**ASSIGNMENT-02**

**NAME:**Shah Alam

**Hall Ticket:**2303A510C1

**Batch:**06

**Q)** Task 1: Word Frequency from Text File

* Scenario:

You are analyzing log files for keyword frequency.

* Task:

Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.

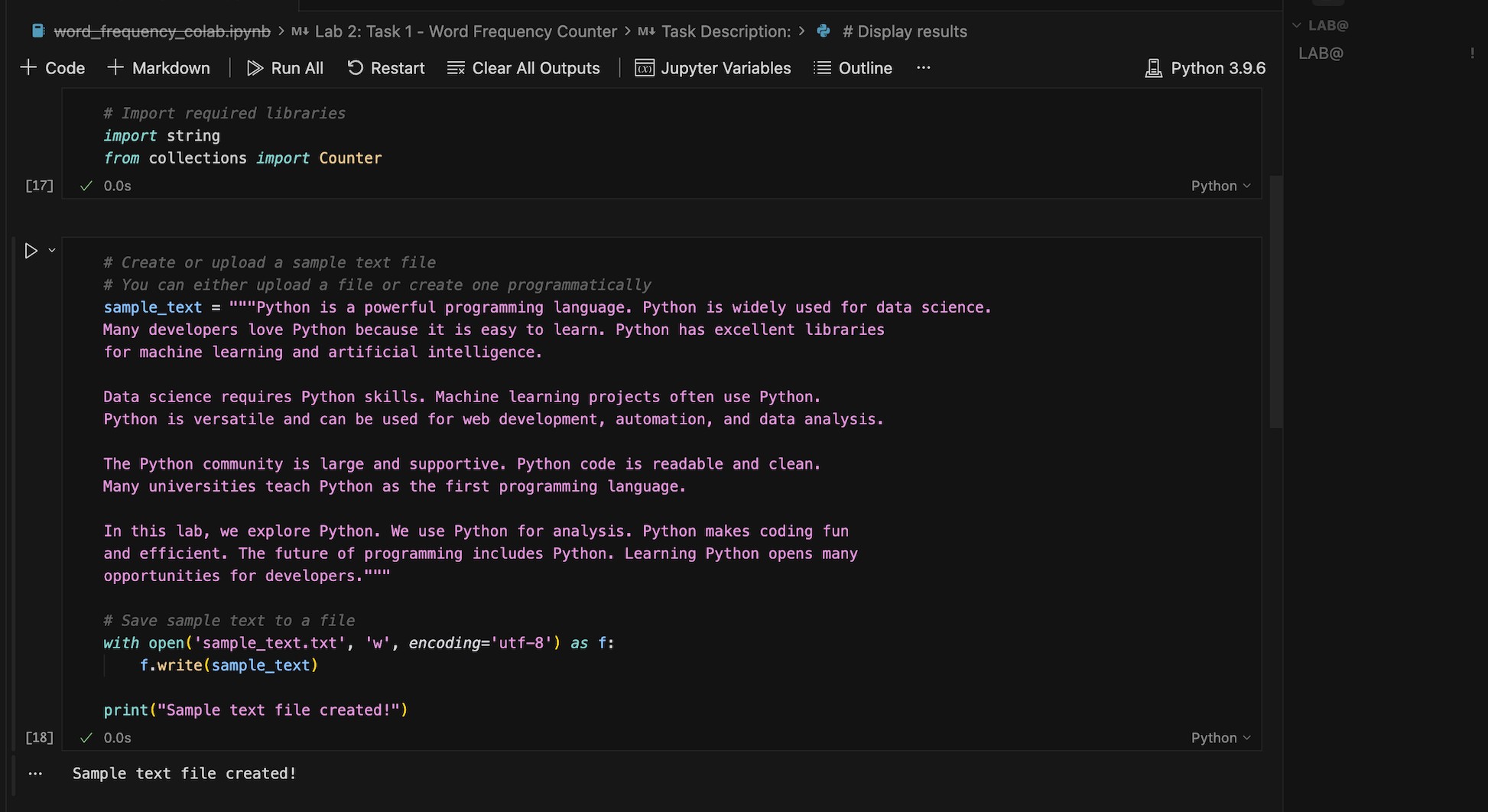
* Expected Output:
* Working code
* Explanation ➢Screenshot

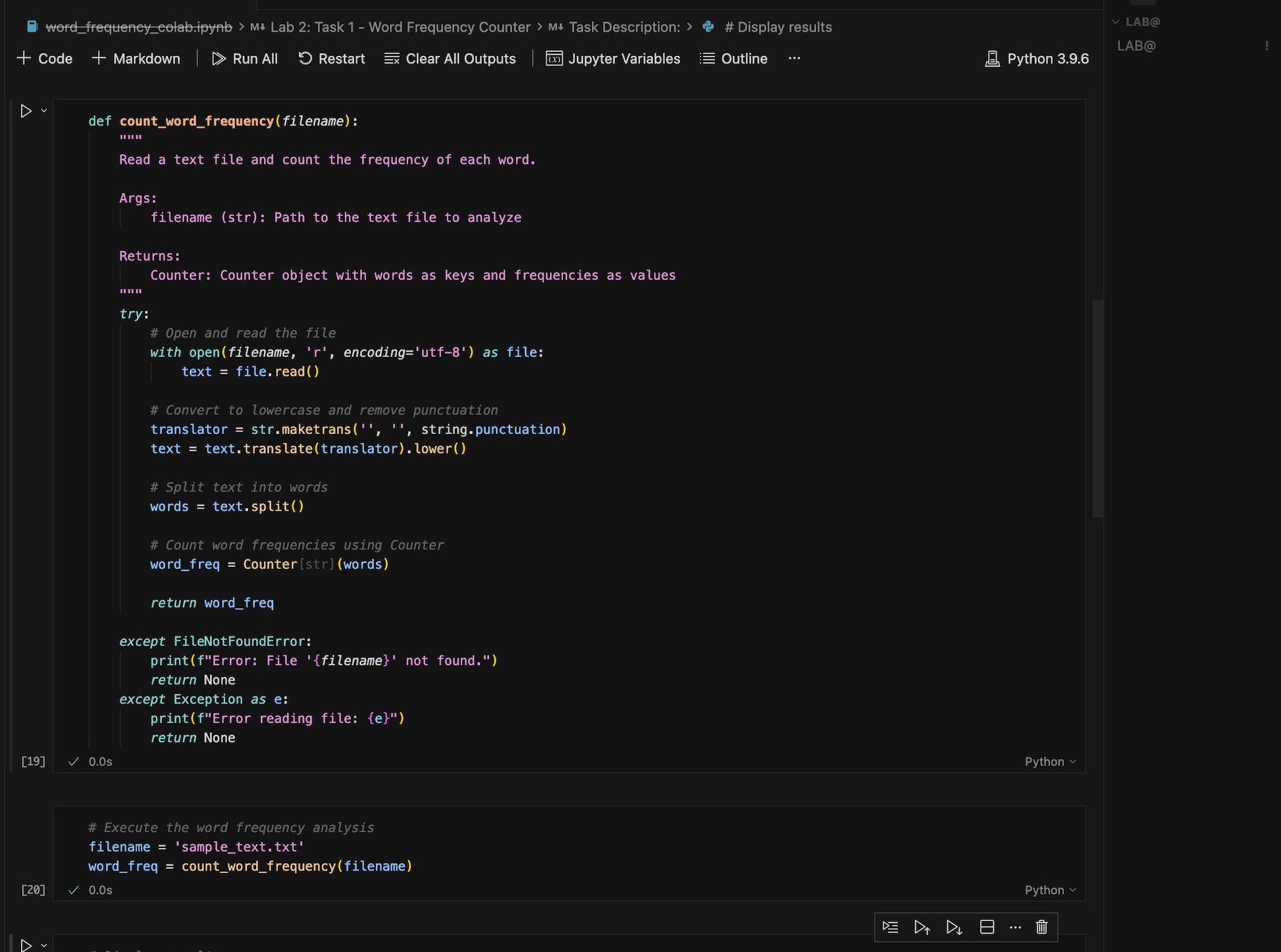
**Solution:**

**PROMPT**

Generate a Python program in Google Colab that reads a text file and counts the frequency of each word.

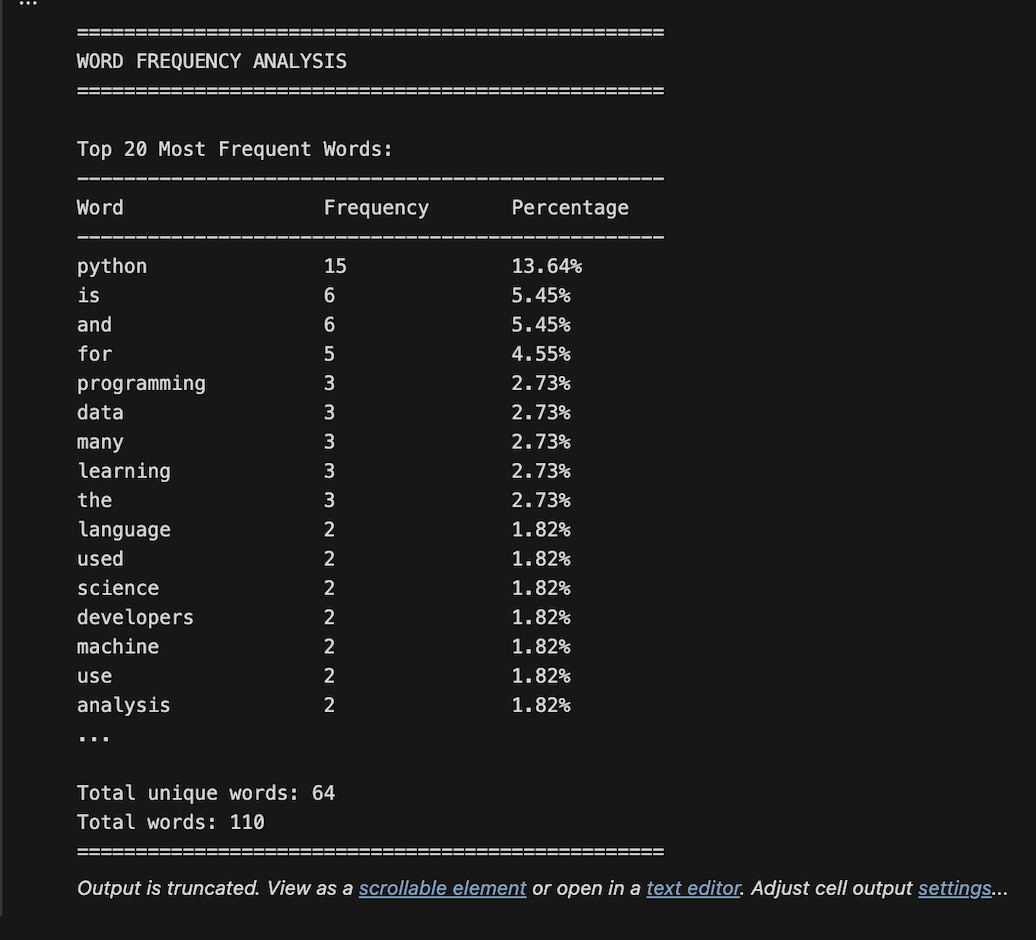
**CODE:**







**OUTPUT:**



**CODE Explanation:**

This Python program works by first importing the required modules to handle punctuation removal and word counting. The text file is opened in read mode and its content is read completely. Then, all punctuation marks are removed and the text is converted to lowercase so that words are counted correctly without case differences. After that, the text is split into individual words. The Counter function is used to count the number of times each word appears in the file. The program also includes error handling to display a message if the file is not found or if any other error occurs. Finally, the word frequencies are displayed in an organized format, making the output easy to understand

Q) Task 2: File Operations Using Cursor AI ❖ Scenario:

You are automating basic file operations.

❖ Task:

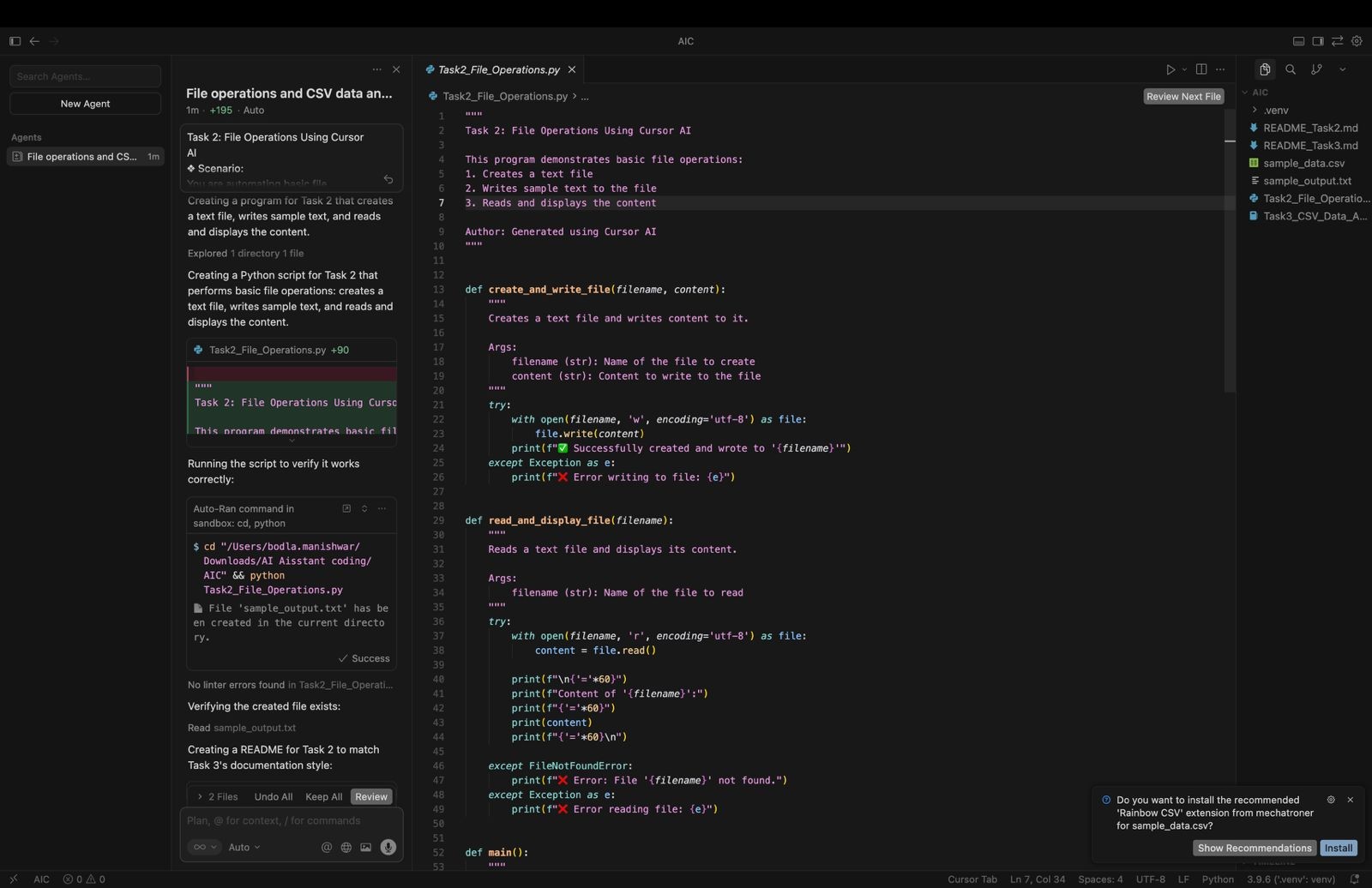
Use Cursor AI to generate a program that:

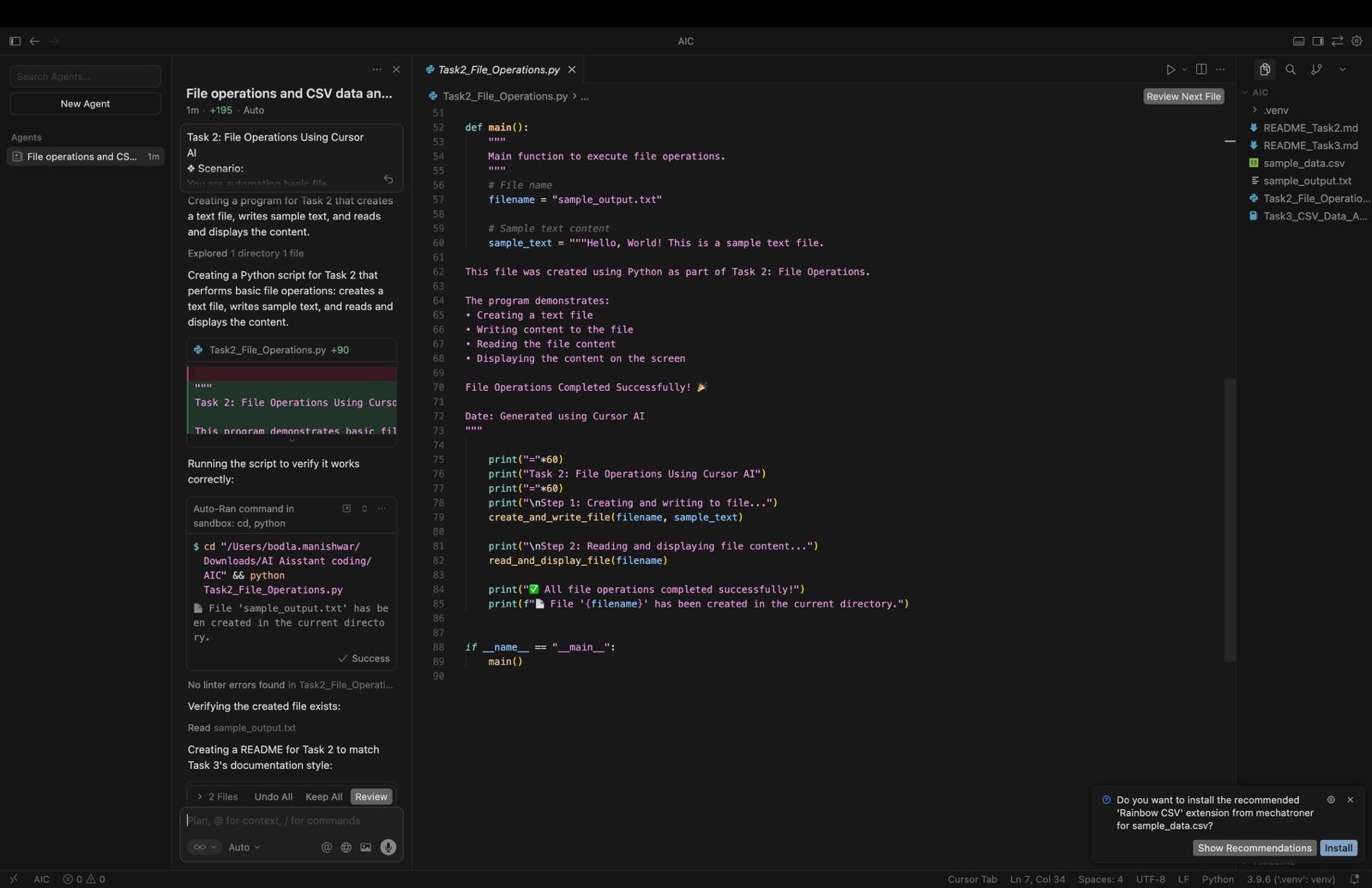
* Creates a text file
* Writes sample text
* Reads and displays the content❖ Expected Output:
* Functional code➢Cursor AI screenshots

**PROMPT:**

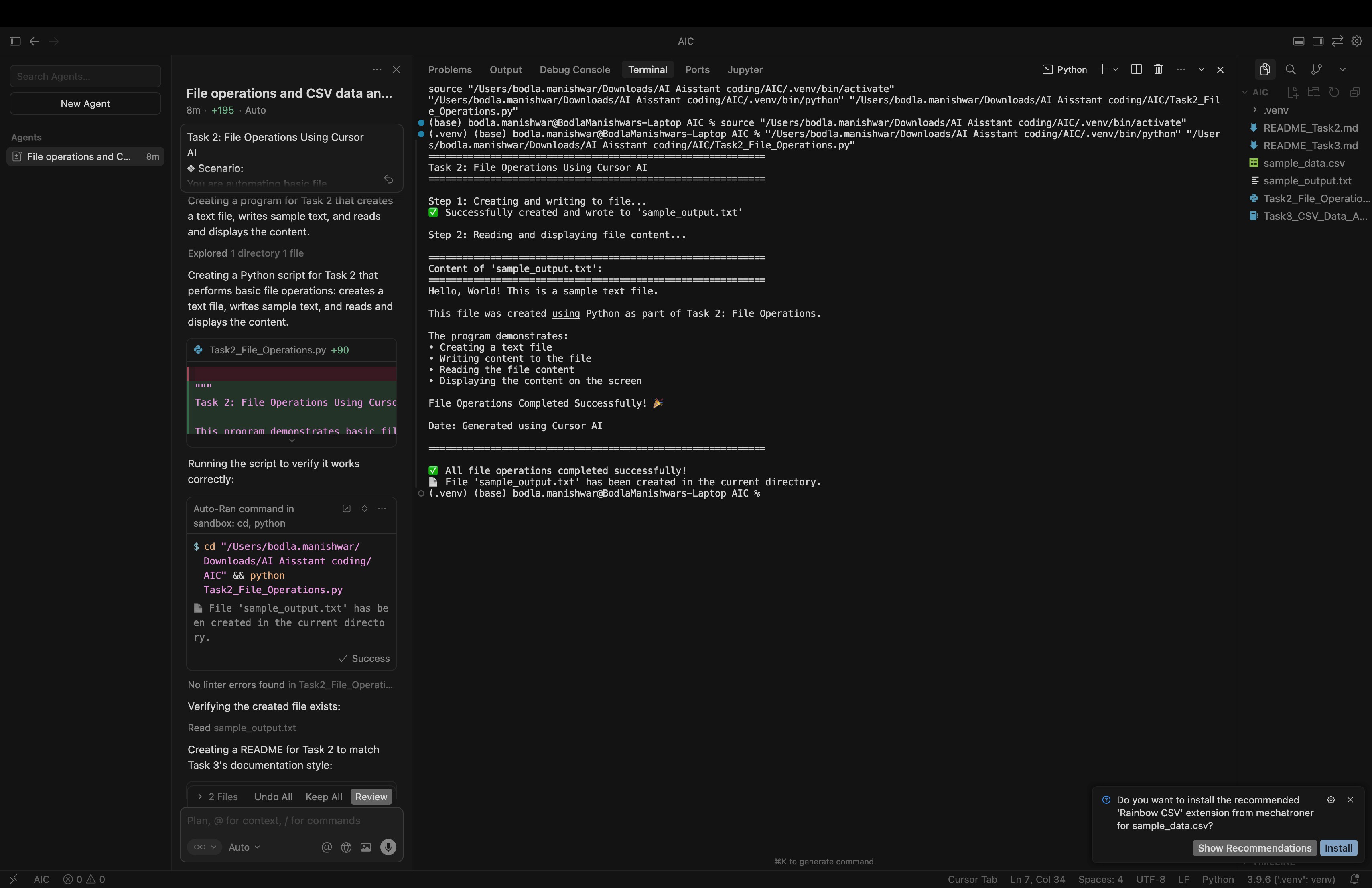
Generate a simple Python program that demonstrates basic file operations. The program should create a text file, write some sample text into it, then read the content from the file and display it on the screen.

**CODE:**





**OUTPUT:**



**CODE EXPLANATION:**

This Python program demonstrates basic file operations by creating a text file, writing sample content to it, and then reading and displaying that content on the screen. It uses separate functions for writing and reading files to keep the code organized and clear. The program also includes exception handling to manage errors such as file access issues, ensuring smooth execution. The main() function controls the overall flow, and the program runs only when executed directly, making it a simple and effective example of file handling in Python.

**Q)Task 3: CSV Data Analysis** ❖ **Scenario:**

**You are processing structured data from a CSV file.**

* **Task:**

**Use Gemini in Colab to read a CSV file and calculate mean, min, and max.**

* **Expected Output:**

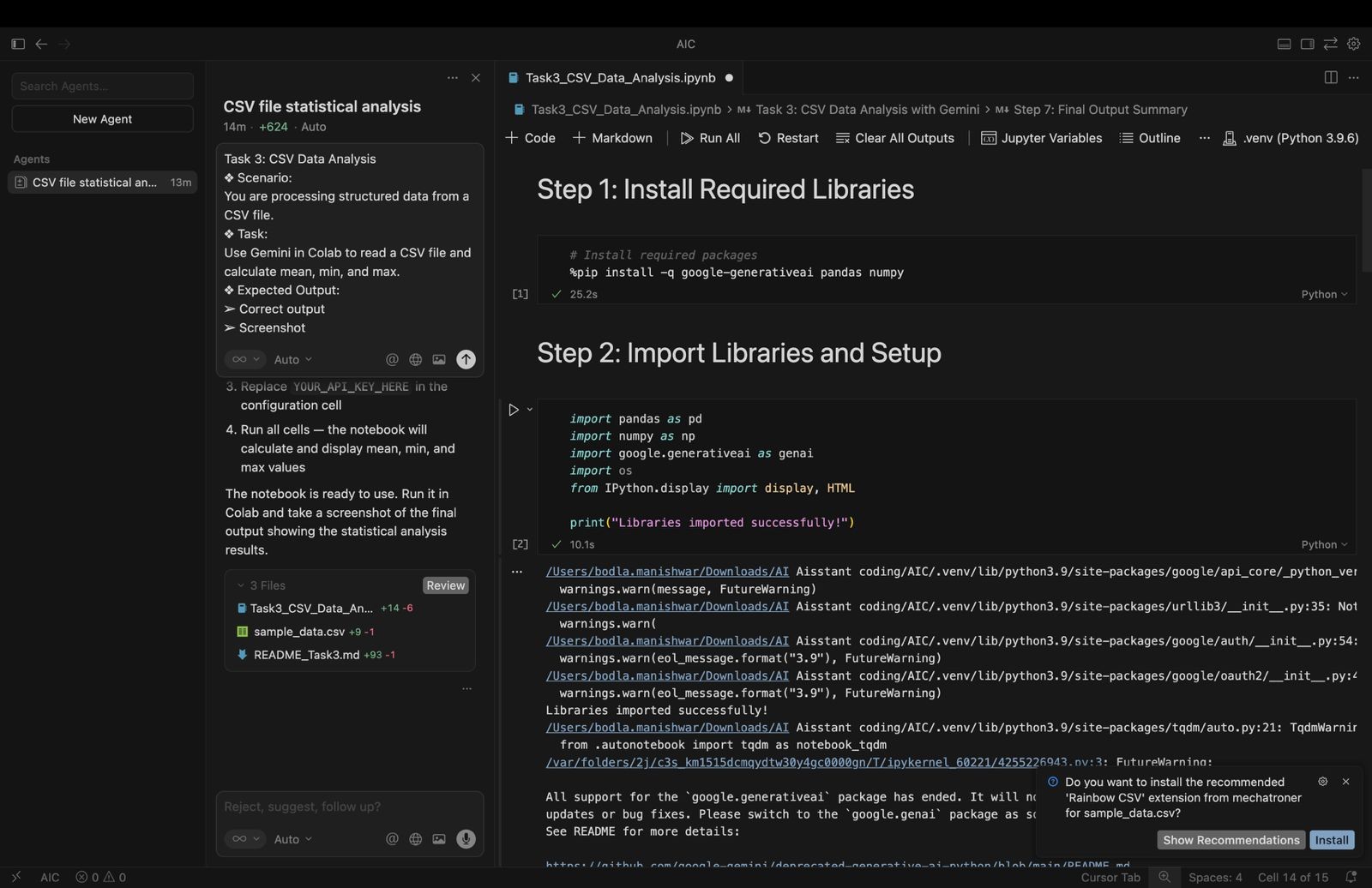
➢ **Correct output**

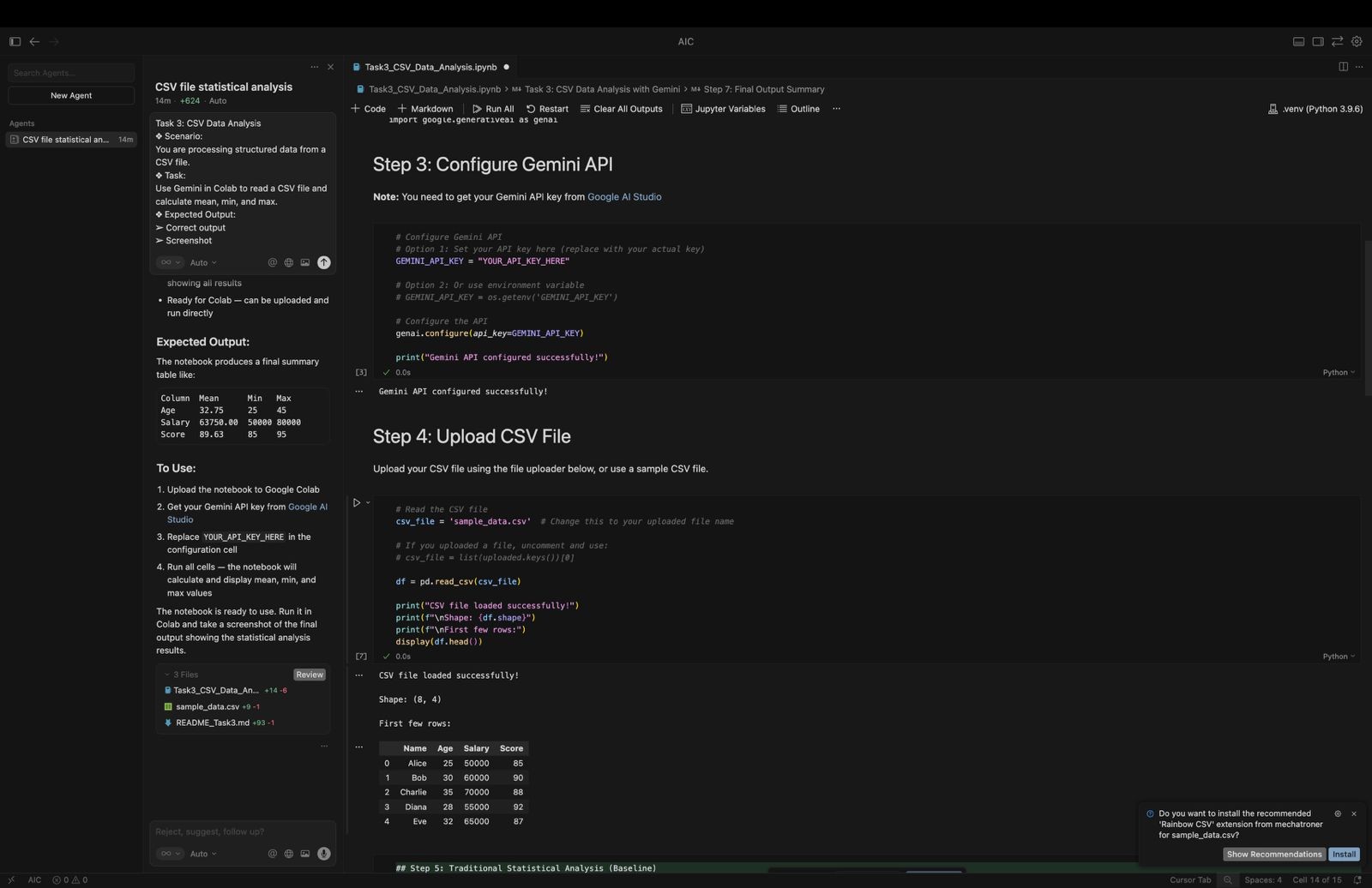
➢**Screenshot**

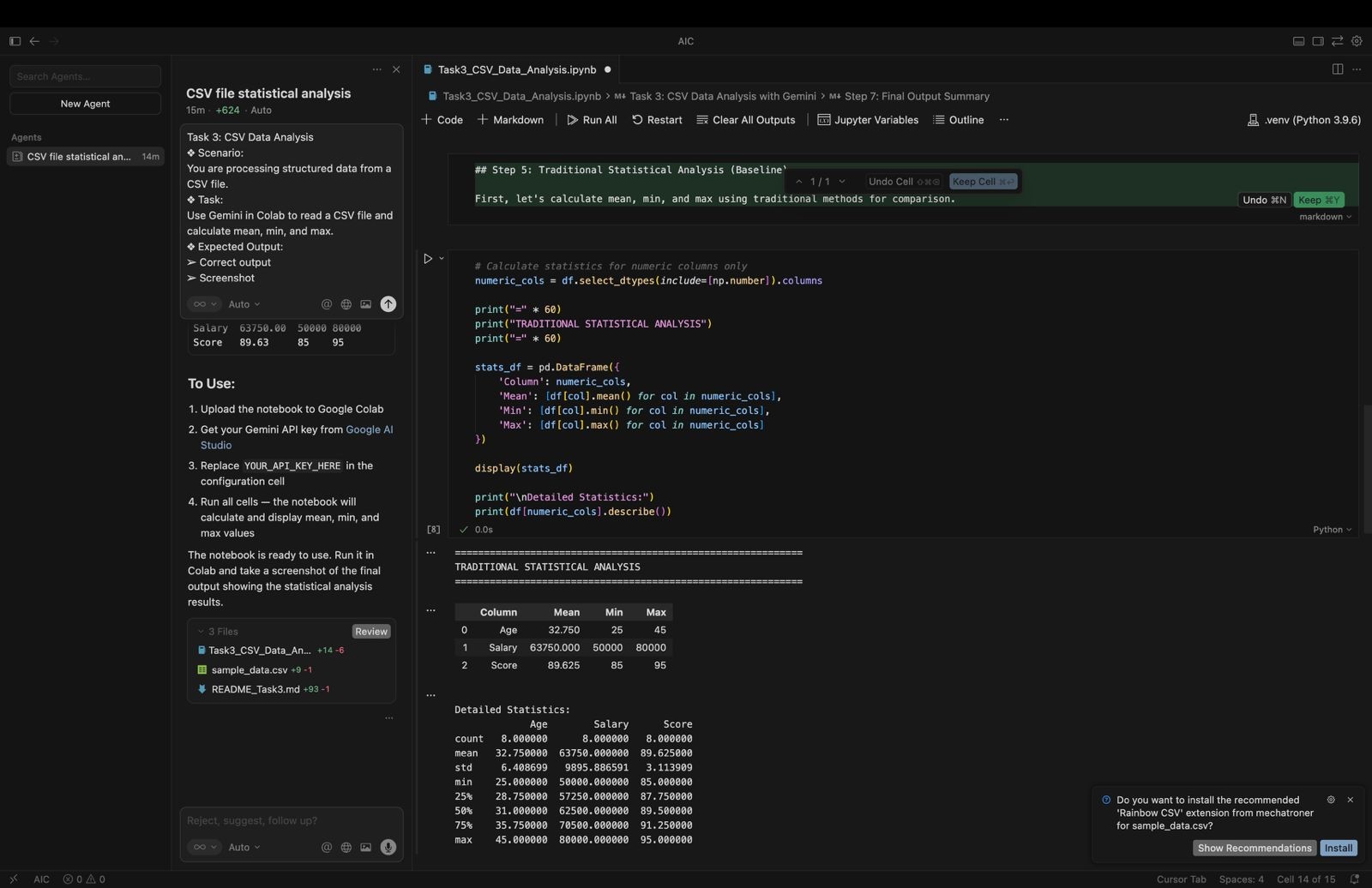
**PROMPT:**

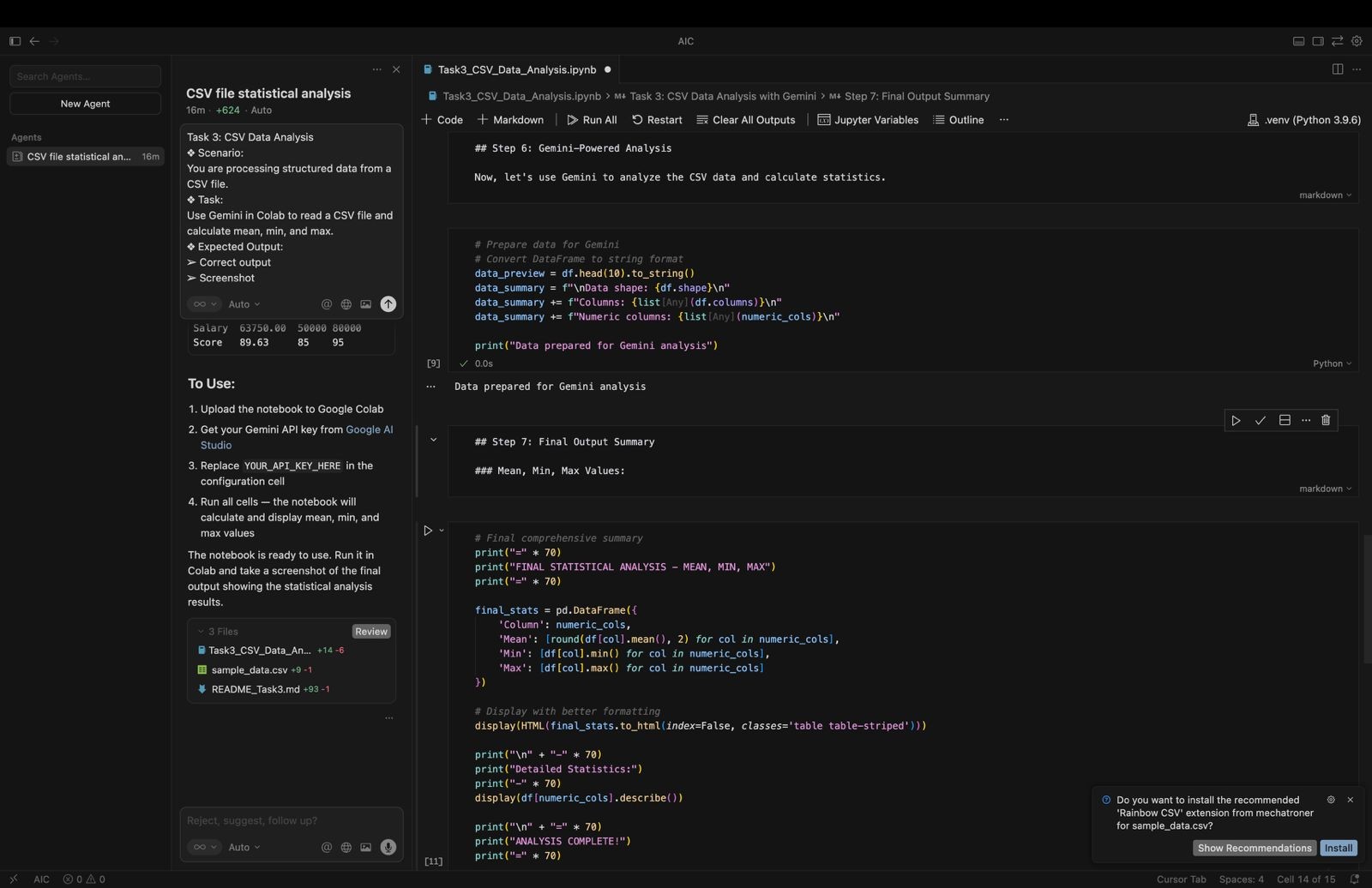
**Write Python code in Google Colab to read a CSV file and calculate mean, minimum, and maximum values using pandas.**

**CODE:**

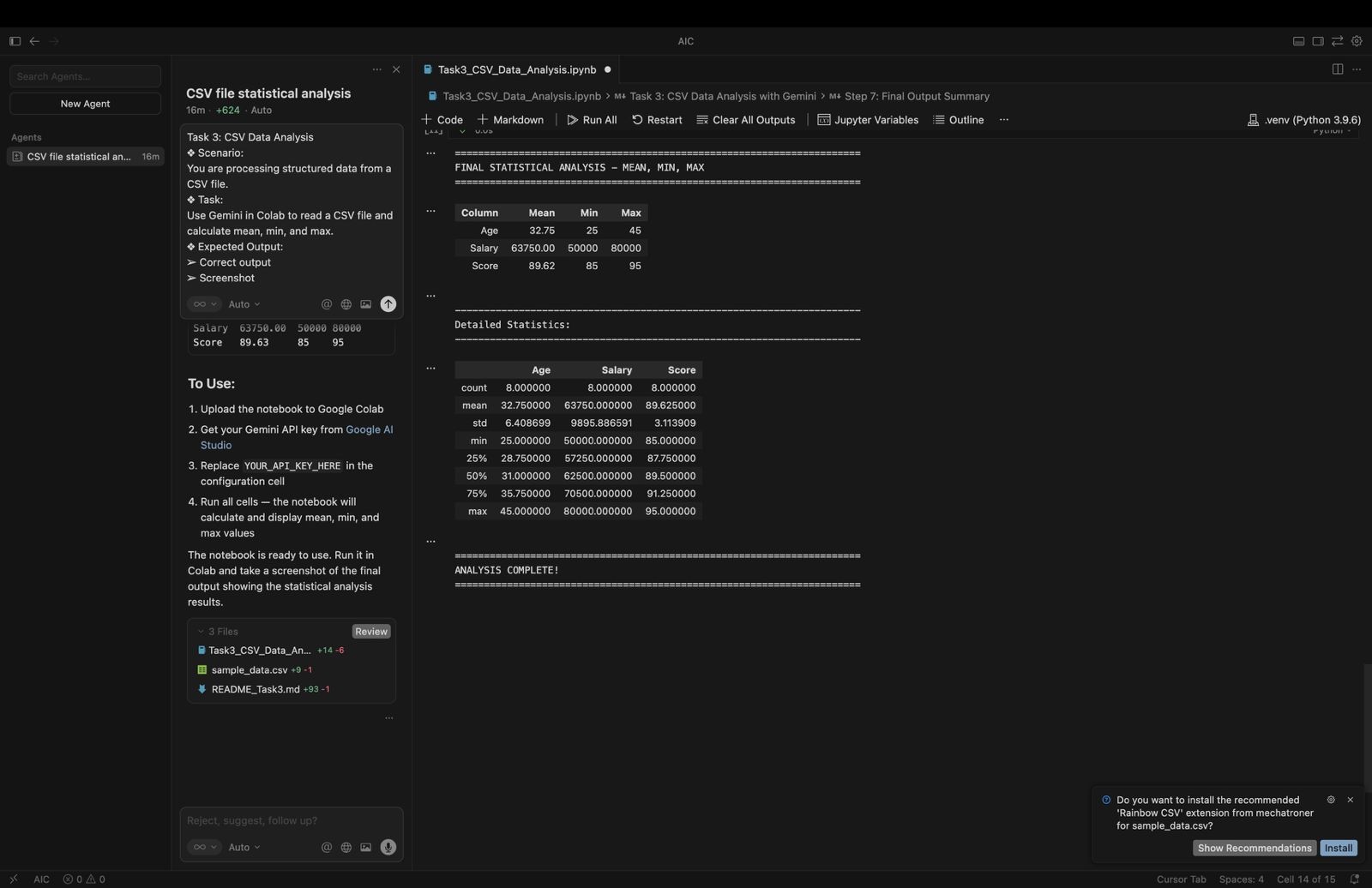








**OUTPUT:**



**CODE EXPLANATION:**

This code performs statistical analysis on numeric columns of a DataFrame (df). First, it identifies all columns that contain numerical data using select\_dtypes(include=[np.number]). Then, for each numeric column, it calculates the mean, minimum, and maximum values and stores them in a new DataFrame called stats\_df. This DataFrame is displayed to show a clean summary of basic statistics.

**Q)Task 4: Sorting Lists – Manual vs Built-in** ❖ **Scenario:**

**You are reviewing algorithm choices for efficiency.**

❖ **Task:**

**Use Gemini to generate:**

* **Bubble sort**
* **Python’s built-in sort()**
* **Compare both implementations.**

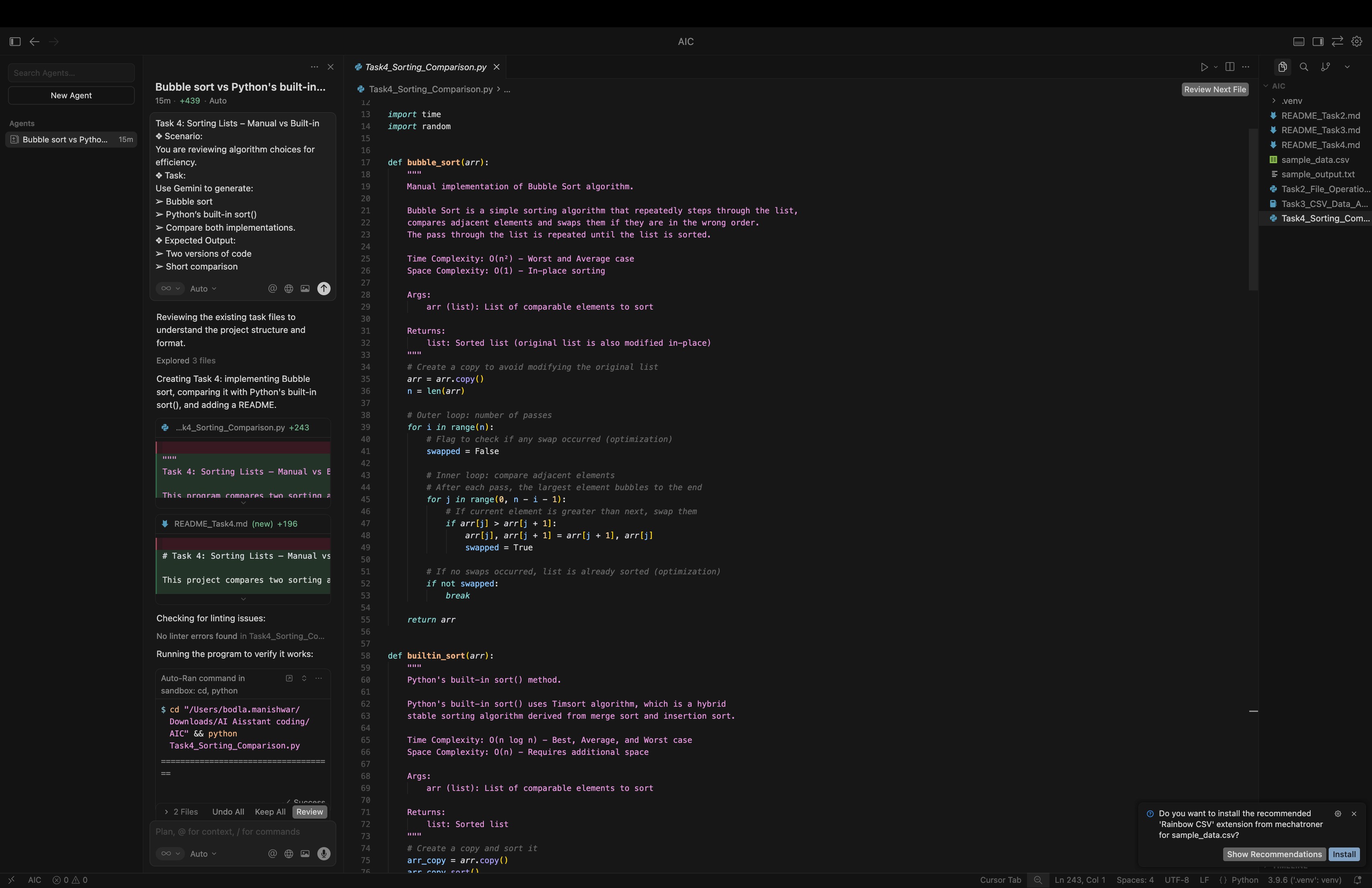
❖ **Expected Output:**

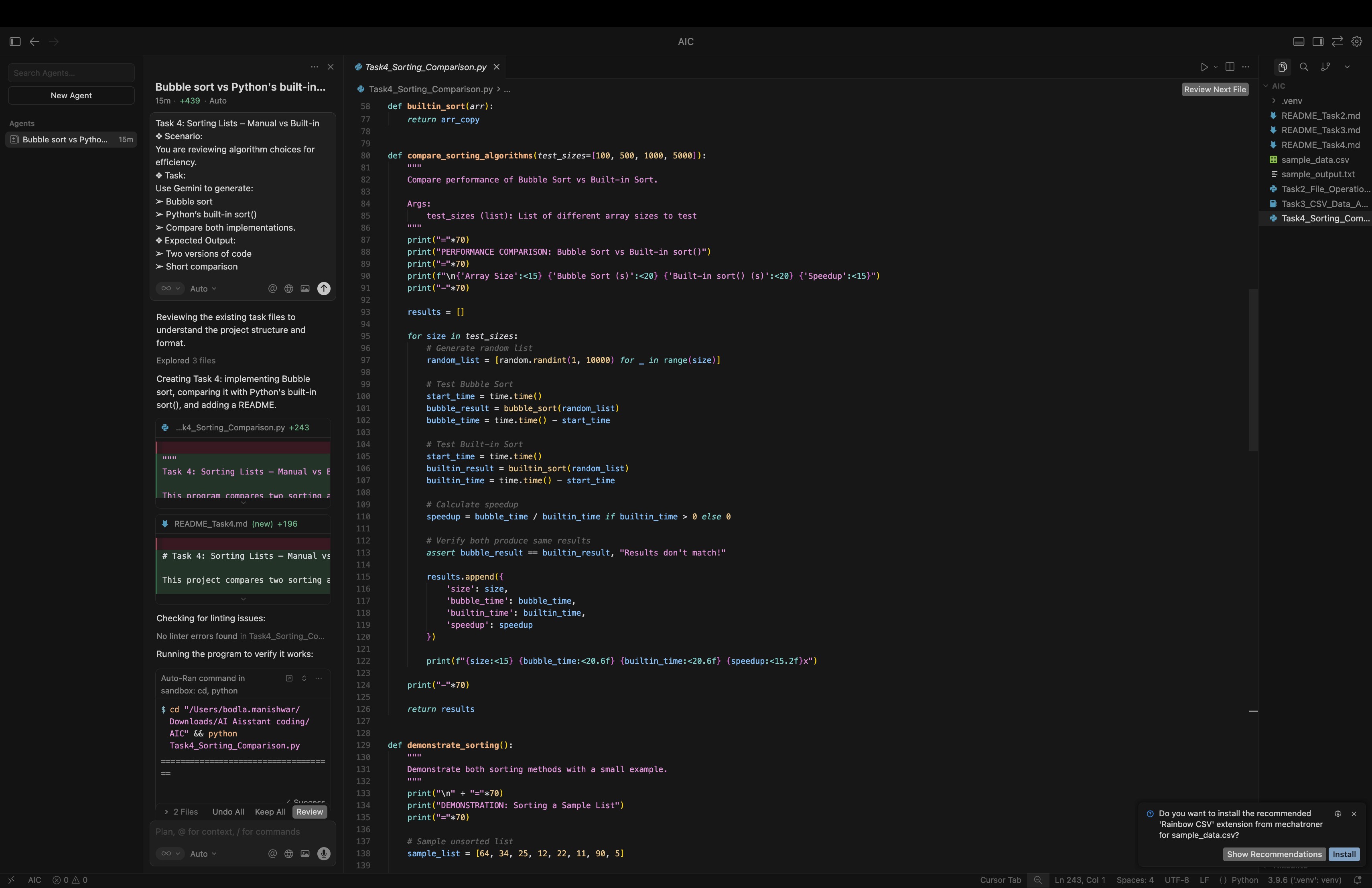
➢ **Two versions of code** ➢**Short comparison**

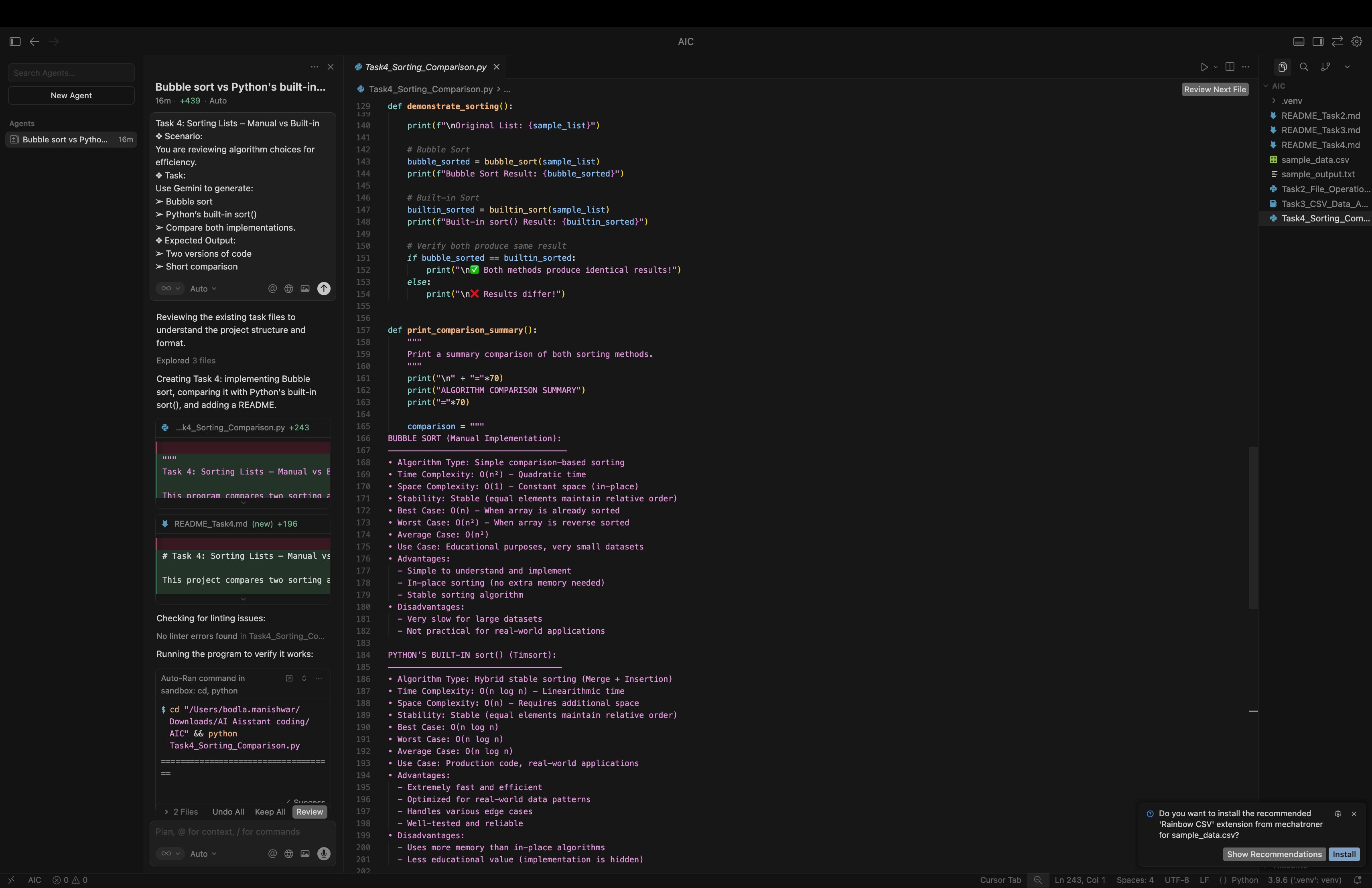
**PROMPT:**

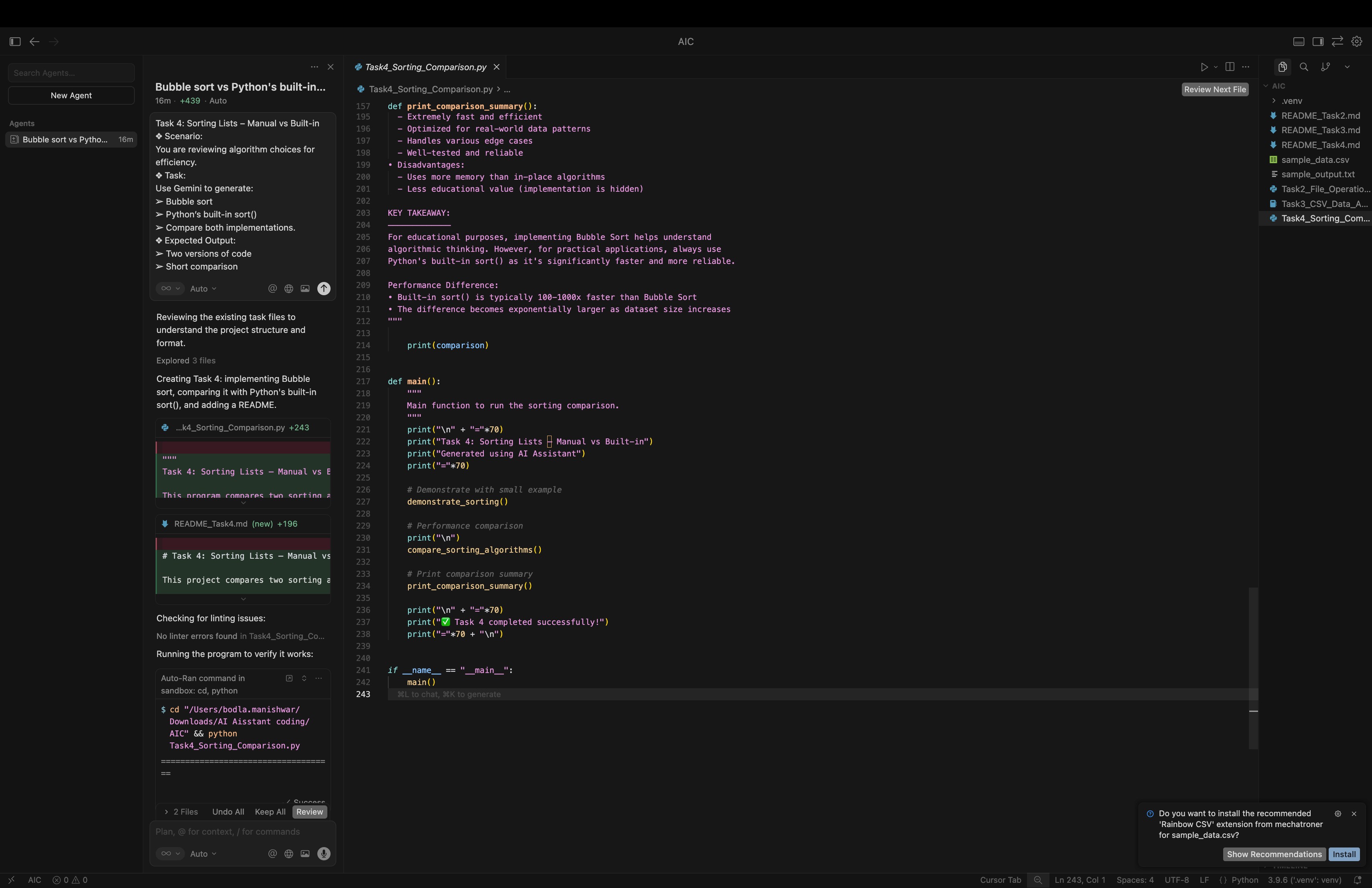
Generate Python code to sort a list using bubble sort and Python’s built-in sort() method. Show both implementations clearly and provide a short comparison explaining their efficiency and usage.

**CODE:**

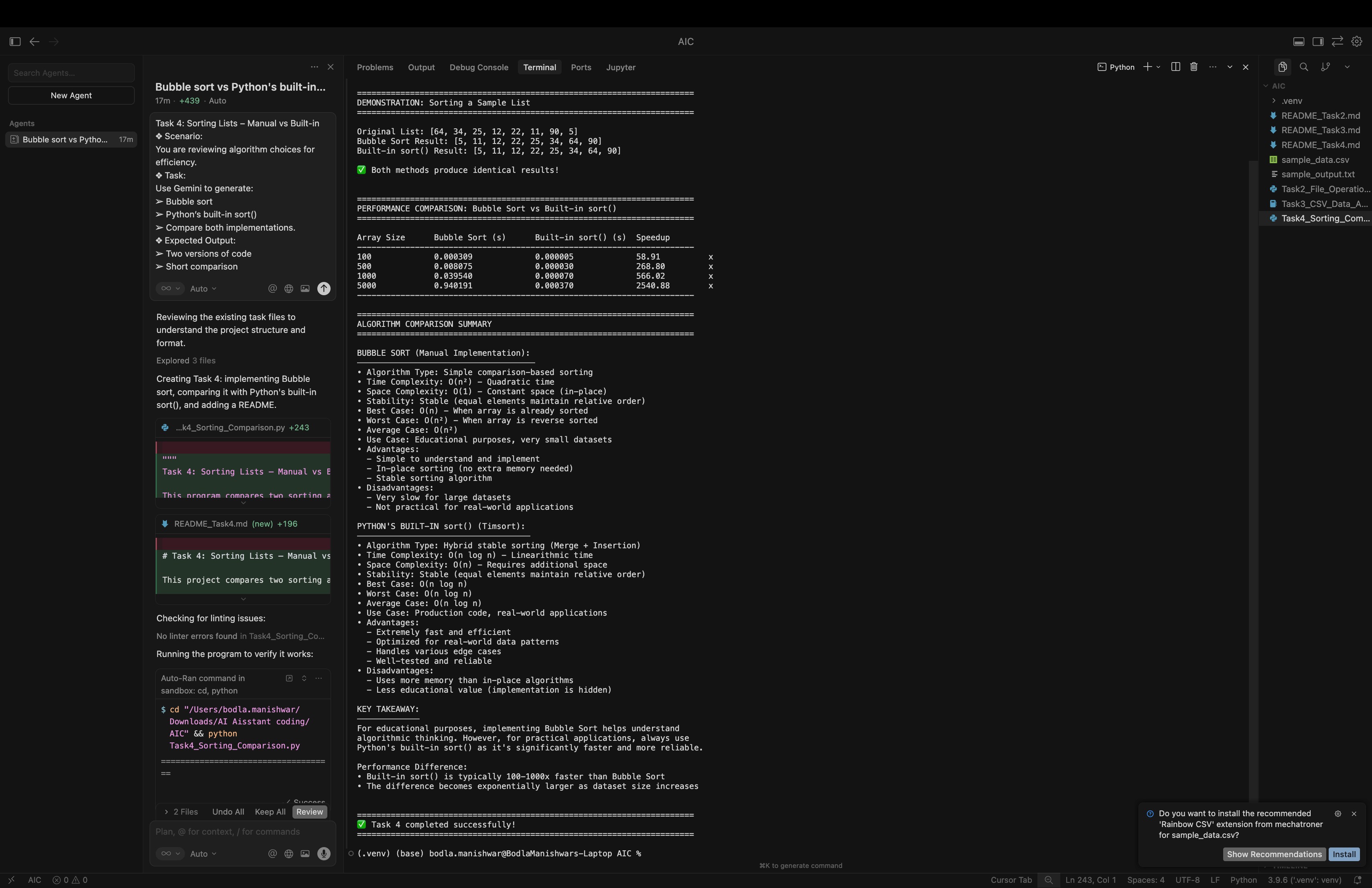








**OUTPUT:**



**CODE EXPLANATION:**

This program compares Bubble Sort and Python’s built-in sort(). Bubble Sort manually compares and swaps elements to arrange them in order, but it is slow for large lists because it has O(n²) time complexity. Python’s built-in sort() uses an efficient algorithm and sorts data much faster with O(n log n) time complexity. The program measures execution time for both methods and shows that the built-in sort is much faster and more suitable for real-world use.