

## ASSIGNMENT-02

**NAME:**M.Shiva Shankar

**Hall Ticket:**2303A51294

**Batch:**05

### 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

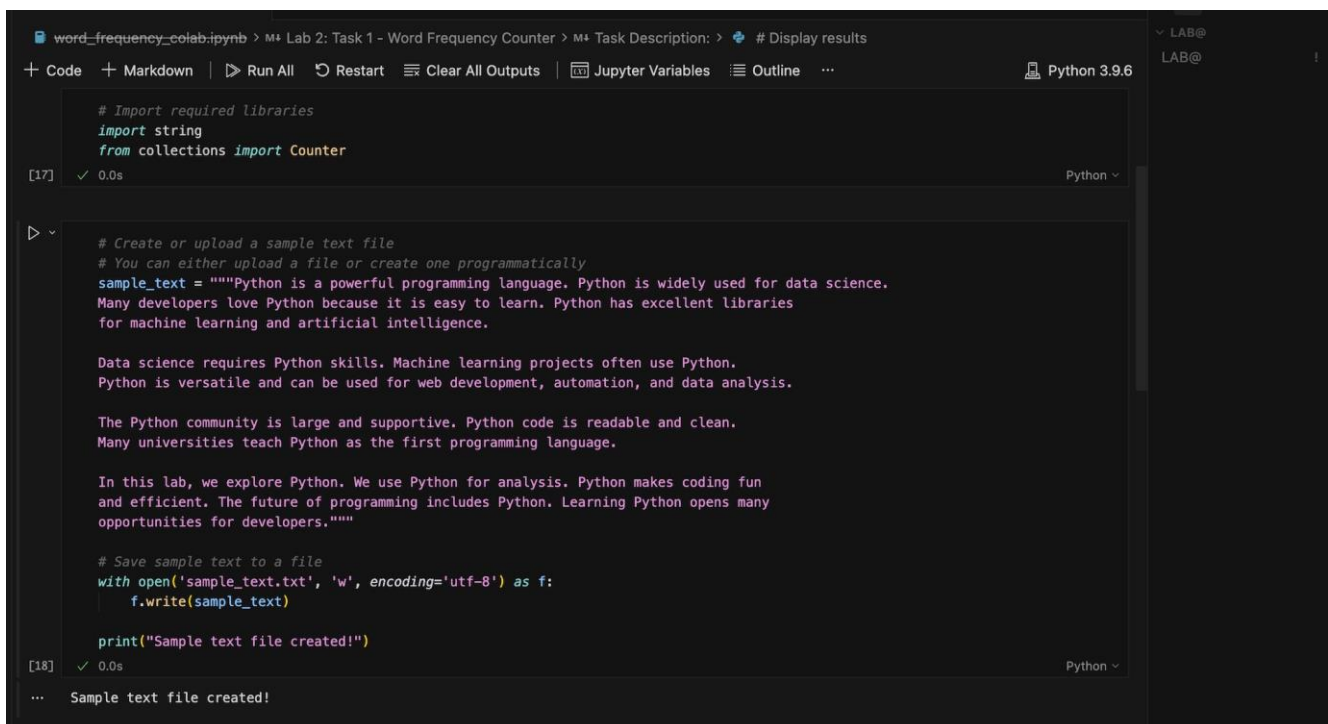
➤ Explanation ➤ Screenshot

### Solution:

### PROMPT

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

### CODE:



```
word_frequency_colab.ipynb > M+ Lab 2: Task 1 - Word Frequency Counter > M+ Task Description: > # Display results
+ Code + Markdown | ▶ Run All ⏮ Restart ⚙ Clear All Outputs | Jupyter Variables Outline ... Python 3.9.6

# Import required libraries
import string
from collections import Counter

[17] ✓ 0.0s Python ▾

# Create or upload a sample text file
# You can either upload a file or create one programmatically
sample_text = """Python is a powerful programming language. Python is widely used for data science.
Many developers love Python because it is easy to learn. Python has excellent libraries
for machine learning and artificial intelligence.

Data science requires Python skills. Machine learning projects often use Python.
Python is versatile and can be used for web development, automation, and data analysis.

The Python community is large and supportive. Python code is readable and clean.
Many universities teach Python as the first programming language.

In this lab, we explore Python. We use Python for analysis. Python makes coding fun
and efficient. The future of programming includes Python. Learning Python opens many
opportunities for developers."""

# Save sample text to a file
with open('sample_text.txt', 'w', encoding='utf-8') as f:
    f.write(sample_text)

print("Sample text file created!")

[18] ✓ 0.0s Python ▾

... Sample text file created!
```

```
word_frequency_colab.ipynb > M* Lab 2: Task 1 - Word Frequency Counter > M* Task Description: > # Display results
+ Code + Markdown | ▶ Run All ⏮ Restart ⏮ Clear All Outputs | 📄 Jupyter Variables 📄 Outline ... Python 3.9.6
LAB@
LAB@

def count_word_frequency(filename):
    """
    Read a text file and count the frequency of each word.

    Args:
        filename (str): Path to the text file to analyze

    Returns:
        Counter: Counter object with words as keys and frequencies as values
    """
    try:
        # Open and read the file
        with open(filename, 'r', encoding='utf-8') as file:
            text = file.read()

        # Convert to lowercase and remove punctuation
        translator = str.maketrans('', '', string.punctuation)
        text = text.translate(translator).lower()

        # Split text into words
        words = text.split()

        # Count word frequencies using Counter
        word_freq = Counter[str](words)

        return word_freq

    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        return None
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

[19] ✓ 0.0s Python ▾

# Execute the word frequency analysis
filename = 'sample_text.txt'
word_freq = count_word_frequency(filename)

[20] ✓ 0.0s Python ▾
```

```
# Display results
if word_freq:
    print("\n" + "="*50)
    print("WORD FREQUENCY ANALYSIS")
    print("="*50)

    # Display top 20 most common words
    print("\nTop 20 Most Frequent Words:")
    print("-"*50)
    print(f"{'Word':<20} {'Frequency':<15} {'Percentage':<15}")
    print("-"*50)

    total_words = sum(word_freq.values())

    for word, count in word_freq.most_common(20):
        percentage = (count / total_words) * 100
        print(f"{'word':<20} {'count':<15} {'percentage':.2f}%")

    print("-"*50)
    print(f"\nTotal unique words: {len(word_freq)}")
    print(f"Total words: {total_words}")
    print("="*50)
```

OUTPUT:

## WORD FREQUENCY ANALYSIS

### Top 20 Most Frequent Words:

Word	Frequency	Percentage
python	15	13.64%
is	6	5.45%
and	6	5.45%
for	5	4.55%
programming	3	2.73%
data	3	2.73%
many	3	2.73%
learning	3	2.73%
the	3	2.73%
language	2	1.82%
used	2	1.82%
science	2	1.82%
developers	2	1.82%
machine	2	1.82%
use	2	1.82%
analysis	2	1.82%
...		

Total unique words: 64

Total words: 110

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings...](#)

## 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:

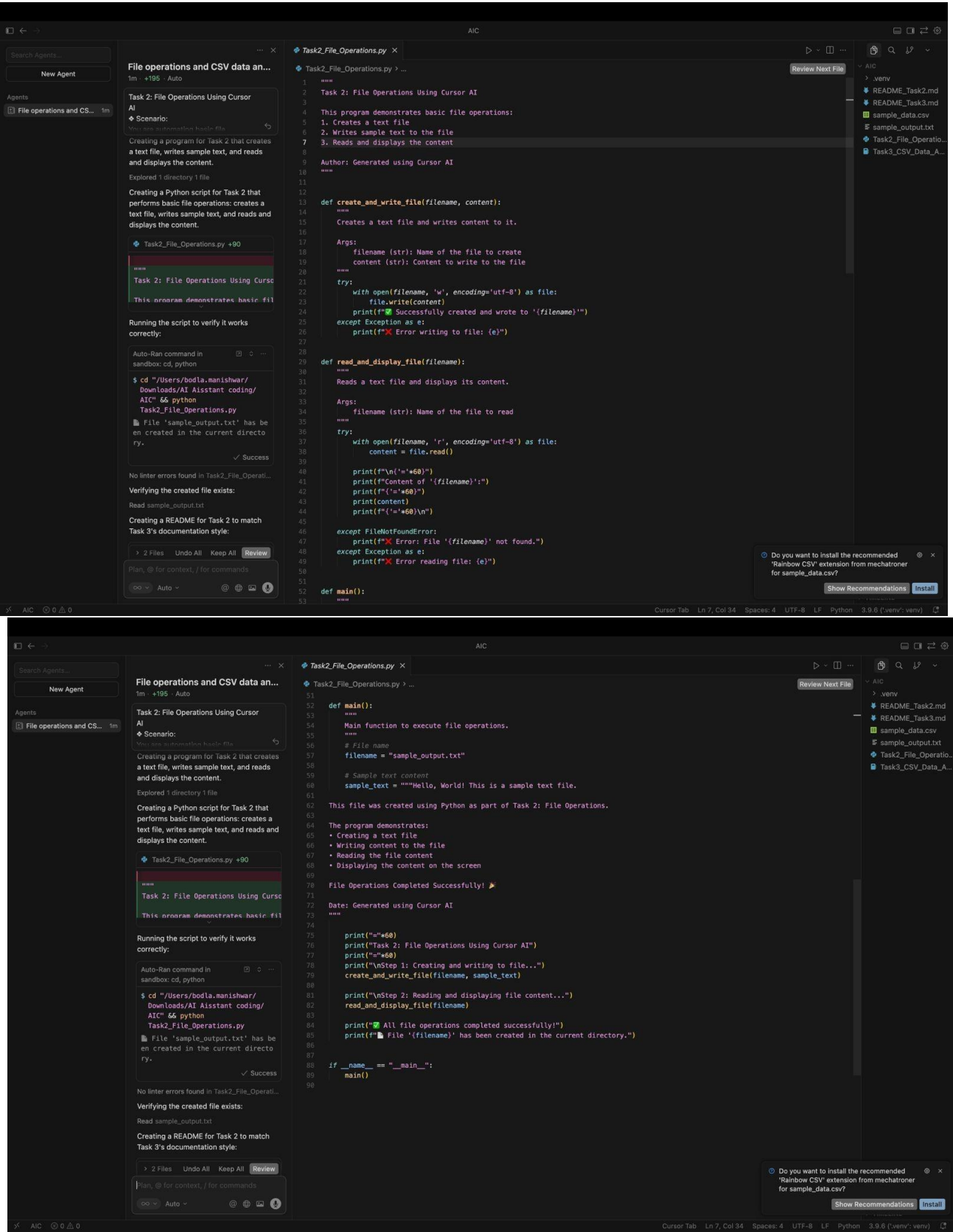
Use Cursor AI to generate a program that:

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

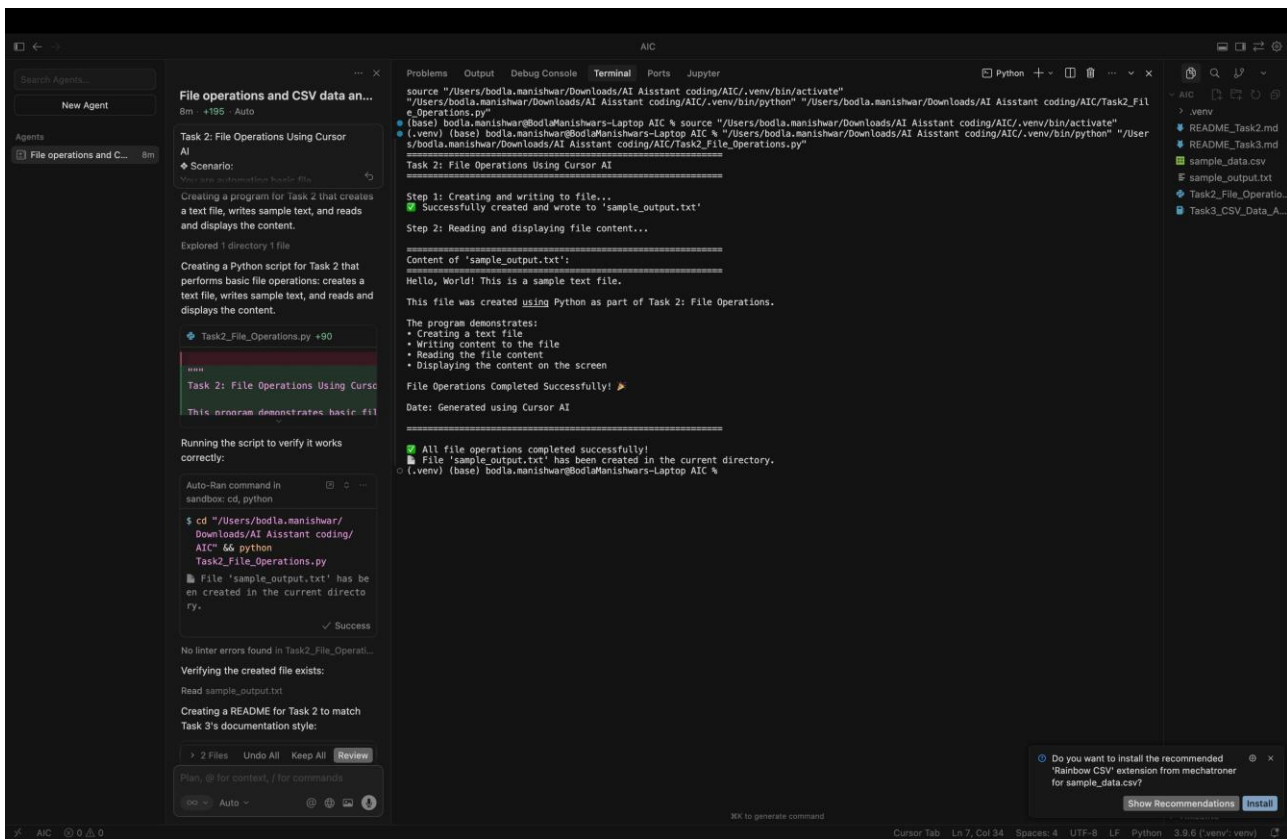
PROMPT:

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:



Search Agents...

New Agent

Agents

CSV file statistical an... 13m

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

3. Replace YOUR\_API\_KEY\_HERE in the configuration cell

4. Run all cells — the notebook will calculate and display mean, min, and max values

The notebook is ready to use. Run it in Colab and take a screenshot of the final output showing the statistical analysis results.

3 Files

Task3\_CSV\_Data\_An... +14 -0

sample\_data.csv +9 -1

README\_Task3.md +93 -1

Reject, suggest, follow up?

Task3\_CSV\_Data\_Analysis.ipynb

Task3: CSV Data Analysis with Gemini > Step 7: Final Output Summary

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ... .venv (Python 3.9.6)

Step 1: Install Required Libraries

# Install required packages  
%pip install -q google-generativeai pandas numpy

Step 2: Import Libraries and Setup

import pandas as pd  
import numpy as np  
import google.generativeai as genai  
import os  
from IPython.display import display, HTML

print("Libraries imported successfully!")

Libraries imported successfully!

All support for the 'google.generativeai' package has ended. It will no longer receive updates or bug fixes. Please switch to the 'google.genai' package as soon as possible. See README for more details:

Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample\_data.csv?

Show Recommendations Install

Search Agents...

New Agent

Agents

CSV file statistical an... 14m

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

showing all results

Ready for Colab — can be uploaded and run directly

Expected Output:

The notebook produces a final summary table like:

Column	Mean	Min	Max
Age	32.75	25	45
Salary	63750.00	50000	80000
Score	89.63	85	95

To Use:

1. Upload the notebook to Google Colab

2. Get your Gemini API key from Google AI Studio

3. Replace YOUR\_API\_KEY\_HERE in the configuration cell

4. Run all cells — the notebook will calculate and display mean, min, and max values

The notebook is ready to use. Run it in Colab and take a screenshot of the final output showing the statistical analysis results.

3 Files

Task3\_CSV\_Data\_An... +14 -0

sample\_data.csv +9 -1

README\_Task3.md +93 -1

Reject, suggest, follow up?

Task3\_CSV\_Data\_Analysis.ipynb

Task3: CSV Data Analysis with Gemini > Step 7: Final Output Summary

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ... .venv (Python 3.9.6)

Step 3: Configure Gemini API

Note: You need to get your Gemini API key from Google AI Studio

# Configure Gemini API  
# Option 1: Set your API key here (replace with your actual key)  
GEMINI\_API\_KEY = "YOUR\_API\_KEY\_HERE"

# Option 2: Or use environment variable  
# GEMINI\_API\_KEY = os.getenv("GEMINI\_API\_KEY")

# Configure the API  
genai.configure(api\_key=GEMINI\_API\_KEY)

print("Gemini API configured successfully!")

Gemini API configured successfully!

Step 4: Upload CSV File

Upload your CSV file using the file uploader below, or use a sample CSV file.

# Read the CSV file  
csv\_file = "sample\_data.csv" # Change this to your uploaded file name

# If you uploaded a file, uncomment and use:  
# csv\_file = list(uploaded.keys())[0]

df = pd.read\_csv(csv\_file)

print("CSV file loaded successfully!")  
print(f"Shape: {df.shape}")  
print(f"First few rows:")  
display(df.head())

CSV file loaded successfully!

Shape: (8, 4)

First few rows:

	Name	Age	Salary	Score
0	Alice	25	50000	85
1	Bob	30	60000	90
2	Charlie	35	70000	88
3	Diana	28	55000	92
4	Eve	32	65000	87

Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample\_data.csv?

Show Recommendations Install

## Step 5: Traditional Statistical Analysis (Baseline)

Search Agents...

New Agent

Agents

CSV file statistical an... 14m

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

Auto

Salary 63750.00 50000 80000  
Score 89.63 85 95

To Use:  
1. Upload the notebook to Google Colab  
2. Get your Gemini API key from Google AI Studio  
3. Replace YOUR\_API\_KEY\_HERE in the configuration cell  
4. Run all cells — the notebook will calculate and display mean, min, and max values

The notebook is ready to use. Run it in Colab and take a screenshot of the final output showing the statistical analysis results.

3 Files  
Task3\_CSV\_Data\_An... +14 -6  
sample\_data.csv +9 -1  
README\_Task3.md +93 -1

Reject, suggest, follow up?

Auto

Task3\_CSV\_Data\_Analysis.ipynb

Task3\_CSV\_Data\_Analysis.ipynb > Task 3: CSV Data Analysis with Gemini > Task 7: Final Output Summary

Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline

.venv (Python 3.9.6)

## Step 5: Traditional Statistical Analysis (Baseline)

First, let's calculate mean, min, and max using traditional methods for comparison.

Undo MN Keep Cell

markdown

# Calculate statistics for numeric columns only

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

print("=" \* 60)

print("TRADITIONAL STATISTICAL ANALYSIS")

print("=" \* 60)

stats\_df = pd.DataFrame({

'Column': numeric\_cols,

'Mean': [df[col].mean() for col in numeric\_cols],

'Min': [df[col].min() for col in numeric\_cols],

'Max': [df[col].max() for col in numeric\_cols]

})

display(stats\_df)

print("\ndetailed Statistics:")

print(df[numeric\_cols].describe())

0.0s Python

TRADITIONAL STATISTICAL ANALYSIS

Column Mean Min Max

0 Age 32.750 25 45

1 Salary 63750.000 50000 80000

2 Score 89.625 85 95

Detailed Statistics:

count 8.000000 8.000000 8.000000

mean 32.750000 63750.000000 89.625000

std 6.408699 9895.886591 3.113069

min 25.000000 50000.000000 85.000000

25% 28.750000 57250.000000 87.750000

50% 31.000000 62500.000000 89.500000

75% 35.750000 70500.000000 91.250000

max 45.000000 80000.000000 95.000000

Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample\_data.csv?

Show Recommendations Install

Search Agents...

New Agent

Agents

CSV file statistical an... 16m

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

Auto

Salary 63750.00 50000 80000  
Score 89.63 85 95

To Use:  
1. Upload the notebook to Google Colab  
2. Get your Gemini API key from Google AI Studio  
3. Replace YOUR\_API\_KEY\_HERE in the configuration cell  
4. Run all cells — the notebook will calculate and display mean, min, and max values

The notebook is ready to use. Run it in Colab and take a screenshot of the final output showing the statistical analysis results.

3 Files  
Task3\_CSV\_Data\_An... +14 -6  
sample\_data.csv +9 -1  
README\_Task3.md +93 -1

Reject, suggest, follow up?

Auto

Task3\_CSV\_Data\_Analysis.ipynb

Task3\_CSV\_Data\_Analysis.ipynb > Task 3: CSV Data Analysis with Gemini > Task 7: Final Output Summary

Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline

.venv (Python 3.9.6)

## Step 6: Gemini-Powered Analysis

Now, let's use Gemini to analyze the CSV data and calculate statistics.

markdown

# Prepare data for Gemini

# Convert DataFrame to string format

data\_preview = df.head(10).to\_string()

data\_summary = f"\nData shape: {df.shape}\n"

data\_summary += f"\nColumns: {list(df.columns)}\n"

data\_summary += f"\nNumeric columns: {list(df[numeric\_cols].columns)}\n"

print("Data prepared for Gemini analysis")

0.0s Python

Data prepared for Gemini analysis

## Step 7: Final Output Summary

## Mean, Min, Max Values:

markdown

# Final comprehensive summary

print("=" \* 70)

print("FINAL STATISTICAL ANALYSIS - MEAN, MIN, MAX")

print("=" \* 70)

final\_stats = pd.DataFrame({

'Column': numeric\_cols,

'Mean': [round(df[col].mean(), 2) for col in numeric\_cols],

'Min': [df[col].min() for col in numeric\_cols],

'Max': [df[col].max() for col in numeric\_cols]

})

# Display with better formatting

display(HTML(final\_stats.to\_html(index=False, classes='table table-striped')))

print("\n" \* 70)

print("detailed Statistics:")

print("\n" \* 70)

display(df[numeric\_cols].describe())

print("\n" \* 70)

print("ANALYSIS COMPLETE!")

print("\n" \* 70)

11s Python

Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample\_data.csv?

Show Recommendations Install

OUTPUT:

**CSV file statistical analysis**

16m • +624 • Auto

**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

Salary 63750.00 50000 80000  
Score 89.63 85 95

**To Use:**

1. Upload the notebook to Google Colab
2. Get your Gemini API key from Google AI Studio
3. Replace YOUR\_API\_KEY\_HERE in the configuration cell
4. Run all cells — the notebook will calculate and display mean, min, and max values

The notebook is ready to use. Run it in Colab and take a screenshot of the final output showing the statistical analysis results.

Task3\_CSV\_Data\_Analysis.ipynb •

Task3\_CSV\_Data\_Analysis.ipynb • Task 3: CSV Data Analysis with Gemini • Step 7: Final Output Summary

Code • Markdown • Run All • Restart • Clear All Outputs • Jupyter Variables • Outline •

FINAL STATISTICAL ANALYSIS - MEAN, MIN, MAX

Column	Mean	Min	Max
Age	32.75	25	45
Salary	63750.00	50000	80000
Score	89.62	85	95

Detailed Statistics:

	Age	Salary	Score
count	8.000000	8.000000	8.000000
mean	32.750000	63750.000000	89.625000
std	6.408699	9895.886591	3.113909
min	25.000000	50000.000000	85.000000
25%	28.750000	57250.000000	87.750000
50%	31.000000	62500.000000	89.500000
75%	35.750000	70500.000000	91.250000
max	45.000000	80000.000000	95.000000

ANALYSIS COMPLETE!

Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample\_data.csv?

Show Recommendations Install

## 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:

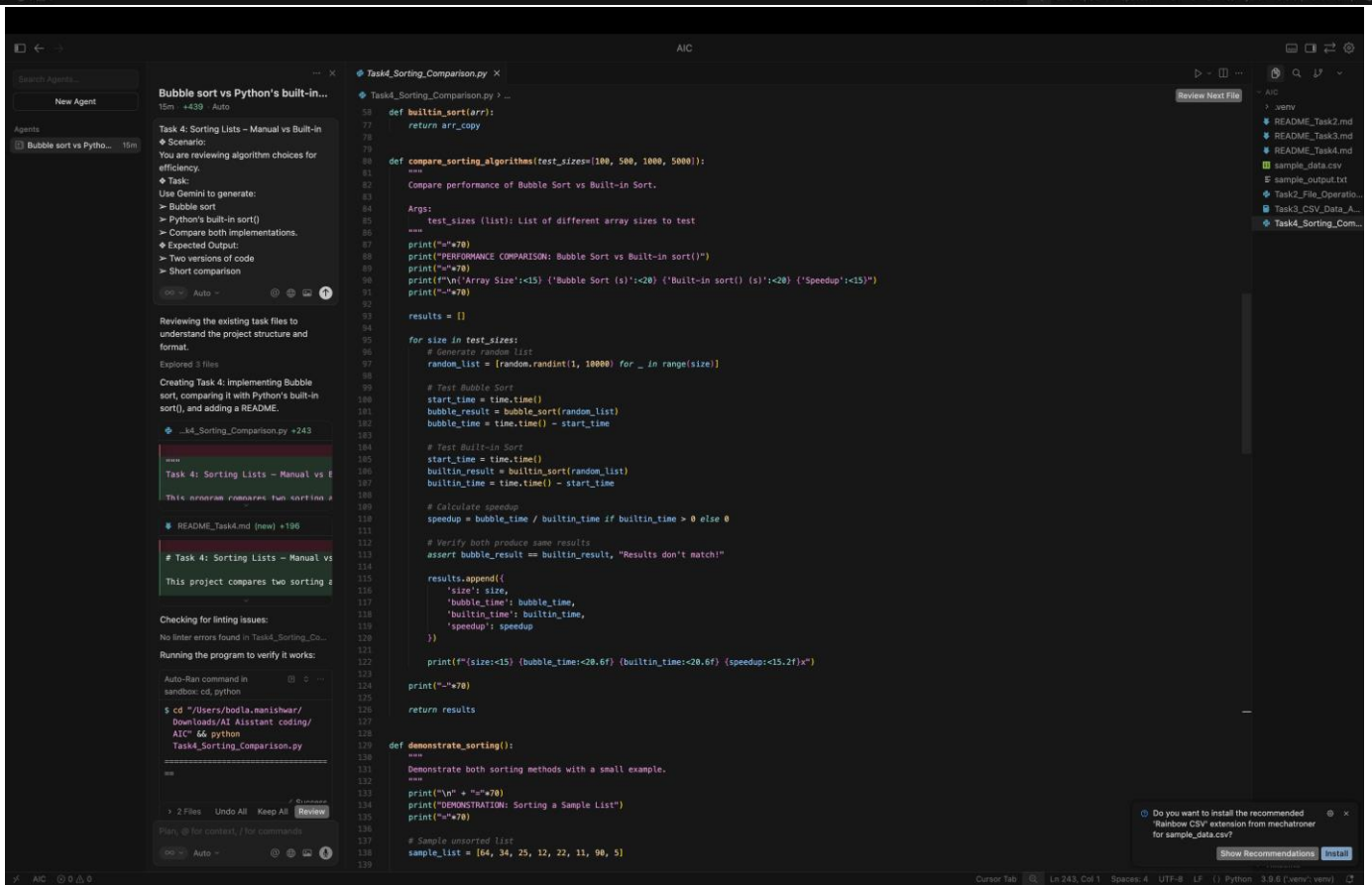
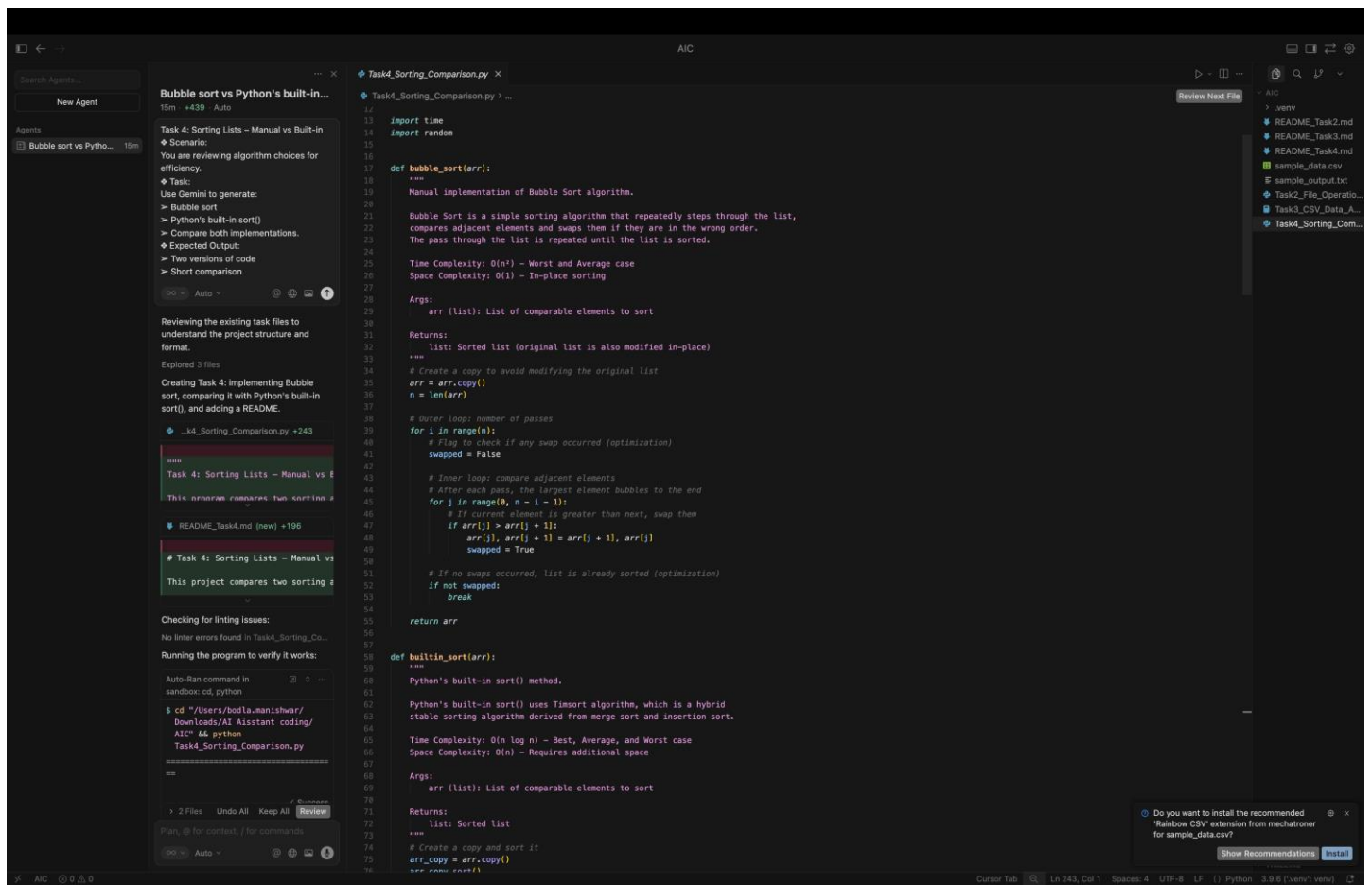
- Two versions of code
- Short comparison

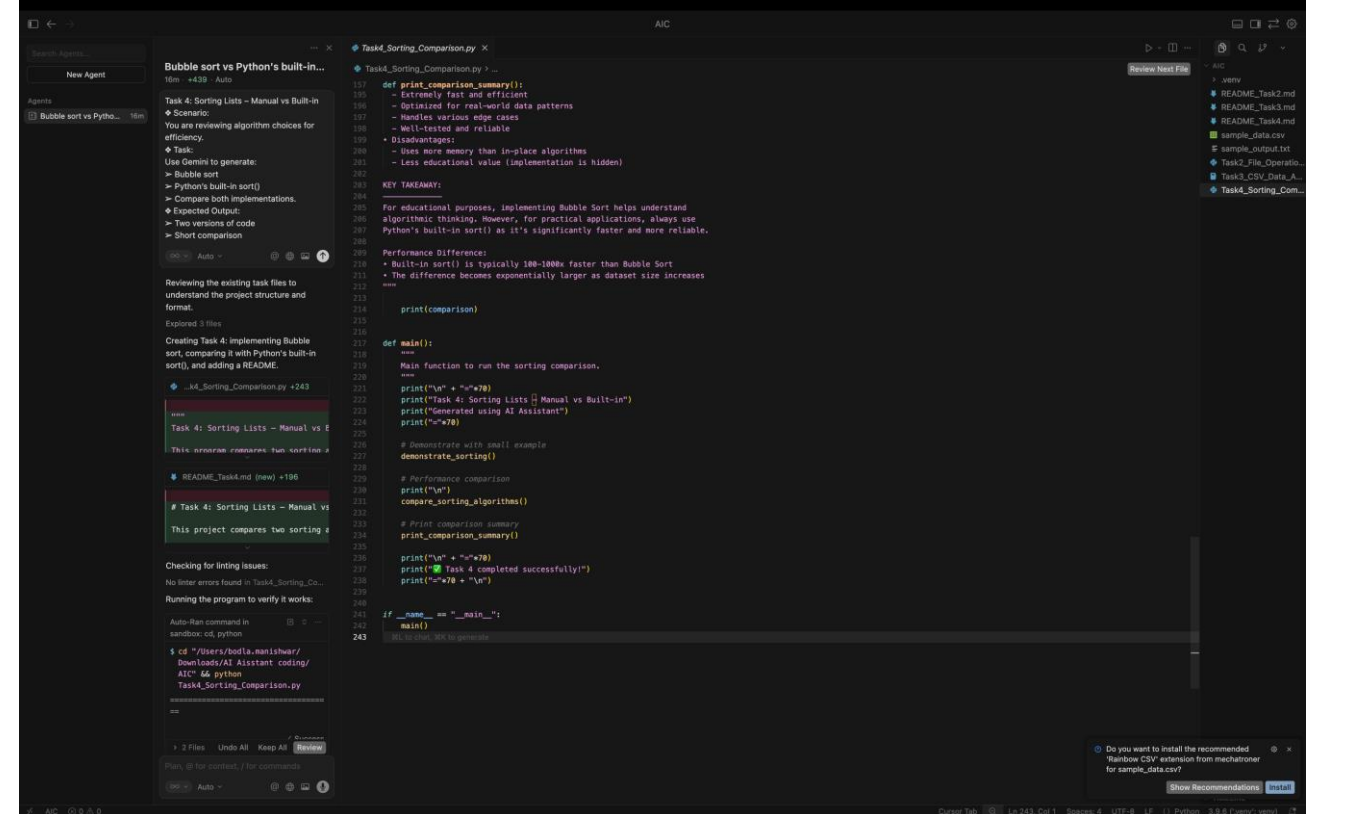
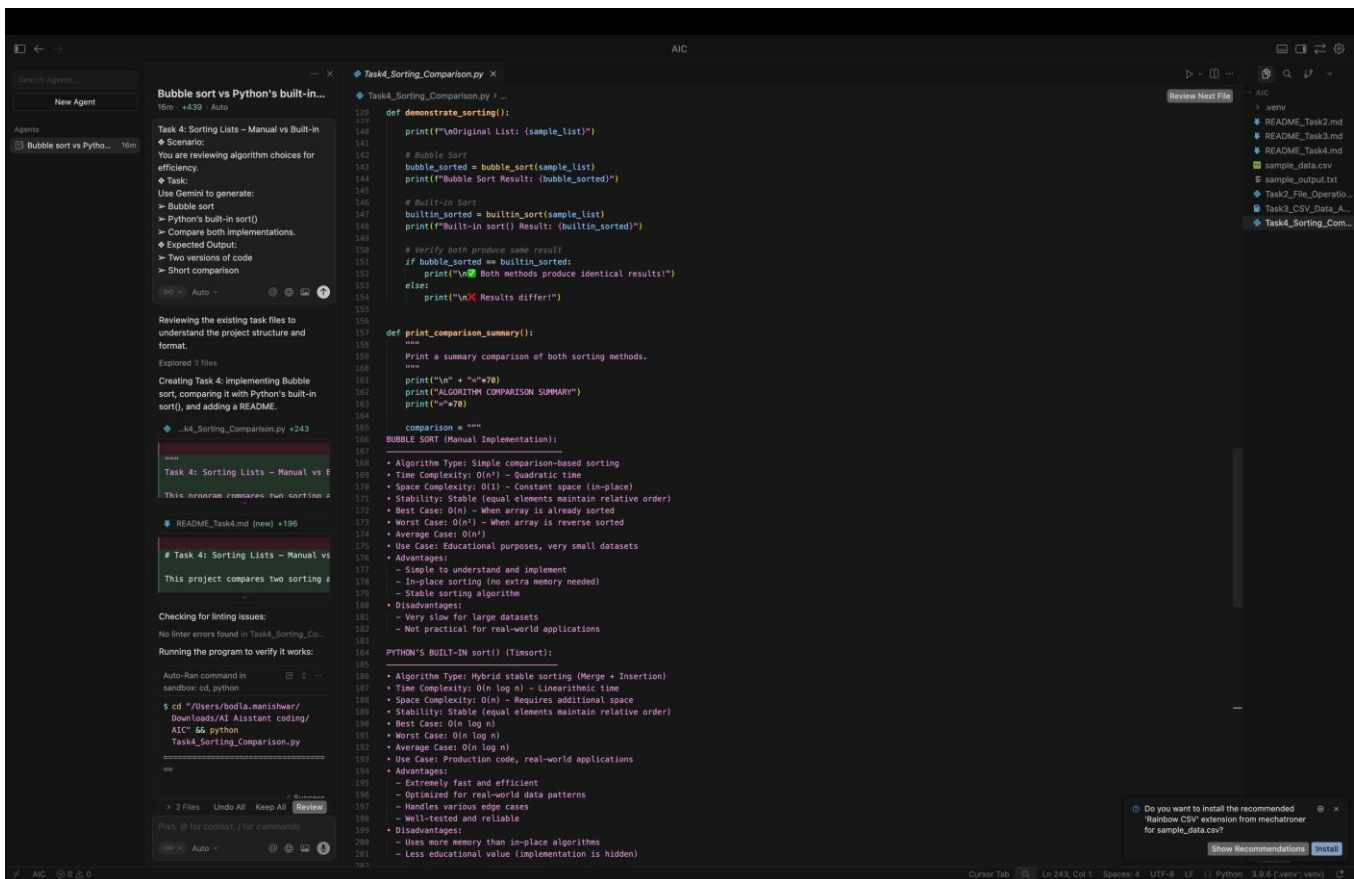
## PROMPT:

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.

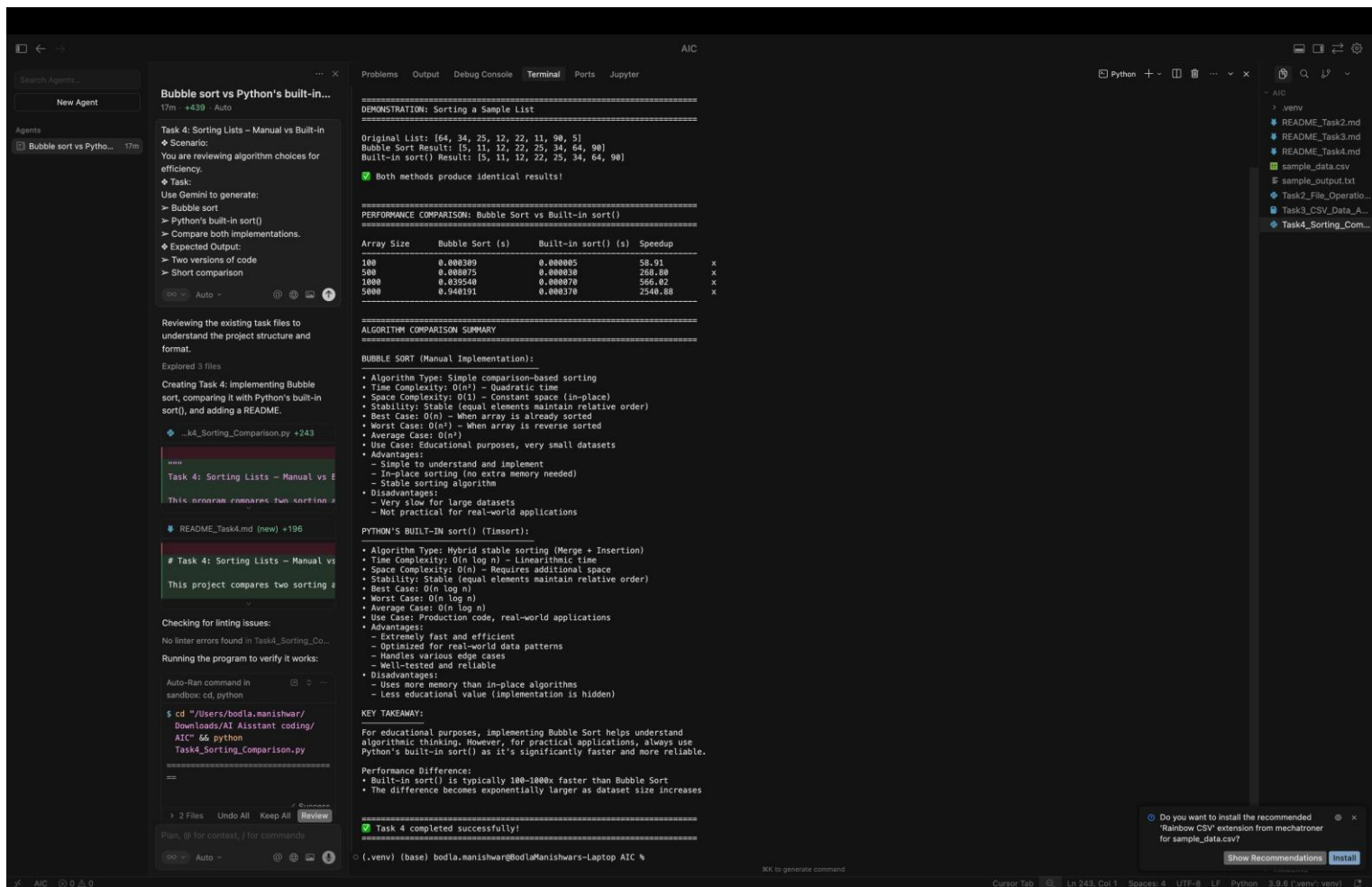
## CODE:







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  $O(n^2)$  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.