

ASSIGNMENT-02

NAME: A. Supreeth reddy

Hall Ticket:2303A51392

Batch:06

Q) Task 1: Word Frequency from Text File

❖ Scenario:

You are analyzing log files for keyword frequency.

❖ Task:

Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.

❖ Expected Output:

➢ Working code

➢ Explanation ➤ Screenshot

Solution:

PROMPT

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

CODE:

The screenshot shows a Google Colab notebook titled "word_frequency_colab.ipynb". The code cell at the top imports string and Counter modules. The main code cell (cell 17) contains a multi-line string with sample text about Python's capabilities in data science, machine learning, and artificial intelligence. It then saves this text to a file named "sample_text.txt" and prints a confirmation message. The output cell (cell 18) shows the message "Sample text file created!". The interface includes a toolbar with Code, Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, and a Python 3.9.6 kernel selector.

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

# 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!")
```

```

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 a Python code editor with the following details:

- File operations and CSV data analysis** tab is active.
- Task 2: File Operations Using Cursor AI** scenario is selected.
- Code Editor Content:**

```
Task2_File_Operations.py
1m - +195 Auto

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!
Date: Generated using Cursor AI
"""

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()
```
- File Browser:** Shows files for Task 2 and Task 3, including README_Task2.md, sample_data.csv, sample_output.txt, Task2_File_Operations.py, and Task3_CSV_Data_Analysis.py.
- Bottom Status Bar:** Shows file count (2), undo/redo, keep all, review, and a message about installing the 'Rainbow CSV' extension.

OUTPUT:

The screenshot shows a Python code editor interface with several tabs and panels:

- Top Bar:** Problems, Output, Debug Console, Terminal, Ports, Jupyter, Python.
- Left Sidebar:** Search Agents, New Agent, Agents, File operations and CSV data analysis.
- Middle Left Panel:** Task 2: File Operations Using Cursor AI. Scenario: Creating a program for Task 2 that creates a text file, writes sample text, and reads and displays the content. Includes a code editor for `Task2_File_Operations.py` and a terminal output showing the creation of `sample_output.txt`.
- Middle Right Panel:** Task 2: File Operations Using Cursor AI. Scenario: 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. Includes a code editor for `Task2_File_Operations.py` and a terminal output showing the creation of `sample_output.txt`.
- Bottom Right Panel:** A message from the AI asking if the user wants to install 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

Task3_CSV_Data_An... +14 -6

sample_data.csv +9 -1

README_Task3.md +93 -1

Reject, suggest, follow up?

Auto · Review

Step 1: Install Required Libraries

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

25.2s

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

```
/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/google/api_core/_python/_version.py:35: Not
warnings.warn(
/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/urllib3/_init_.py:54:
warnings.warn(el.message.format("3.9"), FutureWarning)
/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/google/auth/_init_.py:4
warnings.warn(el.message.format("3.9"), FutureWarning)
Warnings imported successfully!
/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/tqdm/auto.py:21: TqdmWarnin
from .autonotebook import tqdm as notebook_tqdm
/var/folders/2j/c3s_km1515dcmydtv30y4gc000gn/T/ipynbkernel_60221/4255226943.nvs3: FutureWarning:
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?
See README for more details:
```

Show Recommendations | Install

<https://github.com/google-research/degenerated-generativeai-python/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	63758.00	58000	88000
Score	89.63	85	95

To Use:

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

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

3 Files

Task3_CSV_Data_An... +14 -6

sample_data.csv +9 -1

README_Task3.md +93 -1

Reject, suggest, follow up?

Auto · Review

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

Gemini API configured successfully!

Step 4: Upload CSV File

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

```
# Read the CSV file
csv_file = "sample_data.csv" # Change this to your uploaded file name

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

df = pd.read_csv(csv_file)

print("CSV file loaded successfully!")
print(f"\nShape: {df.shape}")
print(f"\nFirst few rows:\n{df.head()}")
display(df.head())
```

CSV file loaded successfully!

Shape: (8, 4)

First few rows:

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

```
## Step 5: Traditional Statistical Analysis (Baseline)
```

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

Show Recommendations | Install

Cursor Tab · Spaces: 4 · Cell 14 of 16

CSV file statistical analysis

16m +624 Auto

Task 3: CSV Data Analysis

◆ Scenario:
You are processing structured data from a CSV file.

◆ Task:
Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

◆ Expected Output:
Correct output
Screenshot

```
Salary 63750.00 50000 80000
Score 89.63 85 95
```

To Use:

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

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

3 Files

- Task3_CSV_Data_An... +14 -6
- sample_data.csv +0 -1
- README_Task3.md +93 -1

Reject, suggest, follow up?

Auto Review

Keep Cell Undo Cell Keep 3N

```
# 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())
```

Column	Mean	Min	Max
0 Age	32.750	25	45
1 Salary	63750.000	50000	80000
2 Score	89.625	85	95

Detailed Statistics:

	Age	Salary	Score
count	8.000000	8.000000	8.000000
mean	32.750000	63750.000000	89.625000
std	6.408699	9895.886591	3.113099
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

CSV file statistical analysis

16m +624 Auto

Task 3: CSV Data Analysis

◆ Scenario:
You are processing structured data from a CSV file.

◆ Task:
Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

◆ Expected Output:
Correct output
Screenshot

```
Salary 63750.00 50000 80000
Score 89.63 85 95
```

To Use:

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

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

3 Files

- Task3_CSV_Data_An... +14 -6
- sample_data.csv +0 -1
- README_Task3.md +93 -1

Reject, suggest, follow up?

Auto Review

Keep Cell Undo Cell Keep 3N

```
# 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.to_string()
data_summary = f"\nData shape: {df.shape}\n"
data_summary += f"Columns: {list[Any](df.columns)}\n"
data_summary += f"Numeric columns: {list[Any](numeric_cols)}\n"

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

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("\n" + "=" * 70)
display(df[numeric_cols].describe())

print("\n" + "=" * 70)
print("ANALYSIS COMPLETE!")
print("\n" + "=" * 70)
```

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

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.

Code:

```
Task3_CSV_Data_Analysis.ipynb
```

Output:

Column	Mean	Min	Max
Age	32.76	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	6.408659	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

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 Assistant interface with the following details:

- Left Panel:** Shows a search bar for "Agents..." and a "New Agent" button. Below it, a section titled "Bubble sort vs Python's built-in..." is displayed, containing a scenario about sorting lists and comparing manual vs built-in implementations.
- Middle Panel:** A code editor for `Task4_Sorting_Comparison.py`. The code implements bubble sort and Python's built-in `sort()` function. It includes comments explaining the algorithm, time complexity ($O(n^2)$), and space complexity ($O(1)$). The code editor has syntax highlighting and a status bar indicating 15m +439 Auto.
- Bottom Panel:** A terminal window showing the output of running the script. It includes the command used, the file path, and the results of the comparison.
- Right Panel:** A sidebar titled "Review Next File" showing a list of files: `AIC`, `venv`, `README_Task2.md`, `README_Task3.md`, `README_Task4.md`, `sample_data.csv`, `sample_output.txt`, `Task2_File_Operation.py`, `Task3_CSV_Data_Analysis.py`, and `Task4_Sorting_Com...`.
- Bottom Right:** A modal dialog asking if the user wants to install the "Rainbow CSV" extension for `sample_data.csv`, with "Install" and "Show Recommendations" buttons.

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

- Search Agents...**: A dropdown menu.
- New Agent**: A button.
- Agents**: A section showing "Bubble sort vs Python's built-in..." and "Task 4: Sorting Lists - Manual vs Built-in...".
- Bubble sort vs Python's built-in...**: A cell containing code for comparing bubble sort and Python's built-in sort. It includes imports, function definitions, and a main loop for testing array sizes.
- Task 4: Sorting Lists - Manual vs Built-in...**: A cell containing code for demonstrating sorting methods. It includes imports, function definitions, and a main loop for testing array sizes.
- Review**: A button at the bottom of the code cells.
- Plan, for context, / for commands**: A button.
- Auto**: A dropdown menu.
- Review Next File**: A button.
- AIC**: A tab at the top right.
- File**, **Cell**, **Kernel**, **Help**: Standard Jupyter Notebook menu items.
- Review Recommendations**: A button at the bottom right.
- Install**: A button at the bottom right.

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.

This project compares two sorting algorithms:

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_Comparison.py

Running the program to verify it works:

```
Auto-Ran command in sandbox: cd; python $ cd "/Users/bodala.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

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.

This project compares two sorting algorithms:

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_Comparison.py

Running the program to verify it works:

```
Auto-Ran command in sandbox: cd; python $ cd "/Users/bodala.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

OUTPUT:

The screenshot shows an AI Assistant interface with several tabs and sections:

- Search Agents...**: A search bar at the top left.
- New Agent**: A button to create a new agent.
- Agents**: A list of agents, with one named "Bubble sort vs Python's built-in..." selected.
- Bubble sort vs Python's built-in...**: The main content area. It includes:
 - Task 4: Sorting Lists – Manual vs Built-in**
 - DEMONSTRATION: Sorting a Sample List**: Shows original and sorted lists for both methods.
 - PERFORMANCE COMPARISON: Bubble Sort vs Built-in sort()**: A table comparing execution times for array sizes 100, 500, 1000, and 5000.
 - ALGORITHM COMPARISON SUMMARY**: Summarizes the differences between Bubble Sort and Python's built-in sort().
 - BUBBLE SORT (Manual Implementation)**: Details the algorithm's properties and use cases.
 - PYTHON'S BUILT-IN sort() (timsort)**: Details the algorithm's properties and use cases.
 - KEY TAKEAWAY**: A summary of the learning points.
 - Performance Difference**: Notes that built-in sort() is faster than Bubble Sort.
 - Task 4 completed successfully!**: Confirmation message.
- Terminal**: Shows the command history and output of the run command.
- Output**: Shows the results of the task execution.
- Debug Console**: Shows the environment variables and command history.
- Problems**: Shows any errors or warnings.
- Ports**: Shows port information.
- Jupyter**: Shows Jupyter notebook files.
- Python**: A sidebar on the right showing the project structure, including files like README_Task2.md, README_Task3.md, README_Task4.md, sample_data.csv, sample_output.txt, Task2_File_Operation.ipynb, Task3_CSV_Data_Analysis.ipynb, and Task4_Sorting_Comparison.ipynb.
- File Explorer**: Shows the file system structure.
- Terminal**: Shows the command history and output of the run command.
- Output**: Shows the results of the task execution.
- Debug Console**: Shows the environment variables and command history.
- Problems**: Shows any errors or warnings.
- Ports**: Shows port information.
- Jupyter**: Shows Jupyter notebook files.

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.