

## 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 interface. The top bar displays the file name 'word\_frequency\_colab.ipynb', the lab number 'Lab 2: Task 1 - Word Frequency Counter', the task description, and a button to 'Display results'. Below the top bar are standard Jupyter navigation buttons: Code, Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, and a Python 3.9.6 dropdown. The main workspace contains two code cells. Cell [17] contains the following code:

```
# Import required libraries
import string
from collections import Counter
```

Cell [17] has a status bar indicating '0.0s' execution time and a Python 3.9.6 dropdown. Cell [18] contains the following code:

```
# 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!")
```

Cell [18] has a status bar indicating '0.0s' execution time and a Python 3.9.6 dropdown. The output of Cell [18] shows the message 'Sample text file created!'.

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 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:

The screenshot shows the AI interface with the following details:

- Search Agents...** and **New Agent** buttons.
- Agents** section: **File operations and CS...** 1m - +195 - Auto.
- Task2\_File\_Operations.py** file content:

```
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
13 def create_and_write_file(filename, content):
14     """
15     Creates a text file and writes content to it.
16
17     Args:
18         filename (str): Name of the file to create
19         content (str): Content to write to the file
20     """
21     try:
22         with open(filename, 'w', encoding='utf-8') as file:
23             file.write(content)
24             print(f"\u2708 Successfully created and wrote to '{filename}'")
25     except Exception as e:
26         print(f"\u270f Error writing to file: {e}")
27
28
29 def read_and_display_file(filename):
30     """
31     Reads a text file and displays its content.
32
33     Args:
34         filename (str): Name of the file to read
35     """
36     try:
37         with open(filename, 'r', encoding='utf-8') as file:
38             content = file.read()
39
40             print(f"\n{content}\n")
41             print(f"Content of '{filename}':")
42             print(f"\n{content}\n")
43             print(f"\n{content}\n")
44             print(f"\n{content}\n")
45
46     except FileNotFoundError:
47         print(f"\u270f Error: File '{filename}' not found.")
48     except Exception as e:
49         print(f"\u270f Error reading file: {e}")
50
51
52 def main():
53     """
```
- Review Next File** button.
- File operations and CSV data an...** folder view on the right.
- Auto-Ran command in** dropdown: **sandbox: cd, python**.
- \$ cd "/Users/hodla.manishwar/Downloads/AI Assistant coding/AIC" && python Task2\_File\_Operations.py** terminal command.
- File 'sample\_output.txt' has been created in the current directory.** status message.
- ✓ Success** status message.
- No linter errors found in Task2\_File\_Operations.py** message.
- Verifying the created file exists:** **Read sample\_output.txt** command.
- Creating a README for Task 2 to match Task 3's documentation style:** **> 2 Files Undo All Keep All Review** button.
- Plan, @ for context, / for commands** input field.
- Show Recommendations** and **Install** buttons at the bottom right.

The screenshot shows the AI Assistant interface with several tabs open:

- File operations and CSV data an...**: A search bar at the top left, followed by "New Agent" and "Agents". Below is a card for "File operations and CSV data an..." with a "Task 2: File Operations Using Cursor AI" section containing code and a "Task 2\_File\_Operations.py +90" snippet.
- Task2\_File\_Operations.py**: The main code editor tab, showing Python code for file operations. It includes comments explaining the program's purpose and demonstrating basic file operations like creating, writing to, reading from, and displaying files.
- Review Next File**: A tab labeled "AIC" with a list of files including "README\_Task2.md", "README\_Task3.md", "sample\_data.csv", "sample\_output.txt", "Task2\_File\_Operations.py", and "Task3\_CSV\_Data\_A\_....

At the bottom, there are navigation buttons for "2 Files", "Undo All", "Keep All", and "Review". A tooltip asks if you want to install "Rainbow CSV extension from mechatroner" for "sample\_data.csv".

### **OUTPUT:**

The screenshot shows a Jupyter Notebook interface with the following details:

- Terminal:** Displays Python command-line output for Task 2, including file creation and reading.
- File Operations and CSV Data Analysis:** A section titled "File operations and CSV data an..." containing code for file operations and a scenario description.
- Task 2: File Operations Using Cursor AI Scenario:** Describes creating a Python script for Task 2 that performs basic file operations: creates a text file, writes sample text, and reads and displays the content.
- Task 2 File Operations.py:** The code for the script, which creates a file named "sample\_output.txt" with the content "Hello, World! This is a sample text file."
- Output:** Shows the generated file "sample\_output.txt" and its content.
- Auto-Ran command in sandbox:** A terminal command to change directory to the AI Assistant coding folder and run the script.
- Verifying the created file exists:** A command to read the file "sample\_output.txt".
- Creating a README for Task 2 to match Task 3's documentation style:** A note about creating a README file.
- Review Bar:** Includes buttons for > 2 Files, Undo All, Keep All, and Review.
- Python Extension Recommendations:** A floating window asking if the user wants to install the "Rainbow CSV" extension from mechatroner for "sample\_data.csv".

## **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:

The screenshot shows a Google Colab notebook titled "Task3\_CSVDATA.ipynb". The notebook has two sections: "Step 1: Install Required Libraries" and "Step 2: Import Libraries and Setup".

**Step 1: Install Required Libraries**

```
# Install required packages
!pip install --quiet 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!")
```

The notebook also contains instructions for the user:

- 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.

At the bottom, there is a message about the deprecation of the 'google.generativeai' package:

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:

<https://github.com/google/generativeai/blob/main/DEPRECATION.md>

**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

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 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
gmail.configure(api_key=GEMINI_API_KEY)

print("Gemini API configured successfully!")
```

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("\nFirst few rows:")
display(df.head())
```

0.0s

CSV file loaded successfully!

Shape: (8, 4)

First few rows:

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

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 16

**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

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	32.750	25	45
Salary	63750.000	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

**## 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	89.630000
std	6.408699	9895.886591	3.113099
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 16

**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

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\_CSVDATA.ipynb +14 -6
- sample\_data.csv +9 -1
- README\_Task3.md +93 -1

Reject, suggest, follow up?

0.0s

**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.numeric_cols)}\n"

print("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'))
```

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

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\_CSVDATA.ipynb +14 -6
- sample\_data.csv +9 -1
- README\_Task3.md +93 -1

Reject, suggest, follow up?

0.0s

**Step 7: Final Output Summary**

FINAL STATISTICAL ANALYSIS - MEAN, MIN, MAX

---

Column	Mean	Min	Max
Age	32.75	26	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%	36.750000	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:

The screenshot shows the AI Code Editor interface with the following details:

- Left Sidebar:** Shows the project structure with files like `Task4_Sorting_Comparison.py`, `Task4_Sorting_Comparison.ipynb`, and `README_Task4.md`.
- Code Editor:** Displays the Python code for comparing bubble sort and Python's built-in `sort()` method.
- Right Sidebar:** Shows a file tree and a message asking if the user wants to install the 'Rainbow CSV' extension.
- Bottom Status Bar:** Provides information about the cursor, file, and system status.

```
Task4_Sorting_Comparison.py
12 import time
13 import random
14
15
16
17 def bubble_sort(arr):
18     """
19         Manual implementation of Bubble Sort algorithm.
20
21         Bubble Sort is a simple sorting algorithm that repeatedly steps through the list,
22         compares adjacent elements and swaps them if they are in the wrong order.
23         The pass through the list is repeated until the list is sorted.
24
25         Time Complexity: O(n^2) - Worst and Average case
26         Space Complexity: O(1) - In-place sorting
27
28     Args:
29         arr (list): List of comparable elements to sort
30
31     Returns:
32         list: Sorted list (original list is also modified in-place)
33
34     # Create a copy to avoid modifying the original list
35     arr = arr.copy()
36     n = len(arr)
37
38     # Outer loop: number of passes
39     for i in range(n):
40         # Flag to check if any swap occurred (optimization)
41         swapped = False
42
43         # Inner loop: compare adjacent elements
44         # After each pass, the largest element bubbles to the end
45         for j in range(0, n-i-1):
46             # If current element is greater than next, swap them
47             if arr[j] > arr[j + 1]:
48                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
49                 swapped = True
50
51             # If no swaps occurred, list is already sorted (optimization)
52             if not swapped:
53                 break
54
55     return arr
56
57
58 def builtin_sort(arr):
59     """
60         Python's built-in sort() method.
61
62         Python's built-in sort() uses Timsort algorithm, which is a hybrid
63         stable sorting algorithm derived from merge sort and insertion sort.
64
65         Time Complexity: O(n log n) - Best, Average, and Worst case
66         Space Complexity: O(n) - Requires additional space
67
68     Args:
69         arr (list): List of comparable elements to sort
70
71     Returns:
72         list: Sorted list
73
74     # Create a copy and sort it
75     arr_copy = arr.copy()
76     arr_copy.sort()
```

AIC

Search Agents... New Agent

Bubble sort vs Python's built-in... 15m +439 - Auto

Agents Bubble sort vs Python... 15m

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\_Comparison.py +243

Task 4: Sorting Lists – Manual vs E

This program compares two sorting a

README\_Task4.md (new) +196

# 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

Plan, @ for context, / for commands

Auto-Ran command in sandbox: cd python

\$ cd "/Users/bodla/manishwar/Downloads/AI Assistant coding/AIC" && python Task4\_Sorting\_Comparison.py

==

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("\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"\n{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.  
 """  
 sample\_list = [64, 34, 25, 12, 22, 11, 90, 51]  
  
 print("\n" + "\*70")  
 print("DEMONSTRATION: Sorting a Sample List")  
 print("\*70")  
  
 # Sample unsorted list  
 sample\_list = [64, 34, 25, 12, 22, 11, 90, 51]

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 UTT-B LF Python 3.9.6 (venv: venv)

AIC

Search Agents... New Agent

Bubble sort vs Python's built-in... 16m +439 - Auto

Agents Bubble sort vs Python... 16m

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\_Comparison.py +243

Task 4: Sorting Lists – Manual vs E

This program compares two sorting a

README\_Task4.md (new) +196

# 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

Plan, @ for context, / for commands

Auto-Ran command in sandbox: cd python

\$ cd "/Users/bodla/manishwar/Downloads/AI Assistant coding/AIC" && python Task4\_Sorting\_Comparison.py

==

def demonstrate\_sorting():  
 print("\nOriginal List: {sample\_list}")  
  
 # Bubble Sort  
 bubble\_sorted = bubble\_sort(sample\_list)  
 print("Bubble Sort Result: {bubble\_sorted}")  
  
 # Built-in Sort  
 builtin\_sorted = builtin\_sort(sample\_list)  
 print("Built-in sort() Result: {builtin\_sorted}")  
  
 # Verify both produce same results  
 if bubble\_sorted == builtin\_sorted:  
 print("X Both methods produce identical results!")  
 else:  
 print("X Results differ!")

def print\_comparison\_summary():  
 """  
 Print a summary comparison of both sorting methods.  
 """  
 print("\n" + "\*70")  
 print("ALGORITHM COMPARISON SUMMARY")  
 print("\*70")  
  
 comparison = ""  
 BUBBLE\_SORT (Manual Implementation):  
  
 • Algorithm Type: Simple comparison-based sorting  
 • Time Complexity: O(n<sup>2</sup>) – Quadratic time  
 • Space Complexity: O(1) – Constant space (in-place)  
 • Stability: Stable (equal elements maintain relative order)  
 • Best Case: O(n) – When array is already sorted  
 • Worst Case: O(n<sup>2</sup>) – When array is reverse sorted  
 • Average Case: O(n<sup>2</sup>)  
 • Use Case: Educational purposes, very small datasets  
 • Advantages:  
 - Easy to understand and implement  
 - In-place 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  
 - Used 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)

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 UTT-B LF Python 3.9.6 (venv: venv)

The screenshot shows the AI Code Editor interface with the following details:

- Left Panel:** Shows a tree view of project files:
  - Agents
  - Bubble sort vs Python... (selected)
  - Task 4: Sorting Lists – Manual vs Built-in
- Middle Panel:** The main code editor window displays `Task4_Sorting_Comparison.py`. The code compares Bubble Sort with Python's built-in `sorted()` function. It includes a `print_comparison_summary()` function for generating reports and a `main()` function for running the comparison. A terminal window at the bottom shows the command to run the script.
- Right Panel:** Shows a sidebar with file navigation and a "Review Next File" button.
- Bottom Bar:** Includes tabs for "Plan", "Review", and "Install", along with other standard UI elements like file operations and status indicators.

## OUTPUT:

The screenshot shows a Jupyter Notebook environment with several cells and a terminal window.

**Code Cells:**

- Bubble sort vs Python's built-in...**:
  - Original List: [64, 34, 25, 12, 22, 11, 90, 5]
  - Bubble Sort Result: [5, 11, 12, 22, 25, 34, 64, 90]
  - Built-in sort() Result: [5, 11, 12, 22, 25, 34, 64, 90]
  - Both methods produce identical results!
- 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.**
- Creating Task 4: implementing Bubble sort, comparing it with Python's built-in sort(), and adding a README.**
- Task 4: Sorting Lists - Manual vs Built-in**:
  - This program compares two sorting algorithms.
  - This project compares two sorting algorithms.
- README\_Task4.md (new) +196**
- # Task 4: Sorting Lists - Manual vs Built-in**:
  - This project compares two sorting algorithms.
- 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
```
- 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!**

**Terminal:**

```
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv? [Install]
```

**File Explorer:**

- AIC
- Python
- README\_Task2.md
- README\_Task3.md
- README\_Task4.md
- sample\_data.csv
- sample\_output.txt
- Task2\_File\_Operation.ipynb
- Task3\_CSV\_Data\_Analysis.ipynb
- Task4\_Sorting\_Comparison.ipynb

**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.