

## 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:**

The screenshot shows a Google Colab notebook titled "word\_frequency\_colab.ipynb". The code cell at the top imports string and Counter from collections. The code then defines a variable sample\_text containing a multi-line string about Python's power in data science. It then saves this text to a file named "sample\_text.txt" and prints a confirmation message. The code cell is labeled [17] and has a runtime of 0.0s. The output cell below it is labeled [18] and also has a runtime of 0.0s, showing the message "Sample text file created!". The interface includes tabs for Code, Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, and a Python 3.9.6 environment indicator.

```
# Import required libraries
import string
from collections import Counter
[17] ✓ 0.0s

# 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."""
[18] ✓ 0.0s

# 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!")
```

```

D ▾
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:

The screenshot shows the AI Code Editor interface. The main pane displays the Python script `Task2_File_Operations.py`. The code defines two functions: `create_and_write_file` and `read_and_display_file`. It also includes a `main` function which calls these methods and prints a success message. The sidebar on the left shows the project structure and files, and the right sidebar shows a file tree and a message about installing the 'Rainbow CSV' extension.

```
1 """
2 Task 2: File Operations Using Cursor AI
3
4 This program demonstrates basic file operations:
5 1. Creates a text file
6 2. Writes sample text to the file
7 3. Reads and displays the content
8
9 Author: Generated using Cursor AI
10 """
11
12 def create_and_write_file(filename, content):
13     """
14         Creates a text file and writes content to it.
15     """
16     Args:
17         filename (str): Name of the file to create
18         content (str): Content to write to the file
19     """
20     try:
21         with open(filename, 'w', encoding='utf-8') as file:
22             file.write(content)
23         print(f"\nSuccessfully created and wrote to '{filename}'")
24     except Exception as e:
25         print(f"\nError writing to file: {e}")
26
27
28 def read_and_display_file(filename):
29     """
30         Reads a text file and displays its content.
31     """
32     Args:
33         filename (str): Name of the file to read
34     """
35     try:
36         with open(filename, 'r', encoding='utf-8') as file:
37             content = file.read()
38
39             print("\n"+="#60")
40             print("Content of '{filename}':")
41             print("#=60")
42             print(content)
43             print("#=60\n")
44
45     except FileNotFoundError:
46         print(f"\nError: File '{filename}' not found.")
47     except Exception as e:
48         print(f"\nError reading file: {e}")
49
50
51 def main():
52     """
53         Main function to execute file operations.
54     """
55     # File name
56     filename = "sample_output.txt"
57
58     # Sample text content
59     sample_text = """Hello, World! This is a sample text file.
60
61 This file was created using Python as part of Task 2: File Operations.
62
63 The program demonstrates:
64 • Creating a text file
65 • Writing content to the file
66 • Reading the file content
67 • Displaying the content on the screen
68
69 File Operations Completed Successfully! ✅
70
71 Date: Generated using Cursor AI
72
73 """
74
75     print("#=60)
76     print("Task 2: File Operations Using Cursor AI")
77     print("#=60)
78     print("\nStep 1: Creating and writing to file...")
79     create_and_write_file(filename, sample_text)
80
81     print("\nStep 2: Reading and displaying file content...")
82     read_and_display_file(filename)
83
84     print("\nAll file operations completed successfully!")
85     print(f"\nFile '{filename}' has been created in the current directory.")
86
87
88 if __name__ == "__main__":
89     main()
```

This screenshot shows the AI Code Editor with the same Python script as the previous one, but with different output. The code remains identical, but the terminal output at the bottom of the interface shows a slightly different execution path and results, indicating a different run or configuration.

```
1 """
2 Task 2: File Operations Using Cursor AI
3
4 This program demonstrates basic file operations:
5 1. Creates a text file
6 2. Writes sample text to the file
7 3. Reads and displays the content
8
9 Author: Generated using Cursor AI
10 """
11
12 def create_and_write_file(filename, content):
13     """
14         Creates a text file and writes content to it.
15     """
16     Args:
17         filename (str): Name of the file to create
18         content (str): Content to write to the file
19     """
20     try:
21         with open(filename, 'w', encoding='utf-8') as file:
22             file.write(content)
23         print(f"\nSuccessfully created and wrote to '{filename}'")
24     except Exception as e:
25         print(f"\nError writing to file: {e}")
26
27
28 def read_and_display_file(filename):
29     """
30         Reads a text file and displays its content.
31     """
32     Args:
33         filename (str): Name of the file to read
34     """
35     try:
36         with open(filename, 'r', encoding='utf-8') as file:
37             content = file.read()
38
39             print("\n"+="#60")
40             print("Content of '{filename}':")
41             print("#=60")
42             print(content)
43             print("#=60\n")
44
45     except FileNotFoundError:
46         print(f"\nError: File '{filename}' not found.")
47     except Exception as e:
48         print(f"\nError reading file: {e}")
49
50
51 def main():
52     """
53         Main function to execute file operations.
54     """
55     # File name
56     filename = "sample_output.txt"
57
58     # Sample text content
59     sample_text = """Hello, World! This is a sample text file.
60
61 This file was created using Python as part of Task 2: File Operations.
62
63 The program demonstrates:
64 • Creating a text file
65 • Writing content to the file
66 • Reading the file content
67 • Displaying the content on the screen
68
69 File Operations Completed Successfully! ✅
70
71 Date: Generated using Cursor AI
72
73 """
74
75     print("#=60)
76     print("Task 2: File Operations Using Cursor AI")
77     print("#=60)
78     print("\nStep 1: Creating and writing to file...")
79     create_and_write_file(filename, sample_text)
80
81     print("\nStep 2: Reading and displaying file content...")
82     read_and_display_file(filename)
83
84     print("\nAll file operations completed successfully!")
85     print(f"\nFile '{filename}' has been created in the current directory.")
86
87
88 if __name__ == "__main__":
89     main()
```

## OUTPUT:

The screenshot shows the AI Coding interface with the following details:

- File operations and CSV data analysis** is the active project.
- Task 2: File Operations Using Cursor AI** is selected.
- Scenario:** Describes creating a program for Task 2 that creates a text file, writes sample text, and reads and displays the content.
- Code:**

```
source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
"~/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task2_File_Operations.py"
❶ (base) bodla.manishwar@BodlaManishwars-Laptop AIC % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
❷ (.venv) (base) bodla.manishwar@BodlaManishwars-Laptop AIC % "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/task2_File_operations.py"
Task 2: File Operations Using Cursor AI
```
- Output:**
  - Step 1: Creating and writing to file... ✓ Successfully created and wrote to 'sample\_output.txt'
  - Step 2: Reading and displaying file content...
  - Content of 'sample\_output.txt':  
Hello, World! This is a sample text file.
  - This file was created using Python as part of Task 2: File Operations.
  - The program demonstrates:
    - Creating a text file
    - Writing content to the file
    - Reading the file content
    - Displaying the content on the screen
  - File Operations Completed Successfully! ➤
  - Date: Generated using Cursor AI
- Terminal:**

```
$ cd "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC" && python Task2_File_Operations.py
❶ File 'sample_output.txt' has been created in the current directory
```
- File Explorer:** Shows files like README\_Task2.md, README\_Task3.md, sample\_data.csv, sample\_output.txt, Task2\_File\_Operations.py, and Task3\_CSV\_Data\_Analysis.py.
- Bottom Right:** A notification about the Rainbow CSV extension.

## 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:

**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

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 @ 📸 🎨

**Step 1: Install Required Libraries**

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

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!")

10.1s Python
```

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

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 stated in the README for more details:

https://github.com/microsoft/generativeai/blob/main/README.md

Cursor Tab Spaces: 4 Cell 14 of 15

**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	58000	88000
Score	89.63	85	95

**To Use:**

- Upload the notebook to Google Colab
- Get your Gemini API key from Google AI Studio
- Replace YOUR\_API\_KEY\_HERE in the configuration cell
- 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 @ 📸 🎨

**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!")
```

0.0s Python

**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: \n{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)

Cursor Tab Spaces: 4 Cell 14 of 16

**CSV file statistical analysis**

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:

- Upload the notebook to Google Colab
- Get your Gemini API key from Google AI Studio
- Replace YOUR\_API\_KEY\_HERE in the configuration cell
- 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\_CS... +14 -6
- sample\_data.csv +9 -1
- README\_Task3.md +93 -1

Reject, suggest, follow up?

Auto

Task3\_CS... .ipynb

```
## Step 5: Traditional Statistical Analysis (Baseline)
First, let's calculate mean, min, and max using traditional methods for comparison.
```

```
# 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

===== TRADITIONAL STATISTICAL ANALYSIS =====

Column	Mean	Min	Max
Age	32.750	25	45
Salary	63750.000	50000	80000
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	3.113999
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

**CSV file statistical analysis**

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:

- Upload the notebook to Google Colab
- Get your Gemini API key from Google AI Studio
- Replace YOUR\_API\_KEY\_HERE in the configuration cell
- 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\_CS... +14 -6
- sample\_data.csv +9 -1
- README\_Task3.md +93 -1

Reject, suggest, follow up?

Auto

Task3\_CS... .ipynb

```
## Step 6: Gemini-Powered Analysis
Now, let's use Gemini to analyze the CSV data and calculate statistics.
```

```
# 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(df.select_dtypes(include=[np.number]).columns)}\n"

print("Data prepared for Gemini analysis")
```

0.0s

Data prepared for Gemini analysis

```
## Step 7: Final Output Summary
### Mean, Min, Max Values:
```

```
# 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

**OUTPUT:**

**CSV file statistical analysis**

**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

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

**Detailed Statistics:**

	Age	Salary	Score
count	8.000000	8.000000	8.000000
mean	32.750000	63750.000000	89.623909
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!**

## 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:

Search Agents... ... X

New Agent

Agents ... Bubble sort vs Python... 15m

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.

Exploded 3 files

Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.

...k4\_Sorting\_Comparison.py +243

Task 4: Sorting Lists – Manual vs Built-in

This program compares two sorting algorithms: Bubble Sort and Python's built-in sort().

README\_Task4.md (new) +196

# Task 4: Sorting Lists – Manual vs Built-in

This project compares two sorting algorithms: Bubble Sort and Python's built-in sort().

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 Assistant coding/AIC" && python Task4\_Sorting\_Comparison.py

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

Show Recommendations Install

Plan, @ for context, / for commands

Review

2 Files Undo All Keep All Review

AIC Cursor Tab G Ln 243, Col 1 Spaces: 4 UTF-8 LF (3) Python 3.9.6 ('venv': venv)

```
❸ Task4_Sorting_Comparison.py x AIC
❸ Task4_Sorting_Comparison.py > ...
1 import time
2 import random
3
4 def bubble_sort(arr):
5     """
6         Manual implementation of Bubble Sort algorithm.
7
8         Bubble Sort is a simple sorting algorithm that repeatedly steps through the list,
9         compares adjacent elements and swaps them if they are in the wrong order.
10        The pass through the list is repeated until the list is sorted.
11
12        Time Complexity: O(n^2) - Worst and Average case
13        Space Complexity: O(1) - In-place sorting
14
15    Args:
16        arr (list): List of comparable elements to sort
17
18    Returns:
19        list: Sorted list (original list is also modified in-place)
20
21    """
22        # Create a copy to avoid modifying the original list
23        arr = arr.copy()
24        n = len(arr)
25
26        # Outer loop: number of passes
27        for i in range(n):
28            # Flag to check if any swap occurred (optimization)
29            swapped = False
30
31            # Inner loop: compare adjacent elements
32            # After each pass, the largest element bubbles to the end
33            for j in range(n-i-1):
34                # If current element is greater than next, swap them
35                if arr[j] > arr[j+1]:
36                    arr[j], arr[j+1] = arr[j+1], arr[j]
37                    swapped = True
38
39            # If no swaps occurred, list is already sorted (optimization)
40            if not swapped:
41                break
42
43        return arr
44
45
46 def builtin_sort(arr):
47     """
48         Python's built-in sort() method.
49
50         Python's built-in sort() uses Timsort algorithm, which is a hybrid
51         stable sorting algorithm derived from merge sort and insertion sort.
52
53         Time Complexity: O(n log n) - Best, Average, and Worst case
54         Space Complexity: O(n) - Requires additional space
55
56     Args:
57         arr (list): List of comparable elements to sort
58
59     Returns:
60         list: Sorted list
61
62     """
63        # Create a copy and sort it
64        arr_copy = arr.copy()
65
66        arr_copy.sort()
67
68
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
139
```

Search Agents... ... X

New Agent

Agents ... Bubble sort vs Python... 15m

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.

Exploded 3 files

Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.

...k4\_Sorting\_Comparison.py +243

Task 4: Sorting Lists – Manual vs Built-in

This program compares two sorting algorithms: Bubble Sort and Python's built-in sort().

README\_Task4.md (new) +196

# Task 4: Sorting Lists – Manual vs Built-in

This project compares two sorting algorithms: Bubble Sort and Python's built-in sort().

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 Assistant coding/AIC" && python Task4\_Sorting\_Comparison.py

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

Show Recommendations Install

Plan, @ for context, / for commands

Review

2 Files Undo All Keep All Review

AIC Cursor Tab G Ln 243, Col 1 Spaces: 4 UTF-8 LF (3) Python 3.9.6 ('venv': venv)

```
❸ Task4_Sorting_Comparison.py x AIC
❸ Task4_Sorting_Comparison.py > ...
59 def builtin_sort(arr):
60     return arr_copy
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
```

**AIC**

Search Agents... New Agent

Agents Bubble sort vs Python's built-in... 16m +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.

Task 4: Sorting Lists – Manual vs E

This program compares two sorting a

# Task 4: Sorting Lists – Manual vs E

This project compares two sorting a

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 Assistant coding/AIC" && python Task4_Sorting_Comparison.py
==
```

> 2 Files Undo All Keep All Review

Plans, @ for context, / for commands

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

Show Recommendations Install

Cursor Tab Un 243, Col 1 Spaces: 4 UTF-8 LF () Python 3.9.6 ('venv', venv)

---

**AIC**

Search Agents... New Agent

Agents Bubble sort vs Python's built-in... 16m +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.

Task 4: Sorting Lists – Manual vs E

This program compares two sorting a

# Task 4: Sorting Lists – Manual vs E

This project compares two sorting a

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 Assistant coding/AIC" && python Task4_Sorting_Comparison.py
==
```

> 2 Files Undo All Keep All Review

Plans, @ for context, / for commands

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

Show Recommendations Install

Cursor Tab Un 243, Col 1 Spaces: 4 UTF-8 LF () Python 3.9.6 ('venv', venv)

## OUTPUT:

**Bubble sort vs Python's built-in...**

DEMONSTRATION: Sorting a Sample List

```
Original List: [64, 34, 25, 12, 22, 11, 98, 5]
Bubble Sort Result: [5, 11, 12, 22, 25, 34, 64, 98]
Built-in sort() Result: [5, 11, 12, 22, 25, 34, 64, 98]
```

Both methods produce identical results!

PERFORMANCE COMPARISON: Bubble Sort vs Built-in sort()

Array Size	Bubble Sort (s)	Built-in sort() (s)	Speedup
100	0.00039	0.00005	x
2000	0.00039	0.00005	x
3000	0.035540	0.000070	566.82
5000	0.946191	0.000370	2540.88

ALGORITHM COMPARISON SUMMARY

**BUBBLE SORT (Manual Implementation):**

- Algorithm Type: Simple comparison-based sorting
- Time Complexity:  $O(n^2)$  - Quadratic time
- Space Complexity:  $O(1)$  - Requires minimal space (in-place)
- Stability: Stable (equal elements maintain relative order)
- Best Case:  $O(n)$  - When array is already sorted
- Worst Case:  $O(n^2)$  - When array is reverse sorted
- Use Case: Educational purposes, very small datasets
- Advantages:
  - Simple to understand and implement
  - Stable sorting (no extra memory needed)
  - Stable sorting algorithm
- Disadvantages:
  - Very slow for large datasets
  - Not practical for real-world applications

**PYTHON'S BUILT-IN sort() (timsort):**

- Algorithm Type: Hybrid stable sorting (Merge + Insertion)
- Time Complexity:  $O(n \log n)$  - Linearithmic time
- Space Complexity:  $O(n)$  - Requires additional space
- Stability: Stable (equal elements maintain relative order)
- Best Case:  $O(n \log n)$
- Worst Case:  $O(n \log n)$
- Average Case:  $O(n \log n)$
- Use Case: Production code, real-world applications
- Advantages:
  - Extremely fast and efficient
  - Optimized for real-world data patterns
  - Handles various edge cases
  - Well-tested and reliable
- Disadvantages:
  - Uses more memory than in-place algorithms
  - Less educational value (implementation is hidden)

**KEY TAKEAWAY:**

For educational purposes, implementing Bubble Sort helps understand algorithmic thinking. However, for practical applications, always use Python's built-in sort() as it's significantly faster and more reliable.

Performance Difference:

- Built-in sort() is typically 100-1000x faster than Bubble Sort
- The difference becomes exponentially larger as dataset size increases

Task 4 completed successfully!

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

## 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.