ASSIGNMENT-02

NAME: A. Supreeth reddy
Hall Ticket:2303A51392
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

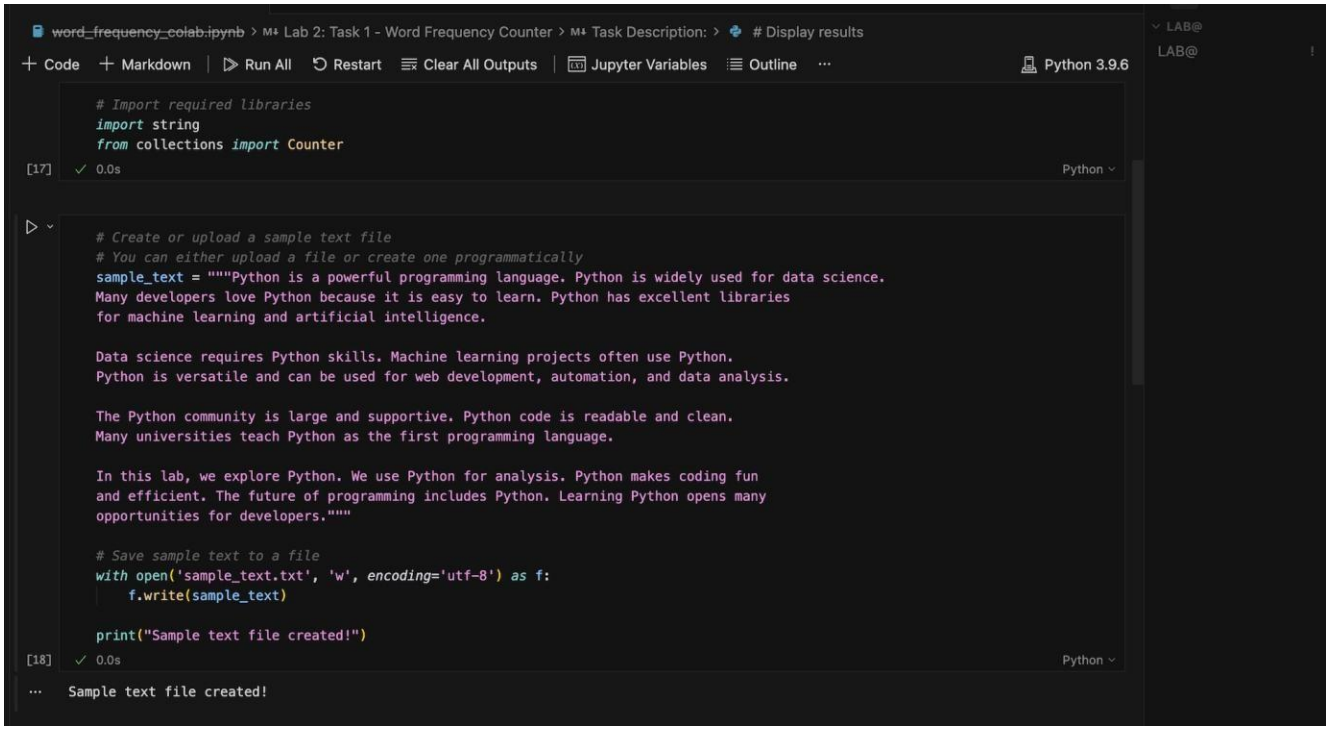
➢ Explanation ➢Screenshot

Solution:

PROMPT

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

CODE:



word_frequency_colab.ipynb > Lab 2: Task 1 - Word Frequency Counter > Task Description: > # Display results

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

```python
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
```

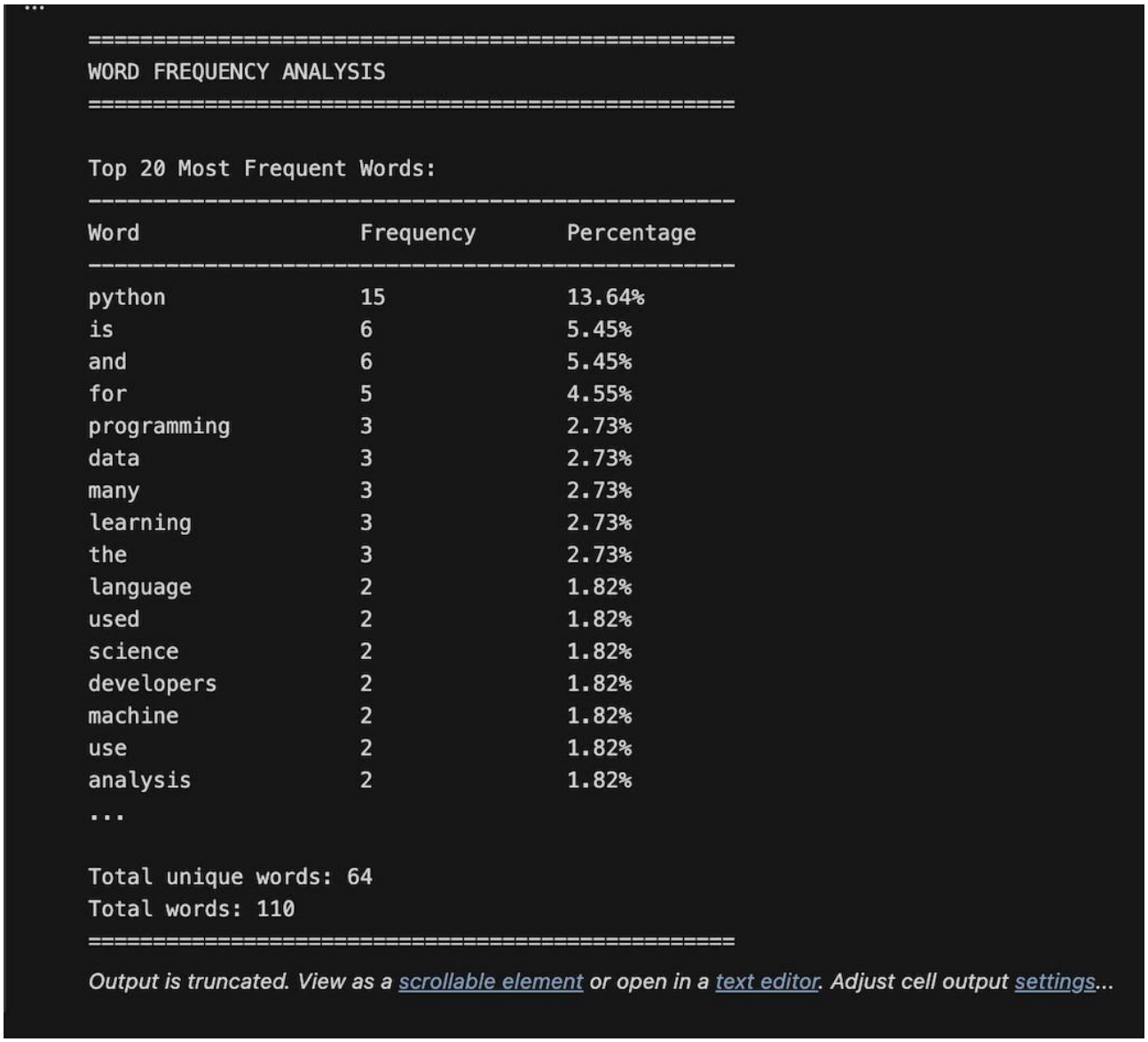
[19] ✓ 0.0s Python

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

[20] ✓ 0.0s Python

```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:



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

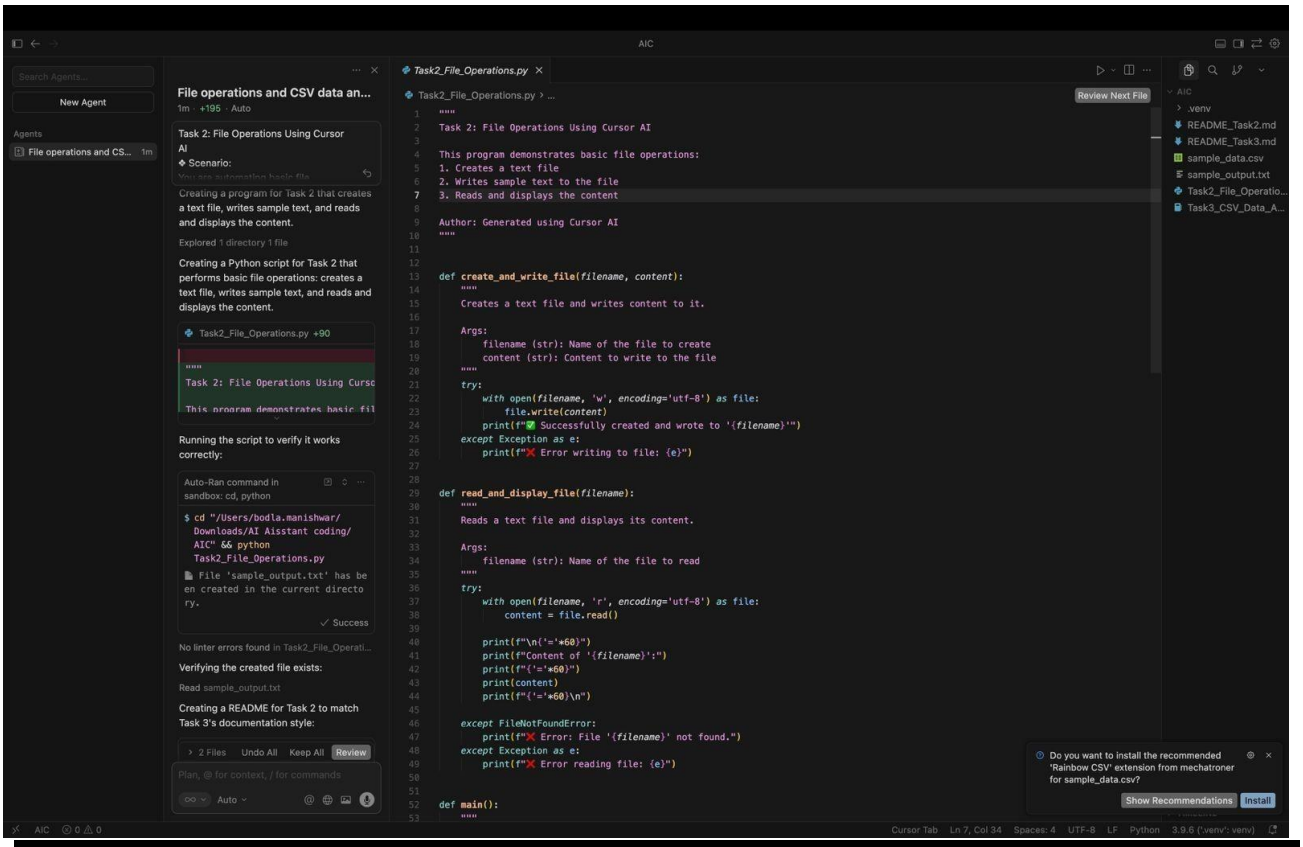
➢ Reads and displays the content❖ Expected Output:

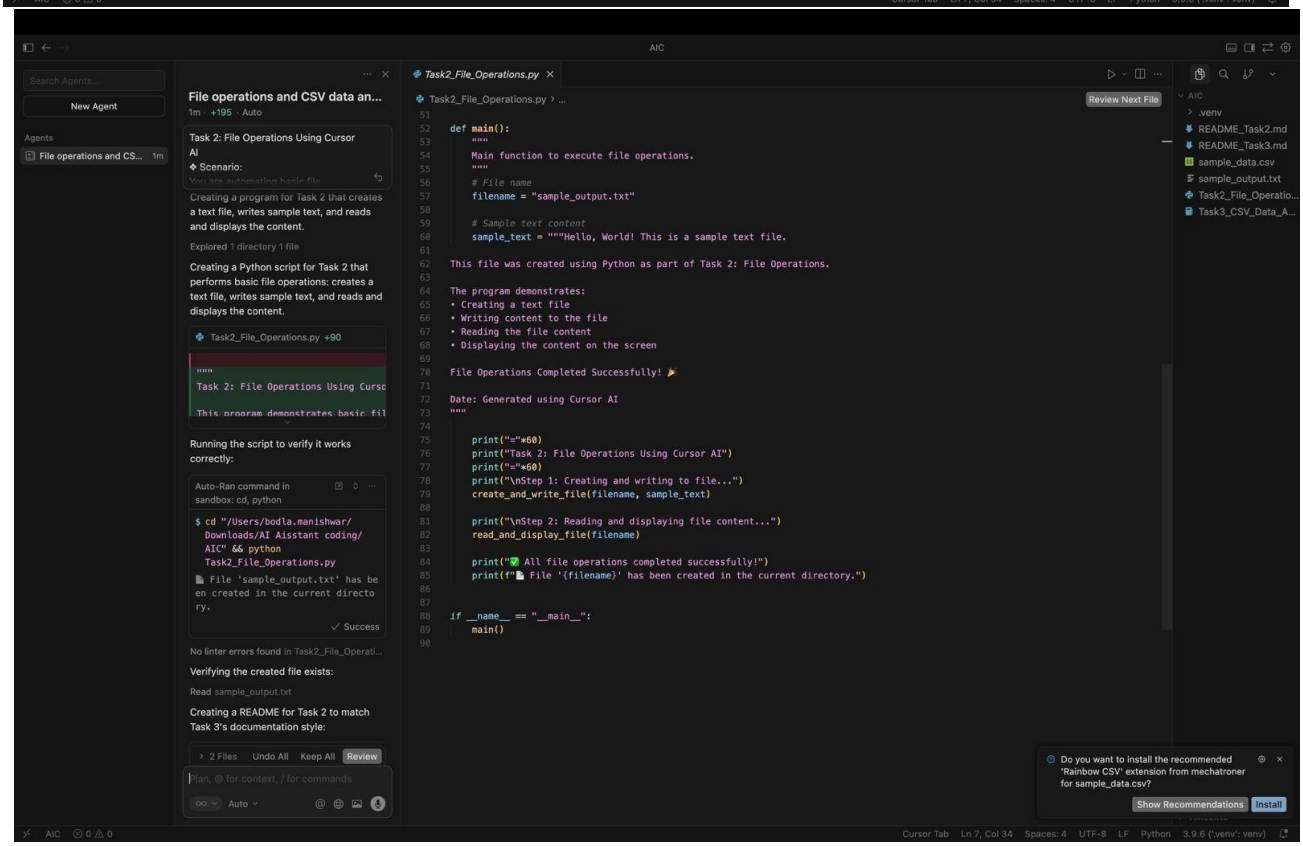
➢ Functional code➢Cursor AI screenshots

PROMPT:

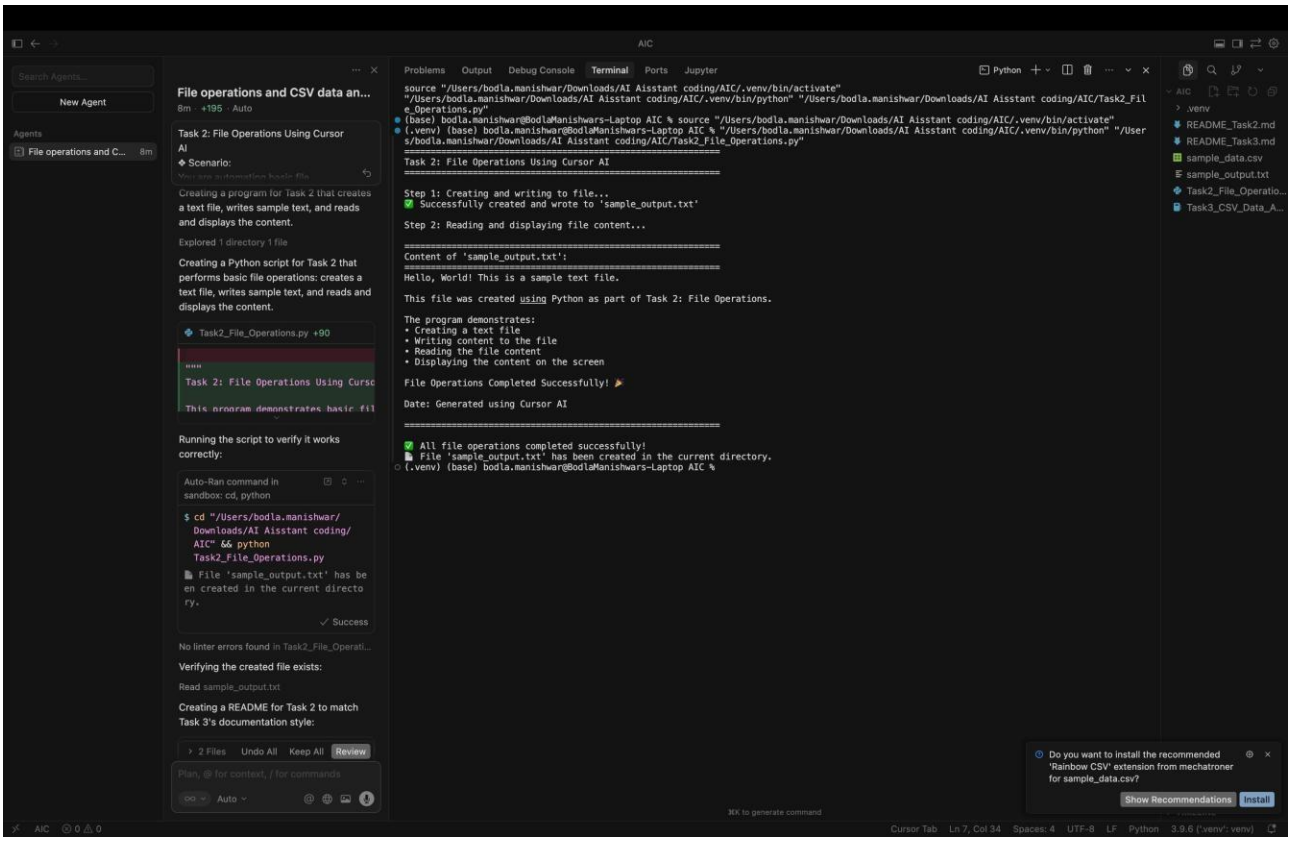
Generate a simple Python program that demonstrates basic file operations. The program should create a text file, write some sample text into it, then read the content from the file and display it on the screen.

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:

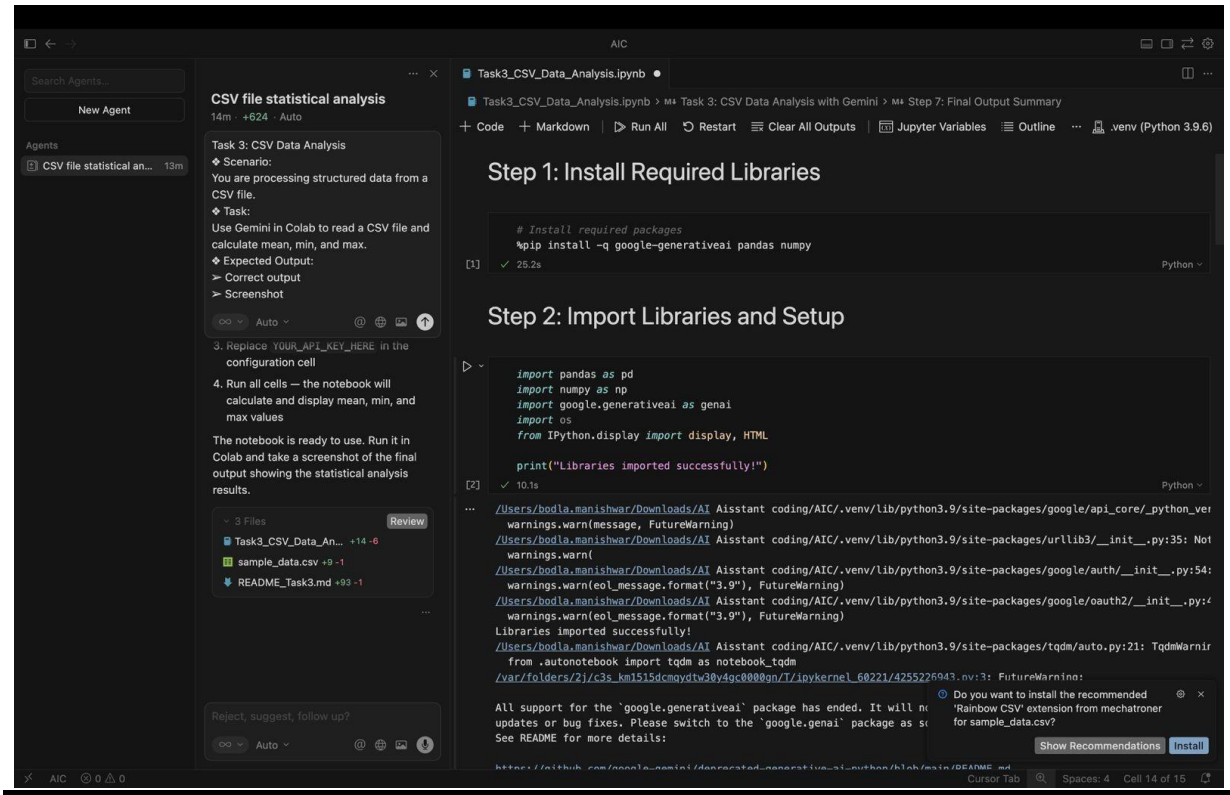
➢ Correct output

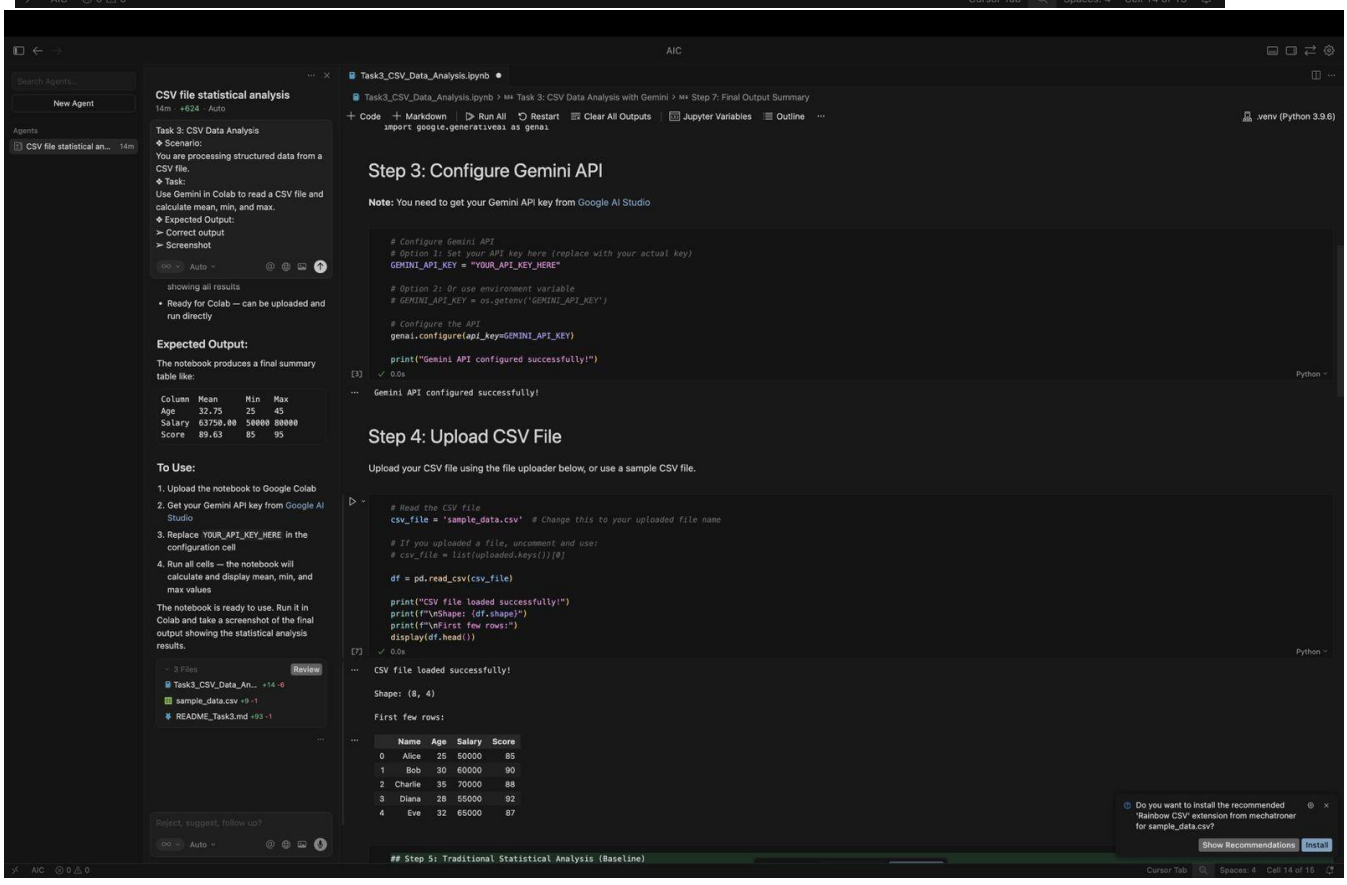
➢ Screenshot

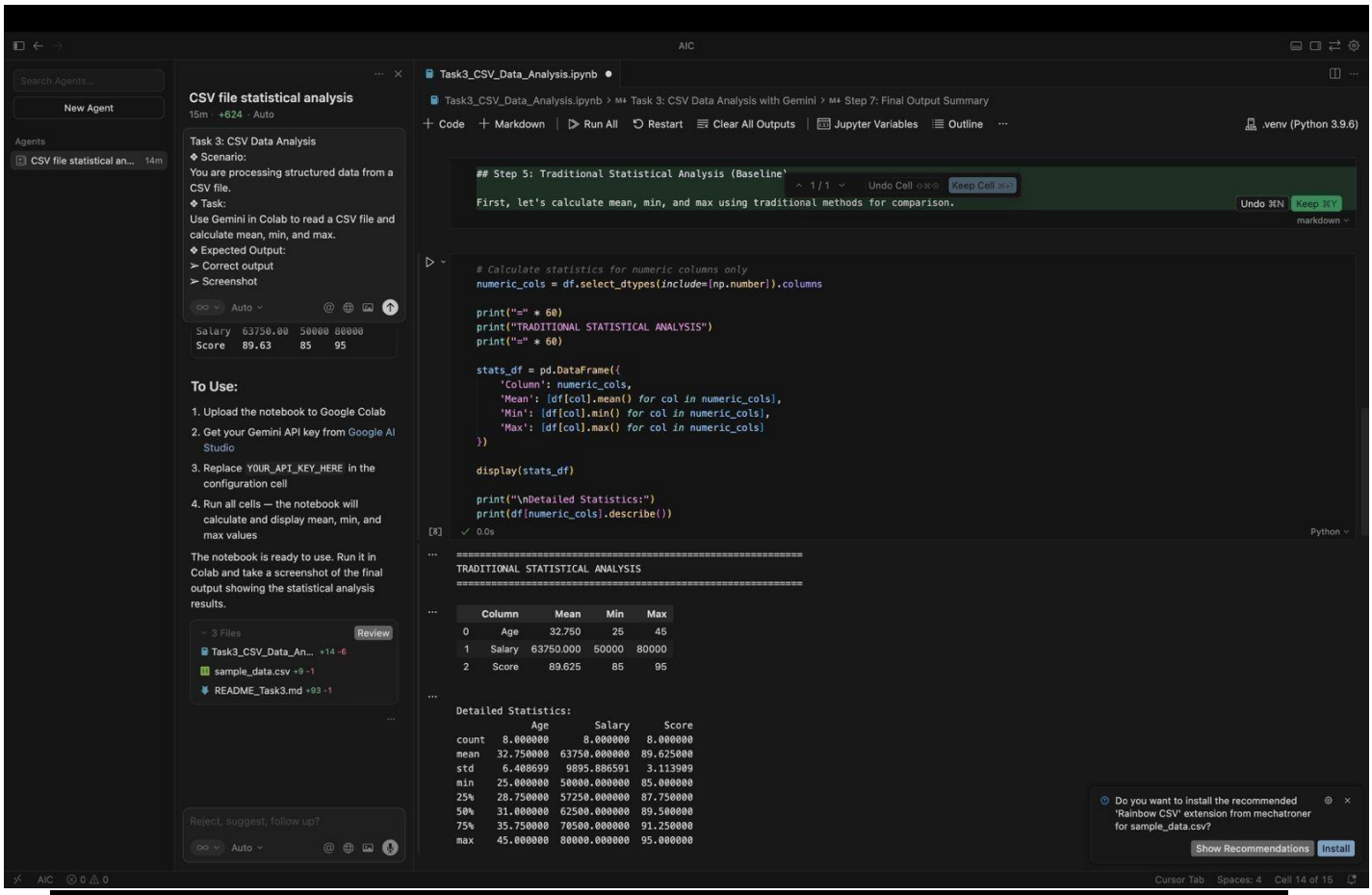
PROMPT:

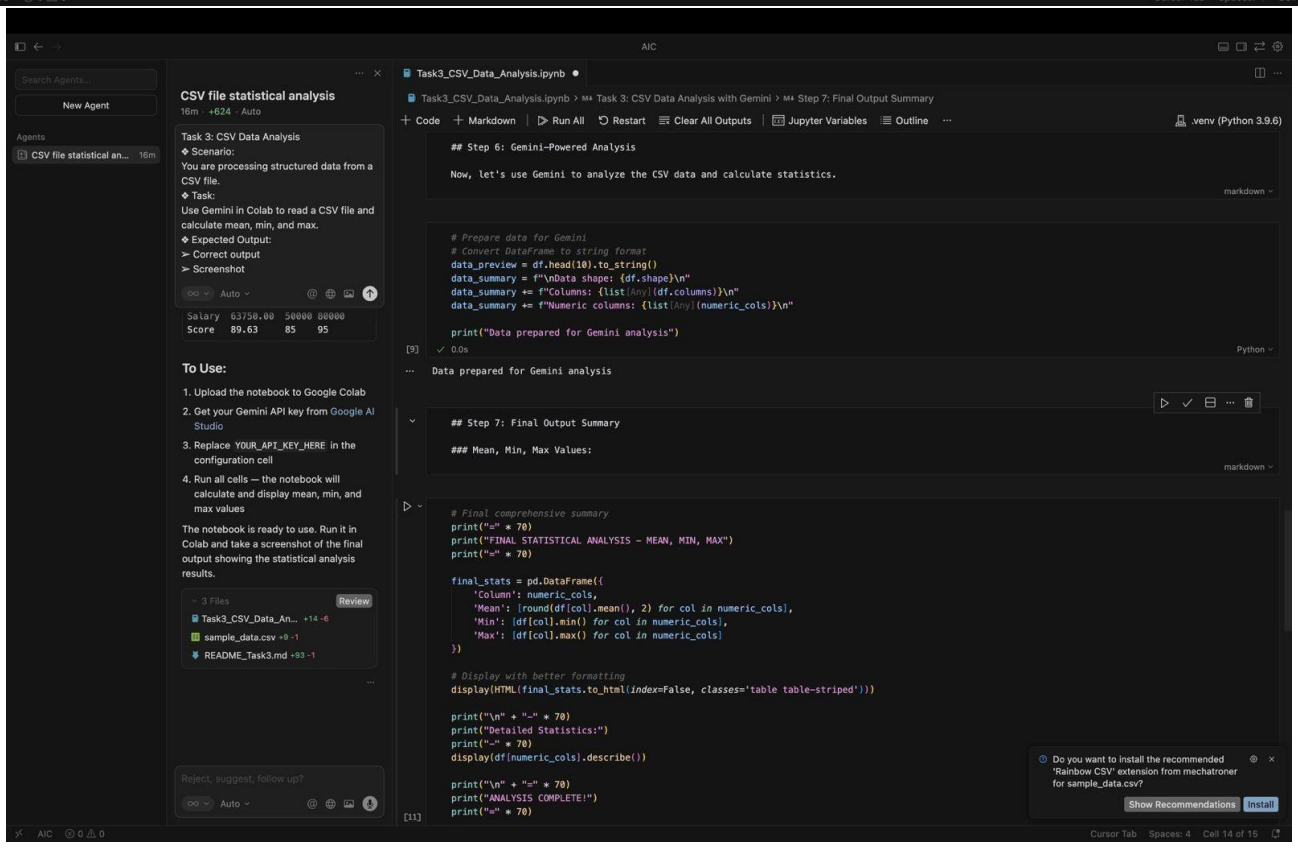
Write Python code in Google Colab to read a CSV file and calculate mean, minimum, and maximum values using pandas.

CODE:

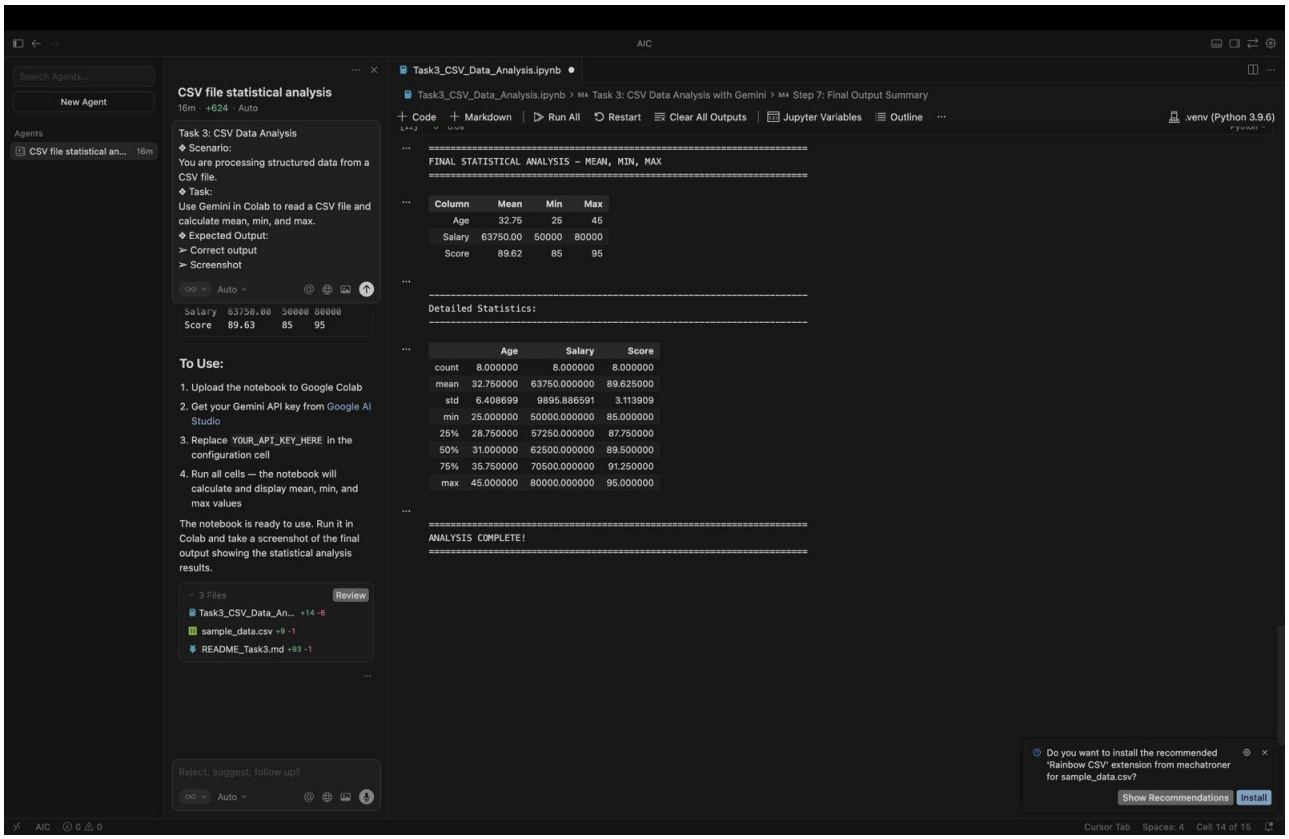

AIC
Search Agents...
New Agent
Agents
CSV file statistical an... 13m
CSV file statistical analysis
14m · +624 · Auto
Task 3: CSV Data Analysis
❖ Scenario:
You are processing structured data from a CSV file.
❖ Task:
Use Gemini in Colab to read a CSV file and calculate mean, min, and max.
❖ Expected Output:
➢ Correct output
➢ Screenshot
Auto
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
Review
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 > 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
[1] 25.2s
Python
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!")
[2] 10.1s
Python
/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC/.venv/lib/python3.9/site-packages/google/api_core/_python_ver
warnings.warn(message, FutureWarning)
/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC/.venv/lib/python3.9/site-packages/urllib3/__init__.py:35: Not
warnings.warn(
/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC/.venv/lib/python3.9/site-packages/google/auth/__init__.py:54:
warnings.warn(eol_message.format("3.9"), FutureWarning)
/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC/.venv/lib/python3.9/site-packages/google/oauth2/__init__.py:4
warnings.warn(eol_message.format("3.9"), FutureWarning)
Libraries imported successfully!
/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC/.venv/lib/python3.9/site-packages/tqdm/auto.py:21: TqdmWarnir
from .autonotebook import tqdm as notebook_tqdm
/var/folders/2j/c3s_km1515dcmqydtw30y4gc0000gn/T/ipykernel_60221/4255226043.py:3: FutureWarning:
All support for the `google.generativeai` package has ended. It will no
updates or bug fixes. Please switch to the `google.genai` package as so
See README for more details:
https://github.com/google-gemini/deprecated-generative-ai-python/blob/main/README.md
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install
AIC
Cursor Tab
Spaces: 4
Cell 14 of 15


AIC
CSV file statistical analysis
14m · +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
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.
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!")
[3] 0.0s
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"\nShape: {df.shape}")
print(f"\nFirst few rows:")
display(df.head())
[7] 0.0s
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
Step 5: Traditional Statistical Analysis (Baseline)
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install


AIC
Search Agents...
New Agent
Agents
CSV file statistical an... 14m
CSV file statistical analysis
15m · +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
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
Review
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 > Step 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.
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())
[8] 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:
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
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install
Cursor Tab
Spaces: 4
Cell 14 of 15


AIC
Search Agents...
New Agent
Agents
CSV file statistical an... 16m
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
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
Review
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 > Step 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"Columns: {list(df.columns)}\n"
data_summary += f"Numeric columns: {list(numeric_cols)}\n"
print("Data prepared for Gemini analysis")
[9] 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("-" * 70)
display(df[numeric_cols].describe())
print("\n" + "=" * 70)
print("ANALYSIS COMPLETE!")
print("=" * 70)
[11]
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install
Cursor Tab
Spaces: 4
Cell 14 of 15

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:

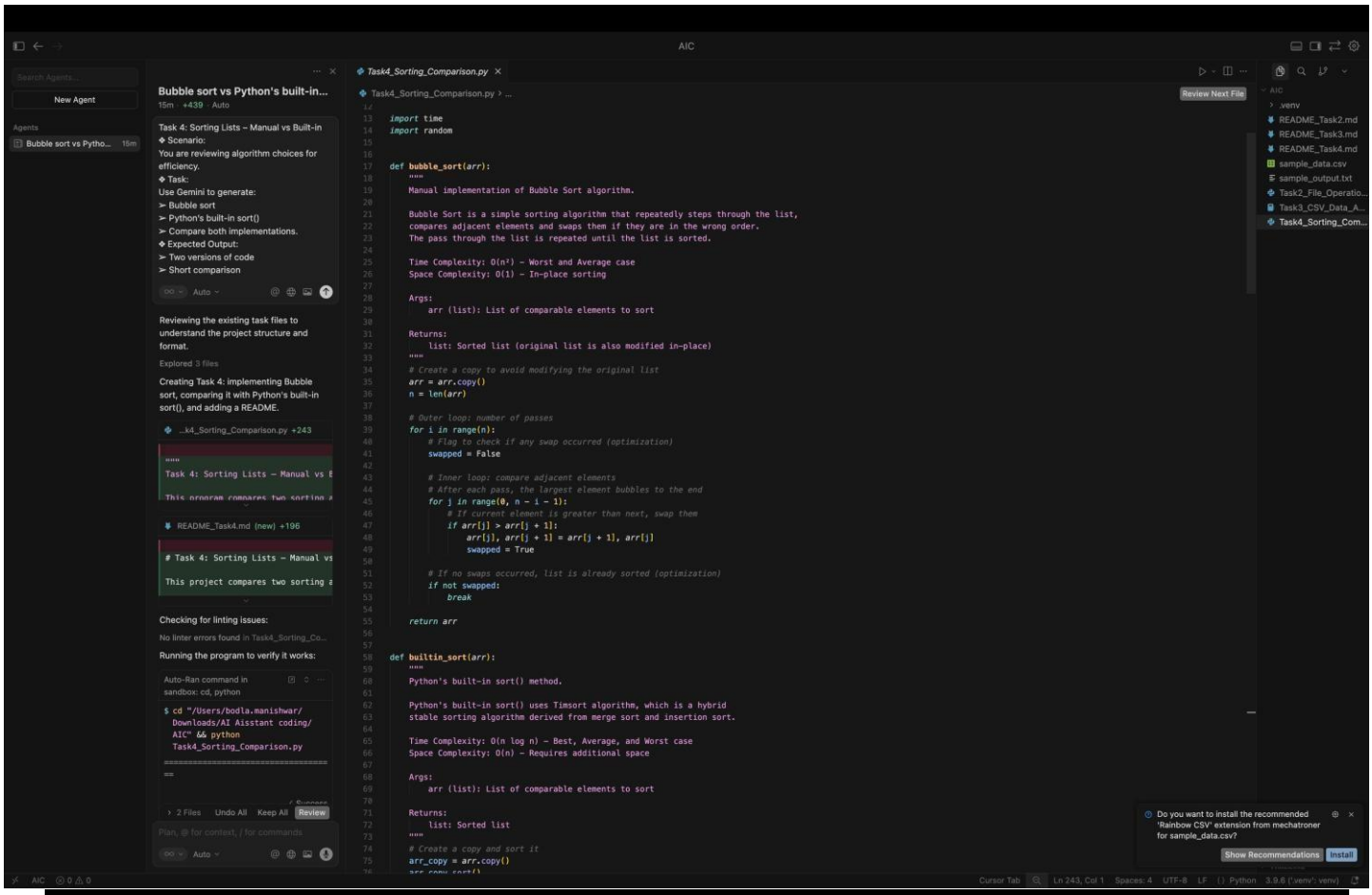
➢ Two versions of code

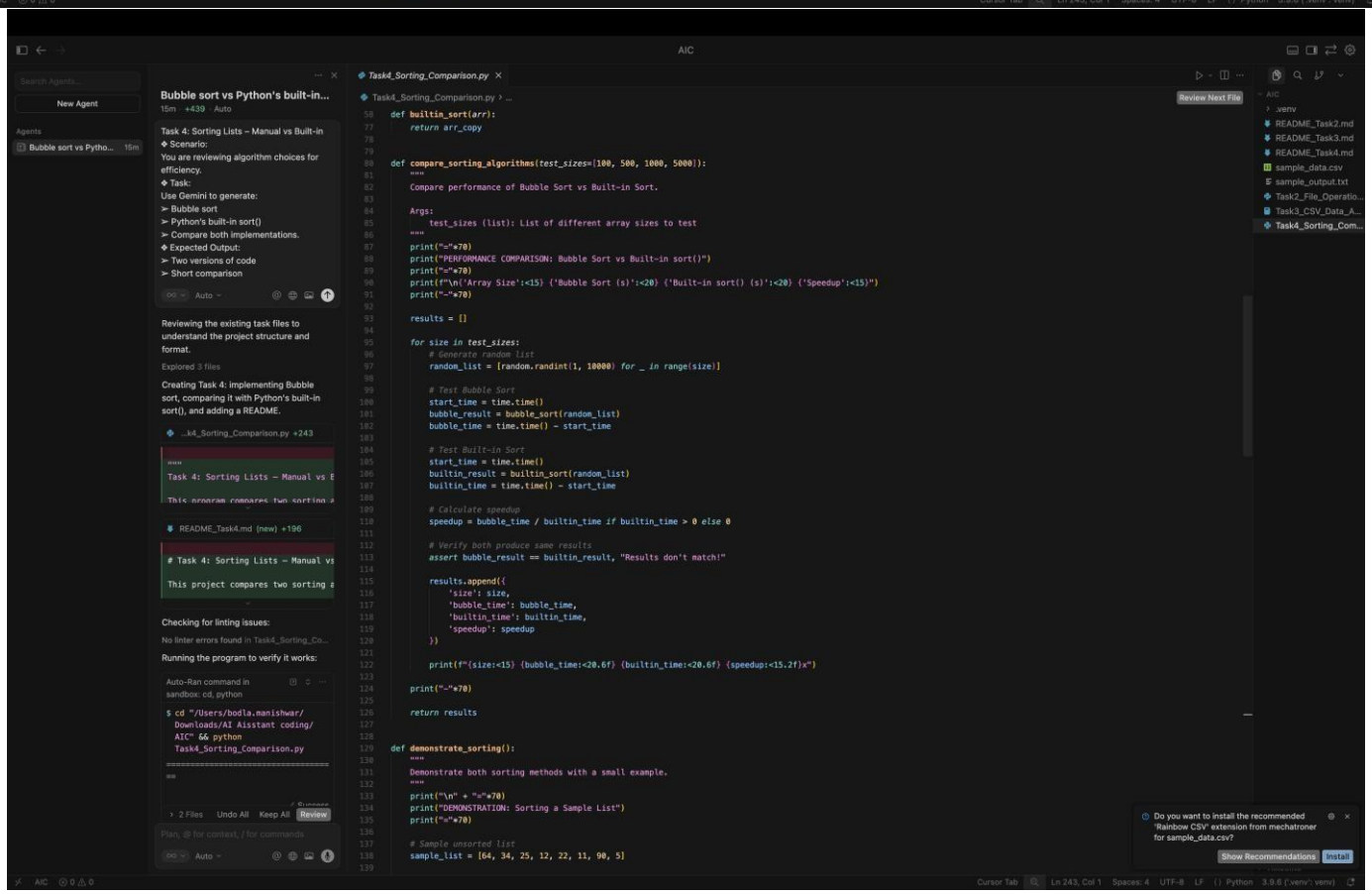
➢ Short comparison

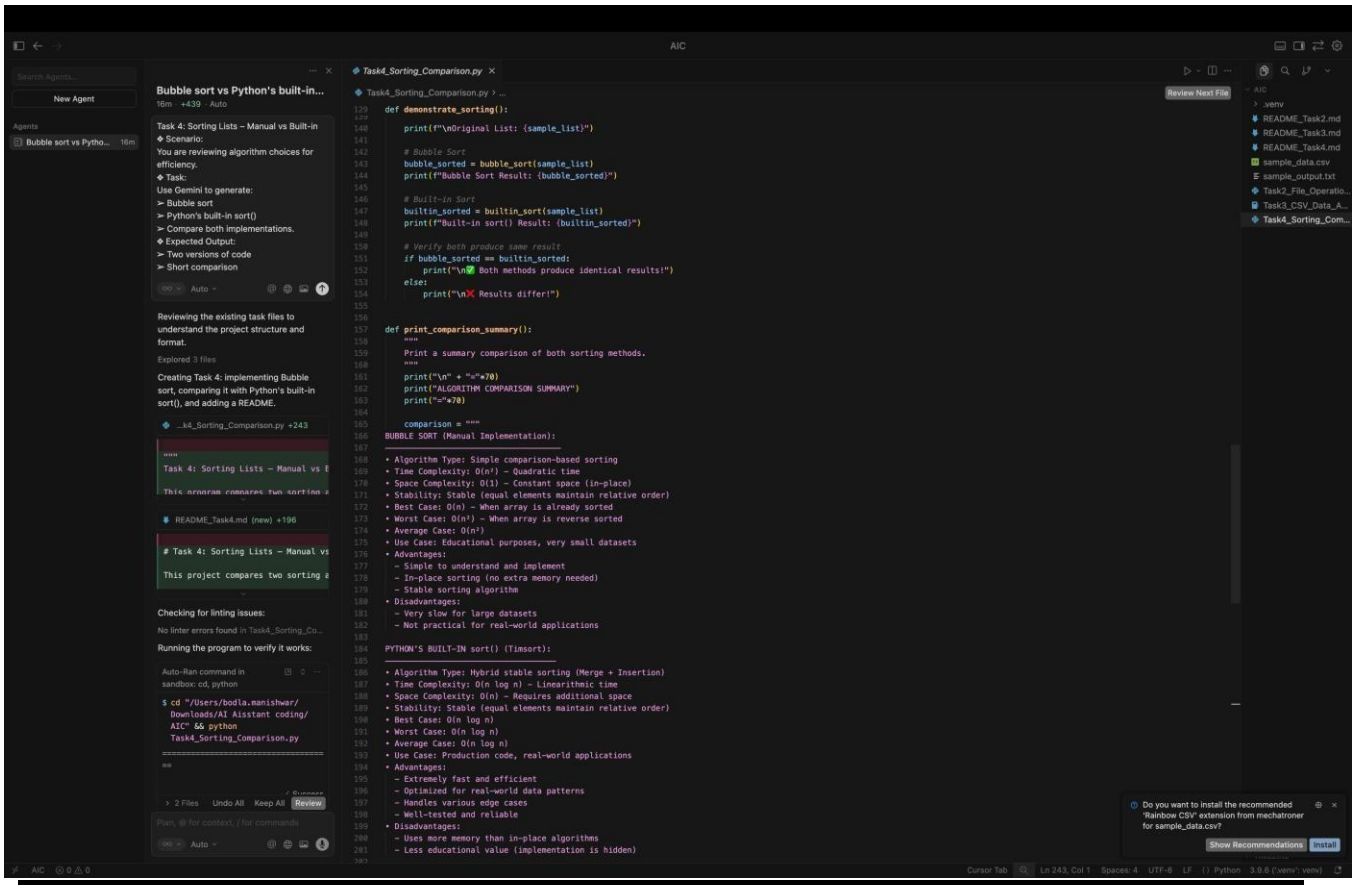
PROMPT:

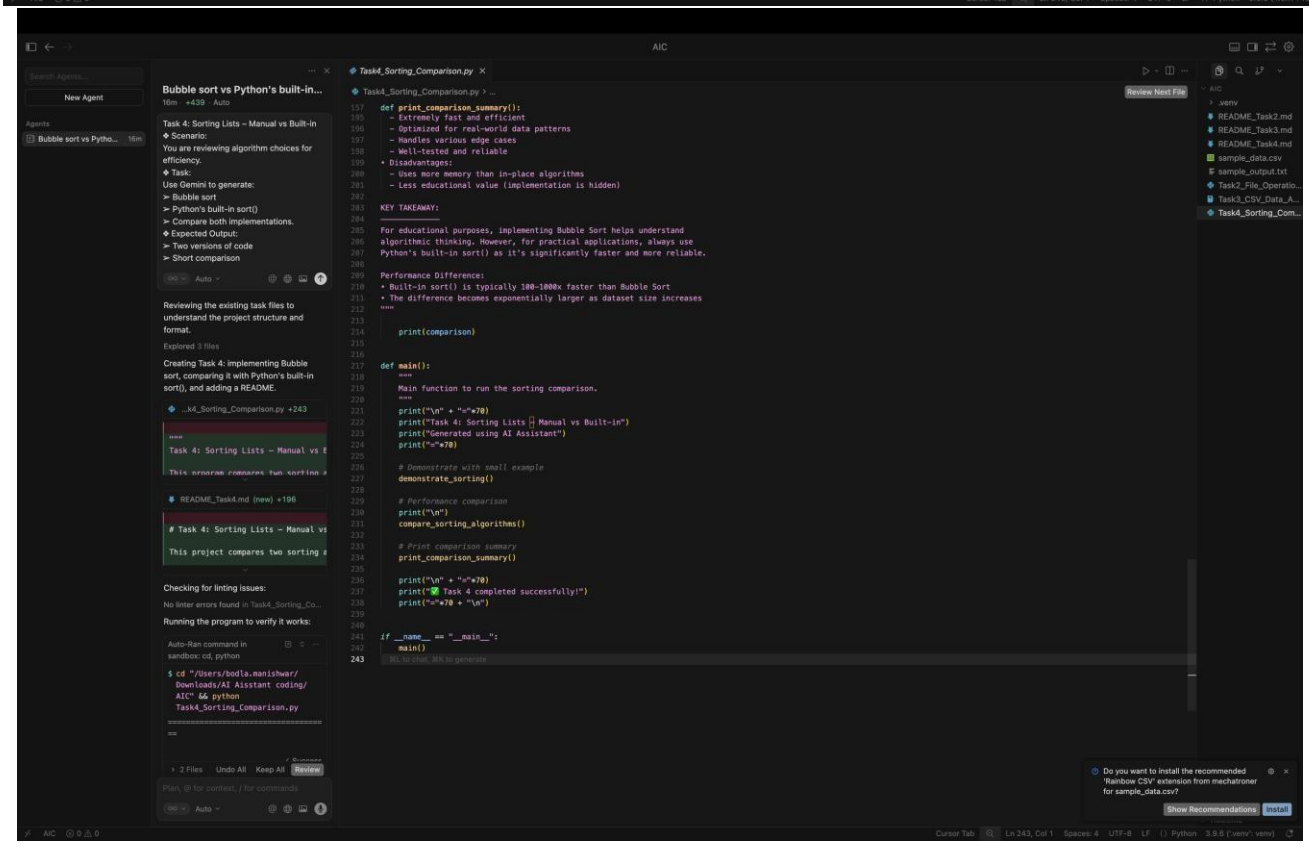
Generate Python code to sort a list using bubble sort and Python's built-in sort() method. Show both implementations clearly and provide a short comparison explaining their efficiency and usage.

CODE:

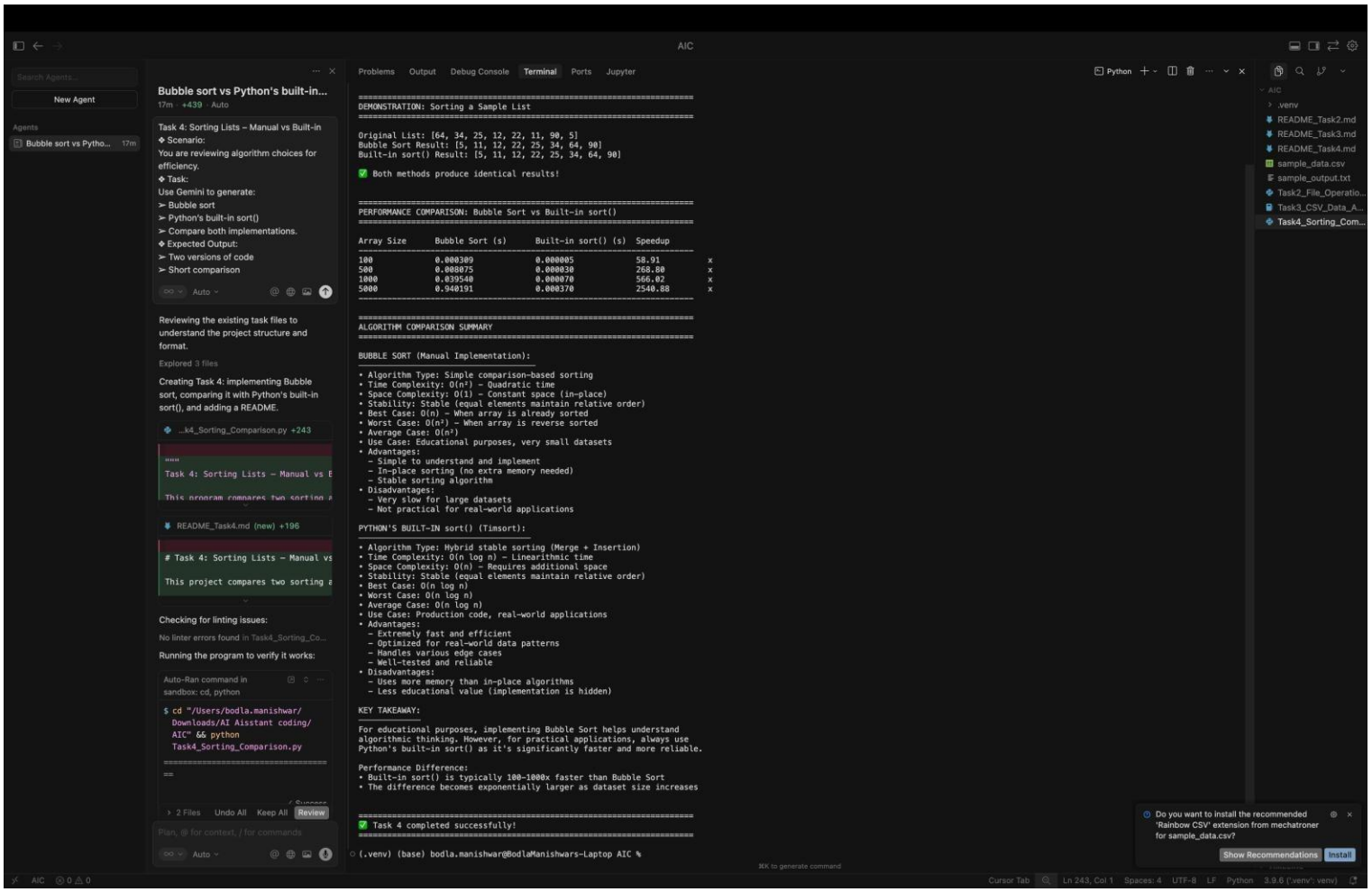

AIC
Bubble sort vs Python's built-in...
15m · +439 · Auto
Task 4: Sorting Lists – Manual vs Built-in
❖ Scenario:
You are reviewing algorithm choices for efficiency.
❖ Task:
Use Gemini to generate:
➢ Bubble sort
➢ Python's built-in sort()
➢ Compare both implementations.
❖ Expected Output:
➢ Two versions of code
➢ Short comparison
Reviewing the existing task files to understand the project structure and format.
Explored 3 files
Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.
Checking for linting issues:
No linter errors found in Task4_Sorting_Co...
Running the program to verify it works:
Auto-Ran command in sandbox: cd, python
$ cd "/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC" && python Task4_Sorting_Comparison.py
2 Files
Undo All
Keep All
Review
Plan, @ for context, / for commands
Auto
Task4_Sorting_Comparison.py
Review Next File
def bubble_sort(arr):
"""
Manual implementation of Bubble Sort algorithm.
Bubble Sort is a simple sorting algorithm that repeatedly steps through the list,
compares adjacent elements and swaps them if they are in the wrong order.
The pass through the list is repeated until the list is sorted.
Time Complexity: O(n²) - Worst and Average case
Space Complexity: O(1) - In-place sorting
Args:
arr (list): List of comparable elements to sort
Returns:
list: Sorted list (original list is also modified in-place)
"""
# Create a copy to avoid modifying the original list
arr = arr.copy()
n = len(arr)
# Outer loop: number of passes
for i in range(n):
# Flag to check if any swap occurred (optimization)
swapped = False
# Inner loop: compare adjacent elements
# After each pass, the largest element bubbles to the end
for j in range(0, n - i - 1):
# If current element is greater than next, swap them
if arr[j] > arr[j + 1]:
arr[j], arr[j + 1] = arr[j + 1], arr[j]
swapped = True
# If no swaps occurred, list is already sorted (optimization)
if not swapped:
break
return arr
def builtin_sort(arr):
"""
Python's built-in sort() method.
Python's built-in sort() uses Timsort algorithm, which is a hybrid
stable sorting algorithm derived from merge sort and insertion sort.
Time Complexity: O(n log n) - Best, Average, and Worst case
Space Complexity: O(n) - Requires additional space
Args:
arr (list): List of comparable elements to sort
Returns:
list: Sorted list
"""
# Create a copy and sort it
arr_copy = arr.copy()
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install
Cursor Tab
Ln 243, Col 1
Spaces: 4
UTF-8
LF
Python
3.9.6 ('venv': venv)


AIC
Bubble sort vs Python's built-in...
15m · +439 · Auto
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
Reviewing the existing task files to understand the project structure and format.
Explored 3 files
Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.
Checking for linting issues:
No linter errors found in Task4_Sorting_Co...
Running the program to verify it works:
Auto-Ran command in sandbox: cd, python
$ cd "/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC" && python Task4_Sorting_Comparison.py
2 Files
Undo All
Keep All
Review
Plan, @ for context, / for commands
Auto
Task4_Sorting_Comparison.py
Review Next File
def builtin_sort(arr):
return arr_copy
def compare_sorting_algorithms(test_sizes=[100, 500, 1000, 5000]):
"""
Compare performance of Bubble Sort vs Built-in Sort.
Args:
test_sizes (list): List of different array sizes to test
"""
print("="*70)
print("PERFORMANCE COMPARISON: Bubble Sort vs Built-in sort()")
print("="*70)
print(f"\n{'Array Size':<15} {'Bubble Sort (s)':<20} {'Built-in sort() (s)':<20} {'Speedup':<15}")
print("-"*70)
results = []
for size in test_sizes:
# Generate random list
random_list = [random.randint(1, 10000) for _ in range(size)]
# Test Bubble Sort
start_time = time.time()
bubble_result = bubble_sort(random_list)
bubble_time = time.time() - start_time
# Test Built-in Sort
start_time = time.time()
builtin_result = builtin_sort(random_list)
builtin_time = time.time() - start_time
# Calculate speedup
speedup = bubble_time / builtin_time if builtin_time > 0 else 0
# Verify both produce same results
assert bubble_result == builtin_result, "Results don't match!"
results.append({
'size': size,
'bubble_time': bubble_time,
'builtin_time': builtin_time,
'speedup': speedup
})
print(f"{size:<15} {bubble_time:<20.6f} {builtin_time:<20.6f} {speedup:<15.2f}x")
print("-"*70)
return results
def demonstrate_sorting():
"""
Demonstrate both sorting methods with a small example.
"""
print("\n" + "="*70)
print("DEMONSTRATION: Sorting a Sample List")
print("="*70)
# Sample unsorted list
sample_list = [64, 34, 25, 12, 22, 11, 90, 5]
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install
Cursor Tab
Ln 243, Col 1
Spaces: 4
UTF-8
LF
Python
3.9.6 ('venv': venv)


AIC
Bubble sort vs Python's built-in...
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
Reviewing the existing task files to understand the project structure and format.
Explored 3 files
Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.
Checking for linting issues:
No linter errors found in Task4_Sorting_Co...
Running the program to verify it works:
Auto-Ran command in sandbox: cd, python
$ cd "/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC" && python Task4_Sorting_Comparison.py
Review
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install


AIC
Bubble sort vs Python's built-in...
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
Reviewing the existing task files to understand the project structure and format.
Explored 3 files
Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.
Checking for linting issues:
No linter errors found in Task4_Sorting_Co...
Running the program to verify it works:
Auto-Ran command in sandbox: cd, python
$ cd "/Users/bodla.manishwar/Downloads/AI Aisstant coding/AIC" && python Task4_Sorting_Comparison.py
Review
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
Show Recommendations
Install

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.