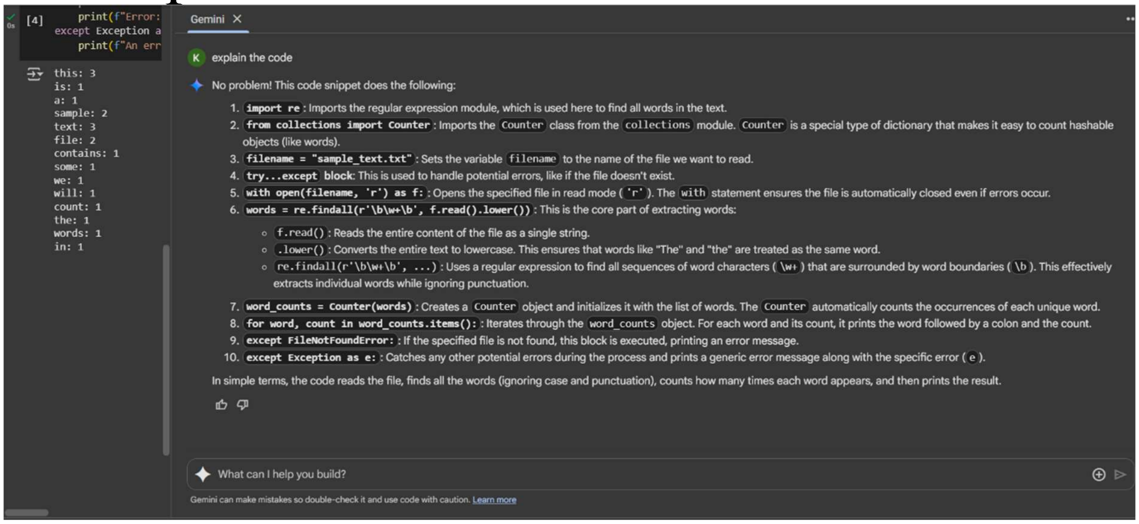
Would you like to try this with a different text file? Or perhaps you'd like to explore sorting the words by frequency, or visualizing the word counts?



```
this: 2
is: 2
a: 1
sample: 2
text: 2
file: 3
for: 1
demonstrating: 1
word: 1
frequency: 1
counting: 1
```

❖

❖ Explanation of code:



```
[4]     print(f"Error:
    except Exception a
        print(f"An err
```

```
this: 3
is: 1
a: 1
sample: 2
text: 3
file: 2
contains: 1
some: 1
we: 1
will: 1
count: 1
the: 1
words: 1
in: 1
```

explain the code

No problem! This code snippet does the following:

1. `import re`: Imports the regular expression module, which is used here to find all words in the text.
2. `from collections import Counter`: Imports the `Counter` class from the `collections` module. `Counter` is a special type of dictionary that makes it easy to count hashable objects (like words).
3. `filename = "sample_text.txt"`: Sets the variable `filename` to the name of the file we want to read.
4. `try...except` **block**: This is used to handle potential errors, like if the file doesn't exist.
5. `with open(filename, 'r') as f:`: Opens the specified file in read mode (`'r'`). The `with` statement ensures the file is automatically closed even if errors occur.
6. `words = re.findall(r'\b\w+\b', f.read().lower())`: This is the core part of extracting words:
    - `f.read()`: Reads the entire content of the file as a single string.
    - `.lower()`: Converts the entire text to lowercase. This ensures that words like "The" and "the" are treated as the same word.
    - `re.findall(r'\b\w+\b', ...)`: Uses a regular expression to find all sequences of word characters (`\w+`) that are surrounded by word boundaries (`\b`). This effectively extracts individual words while ignoring punctuation.
7. `word_counts = Counter(words)`: Creates a `Counter` object and initializes it with the list of words. The `Counter` automatically counts the occurrences of each unique word.
8. `for word, count in word_counts.items():`: Iterates through the `word_counts` object. For each word and its count, it prints the word followed by a colon and the count.
9. `except FileNotFoundError:`: If the specified file is not found, this block is executed, printing an error message.
10. `except Exception as e:`: Catches any other potential errors during the process and prints a generic error message along with the specific error (`e`).

In simple terms, the code reads the file, finds all the words (ignoring case and punctuation), counts how many times each word appears, and then prints the result.

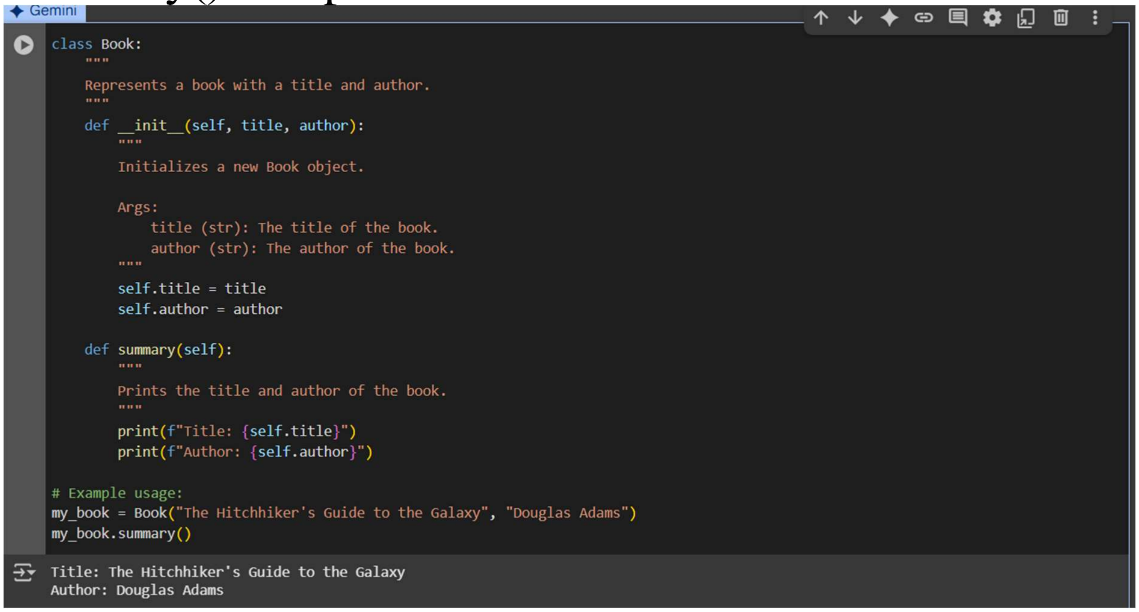
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more

❖

Task3:

❖Description: Use cursor AI to generate a python class called Book with attributes, title, author, and a method summary() that prints the details.

❖Prompt: Generate a python code of class called Book with attributes, title, author, and a method summary() that prints the details.



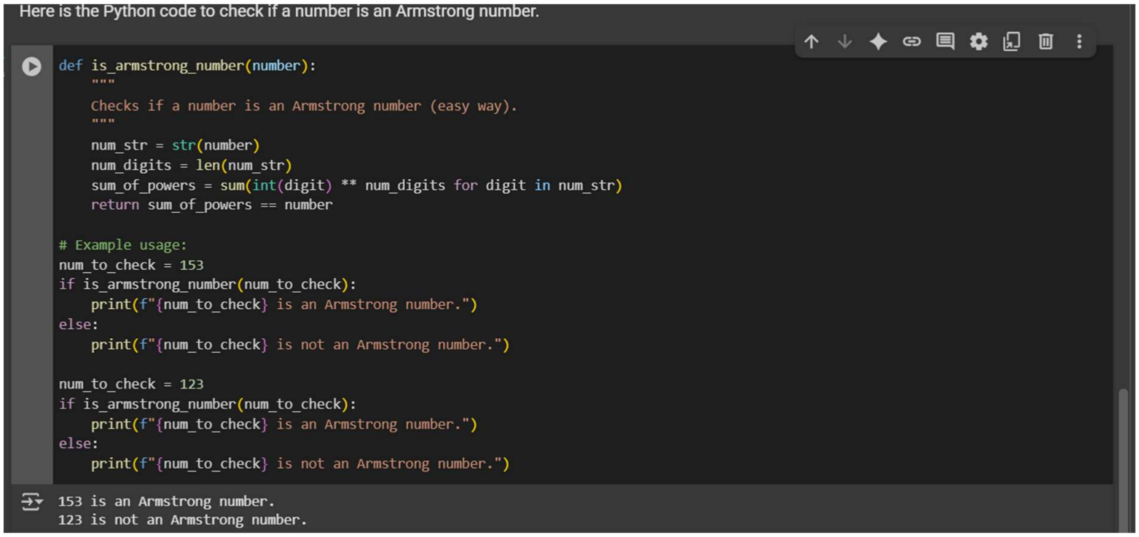
```
class Book:
    """
    Represents a book with a title and author.
    """
    def __init__(self, title, author):
        """
        Initializes a new Book object.

        Args:
            title (str): The title of the book.
            author (str): The author of the book.
        """
        self.title = title
        self.author = author

    def summary(self):
        """
        Prints the title and author of the book.
        """
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")

# Example usage:
my_book = Book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams")
my_book.summary()
```

```
Title: The Hitchhiker's Guide to the Galaxy
Author: Douglas Adams
```

- **Observation**: Cursor AI correctly uses the class keyword to define the book class, following python object-oriented programming conventions. The _init_() method is implemented. The code assigns self.title and self.author properly.

Task4:

- **Description:** Ask the 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.
- **Prompt:** Generate the python code to check whether a number is Armstrong number.



Here is the Python code to check if a number is an Armstrong number.

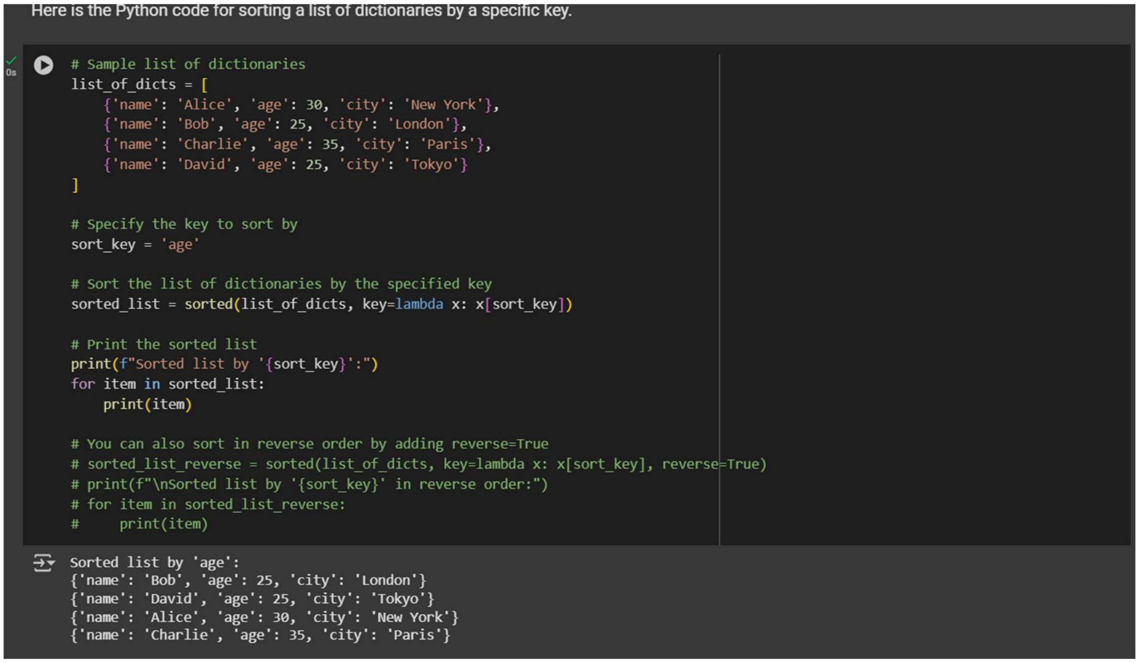
```python
def is_armstrong_number(number):
    """
    Checks if a number is an Armstrong number (easy way).
    """
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number

# 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.")
```

```
153 is an Armstrong number.
123 is not an Armstrong number.
```

- **Observation**: Gemini included sample input like 153 or 9474 and show that code correctly identifies them as Armstrong numbers. Raising each digit to the power of the number of digit.

Task5:

- **Description:** Use the both Gemini and Cursor AI to generate code for sorting a list of dictionaries by a specific key (e.g., age).
- **Prompt:** Generate a python code for sorting a list of dictionaries by a specific key
- **Gemini Code:**

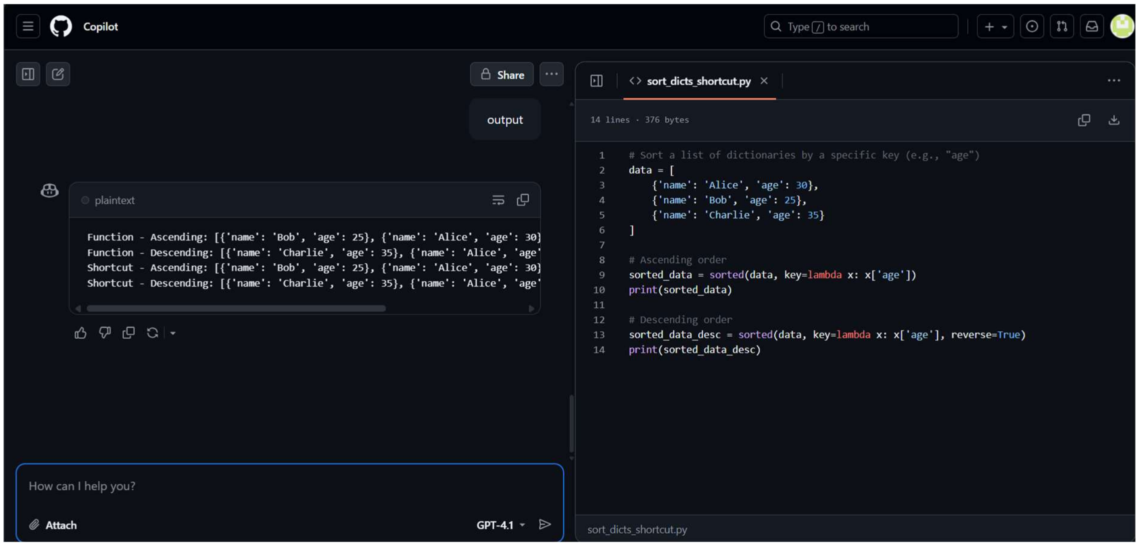


Here is the Python code for sorting a list of dictionaries by a specific key.

```python
# Sample list of dictionaries
list_of_dicts = [
    {'name': 'Alice', 'age': 30, 'city': 'New York'},
    {'name': 'Bob', 'age': 25, 'city': 'London'},
    {'name': 'Charlie', 'age': 35, 'city': 'Paris'},
    {'name': 'David', 'age': 25, 'city': 'Tokyo'}
]

# Specify the key to sort by
sort_key = 'age'

# Sort the list of dictionaries by the specified key
sorted_list = sorted(list_of_dicts, key=lambda x: x[sort_key])

# Print the sorted list
print(f"Sorted list by '{sort_key}':")
for item in sorted_list:
    print(item)

# You can also sort in reverse order by adding reverse=True
# sorted_list_reverse = sorted(list_of_dicts, key=lambda x: x[sort_key], reverse=True)
# print(f"\nSorted list by '{sort_key}' in reverse order:")
# for item in sorted_list_reverse:
#     print(item)
```

```
Sorted list by 'age':
{'name': 'Bob', 'age': 25, 'city': 'London'}
{'name': 'David', 'age': 25, 'city': 'Tokyo'}
{'name': 'Alice', 'age': 30, 'city': 'New York'}
{'name': 'Charlie', 'age': 35, 'city': 'Paris'}
```

❖ Cursor Code:



```
Function - Ascending: [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}
Function - Descending: [{'name': 'Charlie', 'age': 35}, {'name': 'Alice', 'age'
Shortcut - Ascending: [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}
Shortcut - Descending: [{'name': 'Charlie', 'age': 35}, {'name': 'Alice', 'age'
```

```python
# Sort a list of dictionaries by a specific key (e.g., "age")
data = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]

# Ascending order
sorted_data = sorted(data, key=lambda x: x['age'])
print(sorted_data)

# Descending order
sorted_data_desc = sorted(data, key=lambda x: x['age'], reverse=True)
print(sorted_data_desc)
```

❖ **Observation**: Gemini typically uses python bulit-in sorted() function with a key argument like lambda x:x[‘age’] , which is the standard and efficient way to sort.

Copilot sorts a list of dictionaries by the key "age".In ascending order: youngest to oldest.In descending order: oldest to youngest.Uses sorted() with a lambda to pick the "age" value.