

ASSIGNMENT-02

NAME: G.Rahul

Hall Ticket:2303A51100

Batch:05

Q) Task 1: Word Frequency from Text File

❖ Scenario:

You are analyzing log files for keyword frequency.

❖ Task:

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

❖ Expected Output:

➢ Working code

➢ Explanation ➢ Screenshot

Solution:

PROMPT

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

CODE:

The screenshot shows a Google Colab notebook titled "word_frequency_colab.ipynb". The code cell at the top imports string and Counter, and then defines a sample_text variable containing a paragraph about Python's use in data science. The code then saves this text to a file named "sample_text.txt" and prints a confirmation message. The output cell below shows the confirmation message "Sample text file created!".

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

# Create or upload a sample text file
# You can either upload a file or create one programmatically
sample_text = """Python is a powerful programming language. Python is widely used for data science. Many developers love Python because it is easy to learn. Python has excellent libraries for machine learning and artificial intelligence.

Data science requires Python skills. Machine learning projects often use Python. Python is versatile and can be used for web development, automation, and data analysis.

The Python community is large and supportive. Python code is readable and clean. Many universities teach Python as the first programming language.

In this lab, we explore Python. We use Python for analysis. Python makes coding fun and efficient. The future of programming includes Python. Learning Python opens many opportunities for developers."""

# 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!")
[18] ✓ 0.0s
... 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 the AI Assistant interface with the following details:

- 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 "File operations and CSV data analysis..." currently selected.
- File operations and CSV data analysis...**: The selected agent's details. It includes:
 - Task 2: File Operations Using Cursor AI**: A scenario description.
 - Creating a program for Task 2 that creates a text file, writes sample text, and reads and displays the content.**
 - Explored 1 directory 1 file**: A file exploration summary.
 - 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.**
 - Task2_File_Operations.py +90**: A code editor window showing the generated Python script. The code defines two functions: `create_and_write_file` and `read_and_display_file`. The `create_and_write_file` function takes a filename and content, creates the file, and writes the content. The `read_and_display_file` function takes a filename, reads its content, and prints it. Both functions include docstrings and try-except blocks for handling errors.
 - Running the script to verify it works correctly:** A command-line terminal window showing the execution of the script and the creation of a file named `sample_output.txt`.
 - No linter errors found in Task2_File_Operations.py**: A message indicating code quality.
 - Verifying the created file exists:** A message confirming the file was created.
 - Create a README for Task 2 to match Task 3's documentation style:** A task suggestion.
 - Plan, ⌂ for context, / for commands**: Navigation and search buttons.
 - Review Next File**: A button to review the next file in the project.
 - Review Recommendations**: A button to see recommended AI features.
 - Install Rainbow CSV extension**: A call-to-action for installing a CSV extension.

The screenshot shows a dual-pane code editor interface. The left pane displays a Python script for file operations, and the right pane shows a CSV analysis report.

Left Pane: Task2_File_Operations.py

```
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()
```

Right Pane: File operations and CSV data analysis

Review Next File

AIC

File operations and CSV data analysis

Task 2: File Operations Using Cursor AI

Scenario:

You are automating basic file operations for Task 2. You need to create a text file, write sample text, and read its content.

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.

Explored 1 directory 1 file

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 Using Cursor AI

This program demonstrates basic file operations: creating a file, writing content, reading content, and displaying it.

Running the script to verify it works correctly:

```
Auto-Ran command in sandbox: cd, python Task2_File_Operations.py
```

File 'sample_output.txt' has been created in the current directory.

No linter errors found in Task2_File_Operations.py

Verifying the created file exists:

Read sample_output.txt

Creating a README for Task 2 to match Task 3's documentation style:

> 2 Files Undo All Keep All Review

Plan @ for context, / for commands

Auto

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

Cursor Tab Ln 7, Col 34 Spaces: 4 UTF-8 LF Python 3.9.6 (.venv: venv) ↻

OUTPUT:

The screenshot shows a Jupyter Notebook environment with several tabs open. The active tab is titled "File operations and CSV data an...". The code in the cell is as follows:

```

source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
"/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task2_File_Operations.py"
❶ (base) bodla.manishwar@BodlaManishwars-Laptop AIC % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
❷ (.venv) (base) bodla.manishwar@BodlaManishwars-Laptop AIC % "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/task2_File_operations.py"
Task 2: File Operations Using Cursor AI
```

The output shows the execution of the script, which creates a file named "sample_output.txt" containing the text "Hello, World! This is a sample text file.", reads it back, and displays the content on the screen. The program concludes with a success message.

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 Jupyter Notebook interface with two main sections:

CSV file statistical analysis

This section contains a code cell for reading a CSV file and calculating statistics:

```
# 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
```

Below the code cell is a note: "The notebook is ready to use. Run it in Colab and take a screenshot of the final output showing the statistical analysis results."

File list:

- Task3_CS... +14 -6
- sample_data.csv +9 +1
- README_Task3.md +93 -1

Bottom status bar: Reject, suggest, follow up? Auto

Task3_CS... •

Task 3: CSV Data Analysis with Gemini > Step 7: Final Output Summary

Code | Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | .venv (Python 3.9.6)

Step 1: Install Required Libraries

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

25.2s Python

Step 2: Import Libraries and Setup

```
import pandas as pd  
import numpy as np  
import google.generativeai as genai  
import os  
from IPython.display import display, HTML  
  
print("Libraries imported successfully!")
```

10.1s Python

```
...  
/Users/bodla/manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/google/api_core/_python_warnings.warn(message, FutureWarning)  
/Users/bodla/manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/urllib3/_init_.py:35: No  
warnings.warn(  
/Users/bodla/manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/google/auth/_init_.py:54:  
warnings.warn(eol_message.format("3.9"), FutureWarning)  
/Users/bodla/manishwar/Downloads/AI Assistant coding/AIC/.venv/lib/python3.9/site-packages/google/oauth2/_init_.py:  
warnings.warn(eol_message.format("3.9"), FutureWarning)  
Libraries 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/2/c3s_km151dcmydtw30y4gc000gn/T/ipykernel_60221/4255226943.n:3: FutureWarning:  
Do you want to install the recommended 'Rainbow CSV' extension from mechatroner for sample_data.csv?  
Show Recommendations | Install
```

All support for the 'google.generativeai' package has ended. It will no longer receive updates or bug fixes. Please switch to the 'google.genai' package as it is no longer maintained.

See README for more details:

The screenshot shows a Google Colab notebook titled "CSV file statistical analysis". The notebook contains several sections:

- Task 2: 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
 - Ready for Colab — can be uploaded and run directly

The notebook then displays a table of statistics:

Column	Mean	Min	Max
Age	32.75	25	45
Salary	63758.88	58000	80000
Score	89.63	85	95

To Use:

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

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

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!")
```

Step 4: Upload CSV File

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

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

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

df = pd.read_csv(csv_file)

print("CSV file loaded successfully!")
print(f"Shape: {df.shape}")
print(f"First few rows:")
display(df.head())
```

Step 5: Traditional Statistical Analysis (Baseline)

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

CSV file statistical analysis

You are processing structured data from a CSV file.

Task:

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

Expected Output:

- Correct output
- Screenshot

Salary	63750.00	50000	80000
Score	89.63	85	95

To Use:

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

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

3 Files

- Task3_CS... +14 -6
- sample_data.csv +9 -1
- README_Task3.md +83 -1

Reject, suggest, follow up?

Auto

Keep Cell

First, let's calculate mean, min, and max using traditional methods for comparison.

```
# Step 5: Traditional Statistical Analysis (Baseline)
# Calculate statistics for numeric columns only
numeric_cols = df.select_dtypes(include=[np.number]).columns

print("=" * 60)
print("TRADITIONAL STATISTICAL ANALYSIS")
print("=" * 60)

stats_df = pd.DataFrame({
    'Column': numeric_cols,
    'Mean': [df[col].mean() for col in numeric_cols],
    'Min': [df[col].min() for col in numeric_cols],
    'Max': [df[col].max() for col in numeric_cols]
})

display(stats_df)

print("\nDetailed Statistics:")
print(df[numeric_cols].describe())

```

0.0s

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

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

Detailed Statistics:

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

You are processing structured data from a CSV file.

Task:

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

Expected Output:

- Correct output
- Screenshot

Salary	63750.00	50000	80000
Score	89.63	85	95

To Use:

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

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

3 Files

- Task3_CS... +14 -6
- sample_data.csv +9 -1
- README_Task3.md +93 -1

Reject, suggest, follow up?

Auto

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(numeric_cols)}\n"

print("Data prepared for Gemini analysis")

```

0.0s

Data prepared for Gemini analysis

Step 7: Final Output Summary

Mean, Min, Max Values:

```
# Final comprehensive summary
print("=" * 70)
print("FINAL STATISTICAL ANALYSIS - MEAN, MIN, MAX")
print("=" * 70)

final_stats = pd.DataFrame({
    'Column': numeric_cols,
    'Mean': [round(df[col].mean(), 2) for col in numeric_cols],
    'Min': [df[col].min() for col in numeric_cols],
    'Max': [df[col].max() for col in numeric_cols]
})

# Display with better formatting
display(HTML(final_stats.to_html(index=False, classes='table table-striped')))

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

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

11s

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

OUTPUT:

CSV file statistical analysis

Task 3: CSV Data Analysis

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

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

◆ Expected Output:
➢ Correct output
➢ Screenshot

Column	Mean	Min	Max
Age	32.75	25	45
Salary	63750.00	50000	80000
Score	89.62	85	95

To Use:

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

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

Detailed Statistics:

	Age	Salary	Score
count	8.000000	8.000000	8.000000
mean	32.750000	63750.000000	89.623909
std	6.408699	9895.886591	3.113909
min	25.000000	50000.000000	85.000000
25%	28.750000	57250.000000	87.750000
50%	31.000000	62500.000000	89.500000
75%	35.750000	70500.000000	91.250000
max	45.000000	80000.000000	95.000000

ANALYSIS COMPLETE!

CODE EXPLANATION:

This code performs statistical analysis on numeric columns of a DataFrame (df). First, it identifies all columns that contain numerical data using `select_dtypes(include=[np.number])`. Then, for each numeric column, it calculates the mean, minimum, and maximum values and stores them in a new DataFrame called `stats_df`. This DataFrame is displayed to show a clean summary of basic statistics.

Q) Task 4: Sorting Lists – Manual vs Built-in

❖ Scenario:

You are reviewing algorithm choices for efficiency.

❖ Task:

Use Gemini to generate:

➢ Bubble sort

➢ Python's built-in `sort()`

➢ Compare both implementations.

❖ Expected Output:

➢ Two versions of code

➢ Short comparison

PROMPT:

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

CODE:

The screenshot shows a code editor interface with multiple tabs and panels. The main panel displays a Python script named `Task4_Sorting_Comparison.py`. The script compares bubble sort with Python's built-in `sort()` function. It includes comments explaining the bubble sort algorithm and its time complexity. The Python code defines two functions: `bubble_sort` and `builtin_sort`, both taking a list as input and returning a sorted list. The code editor has syntax highlighting and a status bar at the bottom.

```
import time
import random

def bubble_sort(arr):
    """
    Manual implementation of Bubble Sort algorithm.
    Bubble Sort is a simple sorting algorithm that repeatedly steps through the list,
    compares adjacent elements and swaps them if they are in the wrong order.
    The pass through the list is repeated until the list is sorted.

    Time Complexity: O(n^2) - Worst and Average case
    Space Complexity: O(1) - In-place sorting
    """

    args:
        arr (list): List of comparable elements to sort

    Returns:
        list: Sorted list (original list is also modified in-place)
        ....
    # Create a copy to avoid modifying the original list
    arr = arr.copy()
    n = len(arr)

    # Outer loop: number of passes
    for i in range(n):
        # Flag to check if any swap occurred (optimization)
        swapped = False

        # Inner loop: compare adjacent elements
        # After each pass, the largest element bubbles to the end
        for j in range(0, n - i - 1):
            # If current element is greater than next, swap them
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # If no swaps occurred, list is already sorted (optimization)
        if not swapped:
            break

    return arr

def builtin_sort(arr):
    """
    Python's built-in sort() method.

    Python's built-in sort() uses Timsort algorithm, which is a hybrid
    stable sorting algorithm derived from merge sort and insertion sort.

    Time Complexity: O(n log n) - Best, Average, and Worst case
    Space Complexity: O(n) - Requires additional space

    Args:
        arr (list): List of comparable elements to sort

    Returns:
        list: Sorted list
        ....
    # Create a copy and sort it
    arr_copy = arr.copy()
    arr_copy.sort()

    return arr_copy
```

The screenshot shows an AI-powered code editor interface with the following details:

- Search Agents...**: A dropdown menu for managing agents.
- New Agent**: A button to create a new agent.
- Agents**: A list of agents, with "Bubble sort vs Python..." currently selected.
- Bubble sort vs Python's built-in...**: The active file, showing its content and a preview of the output.
- Task 4: Sorting Lists – Manual vs Built-in**: A section describing the task, mentioning the goal of comparing two implementations of bubble sort and adding a README.
- Task 4: Implementing Bubble Sort**: A section detailing the implementation of bubble sort in Python.
- Task 4: Comparing Performance**: A section detailing the comparison of bubble sort with Python's built-in sort function.
- Task 4: Generating Data**: A section detailing the generation of test data for the comparison.
- Task 4: Generating Report**: A section detailing the generation of a report comparing the results.
- Task 4: Conclusion**: A section concluding the task.
- Review Next File**: A button to review the next file in the list.
- File Explorer**: Shows the project structure with files like `Task4_Sorting_Comparison.py`, `README_Task4.md`, `sample_data.csv`, and `sample_output.txt`.
- Code Editor**: The main area where the Python code for the sorting comparison is written.
- Output Terminal**: The bottom pane showing the command-line history and the resulting CSV file content.

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

- Left Sidebar:** Shows the project structure with files like `Bubble sort vs Python's built-in...`, `Task4_Sorting_Comparison.py`, and `README_Task4.md`.
- Code Editor:** The main area displays `Task4_Sorting_Comparison.py`. The code compares bubble sort with Python's built-in sort, including a summary function and a main loop for running tests.
- Bottom Status Bar:** Shows file counts (2 Files), undo/redo, keep all, and a review button.
- Bottom Navigation:** Includes tabs for AIC, AI, and AI+, along with other navigation icons.
- Bottom Right:** A floating window asks if the user wants to install the "Rainbow CSV" extension for `sample_data.csv`.

OUTPUT:

The screenshot shows the AI Assistant Coding (AIC) interface. The main area displays a terminal session comparing the execution times of Bubble Sort and Python's built-in `sort()` function. The terminal output includes:

```

DEMONSTRATION: Sorting a Sample List
=====
Original List: [64, 34, 25, 12, 22, 11, 98, 5]
Bubble Sort Result: [5, 11, 12, 22, 25, 34, 64, 98]
Built-in sort() Result: [5, 11, 12, 22, 25, 34, 64, 98]

Both methods produce identical results!

PERFORMANCE COMPARISON: Bubble Sort vs Built-in sort()
=====
Array Size     Bubble Sort (s)    Built-in sort() (s)   Speedup
100           0.00039            0.000005          x
200           0.00079            0.000009          x
300           0.00159            0.000018          x
400           0.003540           0.0000370         x
500           0.004191           0.0000570         x

ALGORITHM COMPARISON SUMMARY
=====
BUBBLE SORT (Manual Implementation):
=====
  • Algorithm Type: Simple comparison-based sorting
  • Time Complexity:  $O(n^2)$  - Quadratic time
  • Space Complexity:  $O(1)$  - Requires minimal space (in-place)
  • Stability: Stable (equal elements maintain relative order)
  • Best Case:  $O(n)$  - When array is already sorted
  • Worst Case:  $O(n^2)$  - When array is reverse sorted
  • Average Case:  $O(n^2)$ 
  • Use Case: Educational purposes, very small datasets
  • Advantages:
    - Simple 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:
    - Efficient and fast
    - Optimized for real-world data patterns
    - Handles various edge cases
    - Well-tested and reliable
  • Disadvantages:
    - Uses more memory than in-place algorithms
    - Less educational value (implementation is hidden)

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

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

Task 4 completed successfully!
=====
```

The terminal also shows a file browser on the right side, listing files related to Task 4 and previous tasks.

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.