

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 Colab navigation buttons: 'Code' (selected), 'Markdown', 'Run All', 'Restart', 'Clear All Outputs', 'Jupyter Variables', 'Outline', and a three-dot menu. The Python version is listed as 'Python 3.9.6'. The main code cell (cell 17) contains the following Python code:

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

The output of cell 17 shows a blank line with a checkmark and '0.0s' execution time. The next cell (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!")
```

The output of cell 18 shows the message 'Sample text file created!' followed by a blank line with a checkmark and '0.0s' execution time. The sidebar on the right shows 'LAB@' and 'LAB'.

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 API

❖ 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 an AI-powered code editor interface. On the left, a sidebar displays 'File operations and CSV data analysis' and 'Task 2: File Operations Using Cursor AI'. The main workspace contains two tabs: 'Task2_File_Operations.py' and 'Task 2: File Operations Using Cursor AI'. The code in 'Task2_File_Operations.py' demonstrates basic file operations (writing to and reading from a text file). The code in 'Task 2: File Operations Using Cursor AI' is a generated AI version of the same script. A status bar at the bottom indicates the cursor is on line 7, column 34. A floating window on the right asks if the user wants to install the 'Rainbow CSV' extension.

```
def Task2_File_Operations.py > ...
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
22     try:
23         with open(filename, 'w', encoding='utf-8') as file:
24             file.write(content)
25         print(f"\u2708 Successfully created and wrote to '{filename}'")
26     except Exception as e:
27         print(f"\u270d Error writing to file: {e}")
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
37     try:
38         with open(filename, 'r', encoding='utf-8') as file:
39             content = file.read()
40
41             print(f"\n{s*40}")
42             print(f"Content of '{filename}':")
43             print(f"\t{s*40}")
44             print(content)
45             print(f"\t{s*40}\n{n*40}")
46
47     except FileNotFoundError:
48         print(f"\u270d Error: File '{filename}' not found.")
49     except Exception as e:
50         print(f"\u270d Error reading file: {e}")
51
52 def main():
53     """
```

The screenshot shows the AI Coding interface with a Python script named `Task2_File_Operations.py`. The code demonstrates basic file operations: creating a file, writing to it, and reading its content. It includes comments explaining the steps and a main function that prints a success message. The interface also shows a terminal window with the command to run the script and a message indicating success. A sidebar on the right lists related files like `README_Task2.md` and `sample_data.csv`.

```
def main():
    """
    Main function to execute file operations.
    """
    # File name
    filename = "sample_output.txt"

    # Sample text content
    sample_text = """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!"""

    print("="*60)
    print("Task 2: File Operations Using Cursor AI")
    print("="*60)
    print("\nStep 1: Creating and writing to file...")
    create_and_write_file(filename, sample_text)

    print("\nStep 2: Reading and displaying file content...")
    read_and_display_file(filename)

    print("All file operations completed successfully!")
    print(f"File '{filename}' has been created in the current directory.")

if __name__ == "__main__":
    main()
```

OUTPUT:

The screenshot shows the AI Coding interface with the same Python script running in the terminal. The output confirms that the file was created and contains the expected sample text. The interface also shows a message about installing the 'Rainbow CSV' extension.

```
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"
(base) bodla.manishwar@BodlaManishwar-Laptop AIC % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
(base) bodla.manishwar@BodlaManishwar-Laptop AIC % "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/User
s/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task2_File_Operations.py"
=====
Task 2: File Operations Using Cursor AI
=====

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
=====

All file operations completed successfully!
File 'sample_output.txt' has been created in the current directory.
(.venv) (base) bodla.manishwar@BodlaManishwar-Laptop AIC %
```

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 Assistant interface with the following details:

- Project Overview:** Bubble sort vs Python's built-in...
- Task Details:** Task 4: Sorting Lists – Manual vs Built-in
- Description:** You are reviewing algorithm choices for efficiency.
- Scenarios:** Use Gemini to generate:
 - > Bubble sort
 - > Python's built-in sort()
 - > Compare both implementations.
- Expected Output:**
 - > Two versions of code
 - > Short comparison
- Code Snippet:** The main code block contains two implementations of sorting: a manual bubble sort and Python's built-in `sort()`. The manual sort is annotated with comments explaining its steps and complexity. The Python built-in sort is noted as using Timsort, a hybrid stable sorting algorithm.
- File Structure:** Shows files like `Task4_Sorting_Comparison.py`, `README_Task4.md`, and `sample_data.csv`.
- Toolbox:** Includes options for Auto-Ran command, sandbox, and file operations.
- Bottom Bar:** Shows the cursor tab, file number, line number, and other development tools.

AIC

Review Next File

File Explorer

Search Agents...

New Agent

Agents

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.

.../x4_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

==

==

> 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):

"""

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.

"""

print("\n\n")

print("=*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 UTT-B LF Python 3.9.6 (venv: venv)

AIC

Review Next File

File Explorer

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.

.../x4_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

==

==

> 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 the original 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("=*70 Both methods produce identical results!")

else:

print("\nX 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(n²) – 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(n²) – When array is reverse sorted

• Average Case: O(n²)

• Use Cases: 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 Cases: Production code, real-world applications

• Advantages:

- Extremely fast and efficient
- Suitable 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 the project structure and files:
 - Search Agents...
 - New Agent
 - Agents
 - Bubble sort vs Python... (selected)
- 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 sections for printing comparison summary, demonstrating with small examples, and performing a performance comparison.
- Bottom Panel:** Shows the terminal output of running the script, which prints a success message: "Task 4 completed successfully!".
- Right Panel:** Shows the file tree and status bar with "AIC" and "Review Next File".
- Status Bar:** Shows the cursor position (Ln 243, Col 1), spaces used (Spaces: 4), and other system information.

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.