

Assignment-2.3

K.Harinisri
2303A51602

TASK-1

Task 1: Word Frequency from Text File

❖ Scenario:

You are analyzing log files for keyword frequency.

❖ Task:

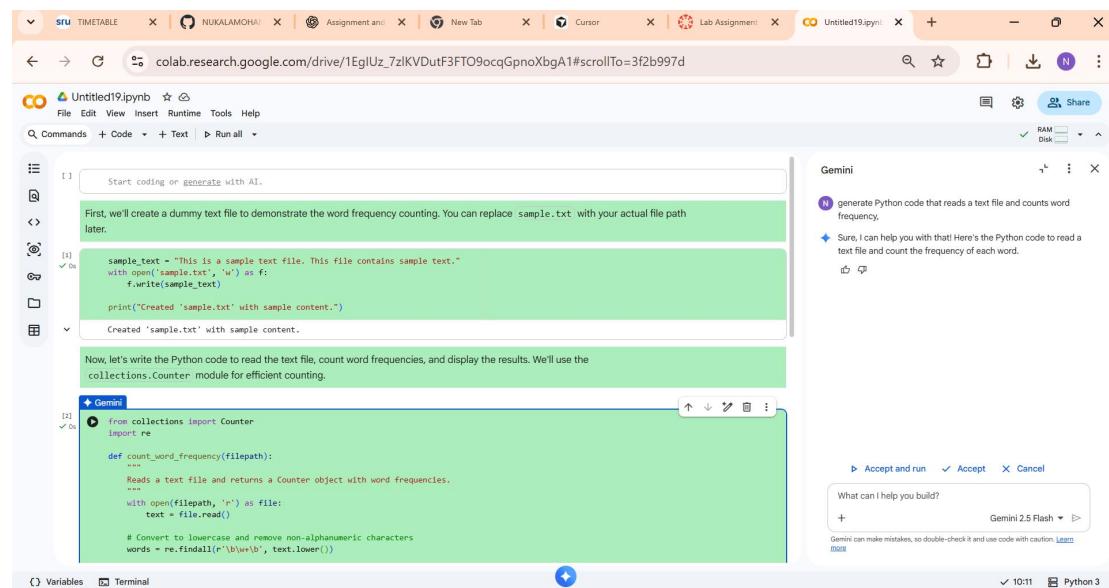
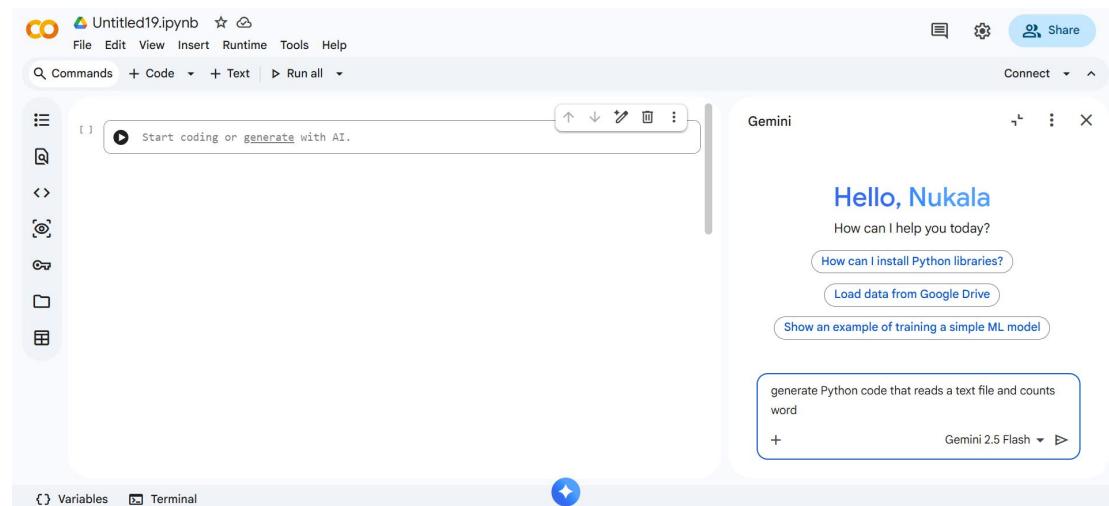
Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.

❖ Expected Output:

➤ Working code

➤ Explanation

➤ Screenshot



The screenshot shows a Google Colab notebook titled "Untitled19.ipynb". The code cell contains the following Python script:

```

# Convert to lowercase and remove non-alphanumeric characters
words = re.findall(r'\b\w+\b', text.lower())
return Counter(words)

# Specify the path to your text file
file_path = 'sample.txt' # Change this to your actual file path

word_counts = count_word_frequency(file_path)

print("Word frequencies in '{}':".format(file_path))
for word, count in word_counts.most_common():
    print("{}: {}".format(word, count))

Word frequencies in 'sample.txt':
'this': 2
'sample': 2
'text': 2
'read': 2
'is': 1
'a': 1
'contains': 1

```

To the right of the code cell is a Gemini AI sidebar. It has a message from Gemini asking for help with generating Python code to read a text file and count word frequency. A response from Gemini provides the code above. Below the sidebar is a "Accept and run" button.

The code defines a function to read a text file, sample.txt, for processing. It converts the entire text to lowercase and extracts all alphanumeric words using regular expressions. The collections.Counter module is then used to efficiently count the frequency of each extracted word. The function is called with the sample.txt file, which was pre-populated with example text. Finally, the code iterates through the word counts and prints each word along with its frequency.

TASK-2

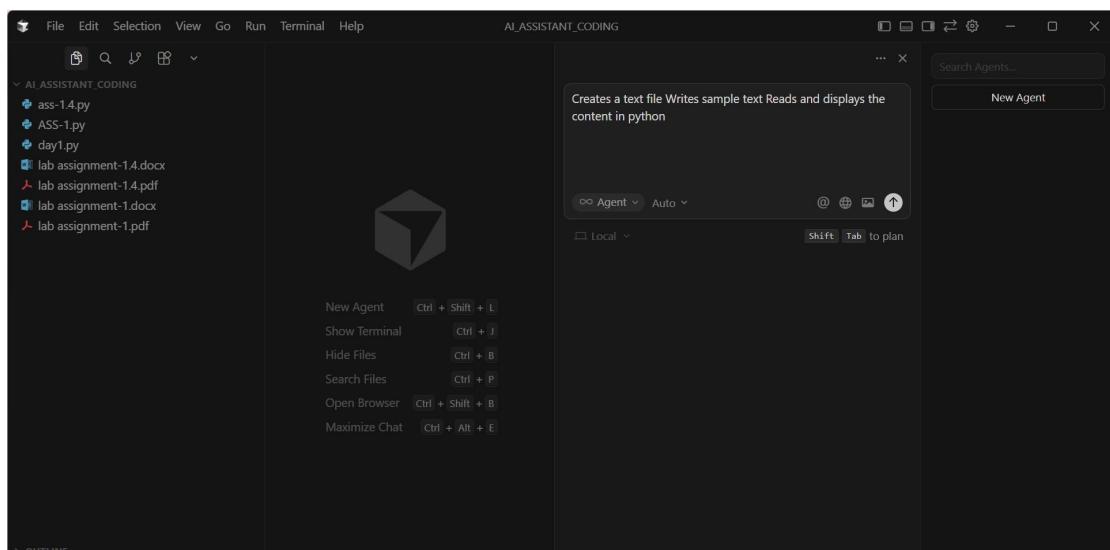
❖ Scenario:

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



```
File Edit Selection View Go Run Terminal Help
AI_ASSISTANT_CODING
23.py x
AI_ASSISTANT_CODING
23.py
ass-14.py
ASS-1.py
day1.py
file_operations.py
lab assignment-1A.docx
lab assignment-1A.pdf
lab assignment-1.docx
lab assignment-1.pdf
sample.txt

3 # Step 1: Create and write sample text to a file
4 filename = "sample.txt"
5 sample_text = """Hello, World!
6 This is a sample text file.
7 Python file operations are simple and powerful.
8 We can write, read, and manipulate files easily."""
9
10 # Write the text to the file
11 with open(filename, 'w') as file:
12     file.write(sample_text)
13
14 print(f"✓ Created and wrote sample text to '{filename}'")
15 print()
16
17 # Step 2: Read and display the content
18 with open(filename, 'r') as file:
19     content = file.read()
20
21 print(f"Content of '{filename}':")
22 print("-" * 50)
23 print(content)
24 print("-" * 50)

Problems Output Debug Console Terminal Ports
Content of 'sample.txt':
-----
Hello, World!
This is a sample text file.
Python file operations are simple and powerful.
We can write, read, and manipulate files easily.

PS D:\AI_ASSISTANT_CODING> Cursor Tab ln 17, Col 39 Spaces: 4 UTF-8 ENG IN 10:25 AM 1/21/2026
```

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

o=x8lEKrop2ZcU

Gemini

◆ Certainly! Here's a 5-line explanation of the code:

1. The code defines a function to read a text file, `sample.txt`, for processing.
2. It converts the entire text to lowercase and extracts all alphanumeric words using regular expressions.
3. The `collections.Counter` module is then used to efficiently count the frequency of each extracted word.
4. The function is called with the `sample.txt` file, which was pre-populated with example text.
5. Finally, the code iterates through the word counts and prints each word along with its frequency.

Show me the Python code to count word frequency

Run the word frequency counting code

How can I sort the word counts by frequency

read a CSV file and calculate mean, min, and max.

+ Gemini 2.5 Flash ▶

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

✓ 10:11 Python 3

10:34 AM 1/21/2026

The screenshot shows a Jupyter Notebook interface with a sidebar titled "Files" containing files like config, sample_data, machine-readable-business-em..., sample.txt, and sample_data.csv. The main area has a code cell starting with "import pandas as pd". It creates a DataFrame "df_sample" from a dictionary "data" and saves it to "sample_data.csv". The output shows the DataFrame content:

	Category	Value	Quantity
0	A	18	3
1	B	25	5
2	A	15	2
3	C	38	8
4	B	20	3
5	A	12	2
6	C	35	7

Below, another cell reads the CSV back into a DataFrame and prints statistics for the "Value" column.

The screenshot shows a Jupyter Notebook interface with a sidebar titled "Files" containing files like config, sample_data, machine-readable-business-em..., sample.txt, and sample_data.csv. The main area has a code cell starting with "import pandas as pd". It reads "sample_data.csv" into a DataFrame "df", prints the first few rows, and then calculates mean, min, and max for the "Value" column. The output shows the DataFrame content and the calculated statistics:

	Category	Value	Quantity
0	A	18	3
1	B	25	5
2	A	15	2
3	C	38	8
4	B	20	3
5	A	12	2
6	C	35	7

Statistics for "Value":

- Mean: 23.00
- Min: 12.00
- Max: 38.00

The screenshot shows a Google Colab notebook titled "Untitled19.ipynb". The code cell contains Python code to read a CSV file, calculate mean, min, and max values, and print the first few rows of the data. The output shows the data frame and its statistics.

```

print(f"Mean: {mean_value:.2f}")
print(f"Min: {min_value:.2f}")
print(f"Max: {max_value:.2f}")
else:
    print(f"\nError: Column '{numerical_column}' not found or is not numeric in the CSV file.")
    print(f"Available columns: {df.columns.tolist()}")
except FileNotFoundError:
    print(f"\nError: The file '{csv_file_path}' was not found. Please check the path.")
except Exception as e:
    print(f"\nAn error occurred: {e}")

```

... Successfully loaded 'sample_data.csv'. Here's the first few rows:

	Category	Value	Quantity
0	A	10	1
1	B	25	5
2	A	15	2
3	C	30	8
4	B	20	3

Statistics for column 'Value':
 Mean: 21.00
 Min: 10.00
 Max: 35.00

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

The screenshot shows a web browser window with the following details:

- Tab Bar:** "New Tab" (closed), "NZ CSV files" (closed), "+" (new tab), "-" (minimize), square icon (maximize/minimize), and "X" (close).
- Toolbar:** Includes icons for search, star, copy, download, profile (labeled "N"), and more.
- Share Panel:** Buttons for "Share", "Comment", and "Settings".
- Performance Metrics:** RAM and Disk usage indicators.
- Content Area:** A sidebar on the left has a trash bin icon and three dots. The main area is titled "Gemini".
- Text Interaction:** A purple circle with "N" says "read a CSV file and calculate mean, min, and max." A blue diamond icon replies: "Okay, I can help with that! Here's the Python code to read a CSV file, then calculate and display the mean, minimum, and maximum values for a numerical column. I'll include a step to create a sample CSV first." Below this is a "gemini" logo.
- Action Buttons:** "Accept and run", "Accept", and "Cancel".
- Input Field:** "give a python code for bubble sort" with a plus sign icon.
- Output Area:** "Gemini 2.5 Flash" dropdown, "Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)".
- System Status:** "10:35", "Python 3", "ENIG", and "10:20 AM".

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains a Python function for bubble sort and some test cases. The output pane shows the execution results, including the original and sorted lists for both examples.

```

def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j] # Swap elements

    return arr

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", my_list)

sorted_list = bubble_sort(my_list)
print("Sorted list:", sorted_list)

another_list = [5, 1, 4, 2, 8]
print("Original list:", another_list)

sorted_another_list = bubble_sort(another_list)
print("Sorted list:", sorted_another_list)

```

Output:

```

Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list: [11, 12, 22, 25, 34, 64, 90]
Original list: [5, 1, 4, 2, 8]
Sorted list: [1, 2, 4, 5, 8]

```

N use python built in function for the same question sort()

◆ You're right! Python has very efficient built-in sorting mechanisms. Here's how you can sort lists using `list.sort()` (which sorts the list in-place) and `sorted()` (which returns a new sorted list).

The screenshot shows a Jupyter Notebook interface in Google Colab. It compares two methods for sorting lists: `list.sort()` (in-place) and `sorted()` (returns a new list). The code demonstrates both methods on the same list and prints the results.

```

# Using list.sort() (sorts in-place)
list_to_sort_in_place = [64, 34, 25, 12, 22, 11, 90]
print("Original list (for in-place sort):", list_to_sort_in_place)
list_to_sort_in_place.sort()
print("Sorted list (in-place):", list_to_sort_in_place)

print("\n---")

# Using sorted() (returns a new sorted list)
original_list_for_sorted = [5, 1, 4, 2, 8]
print("Original list (for sorted() function):", original_list_for_sorted)
new_sorted_list = sorted(original_list_for_sorted)
print("New sorted list (from sorted() function):", new_sorted_list)
print("Original list remains unchanged:", original_list_for_sorted)

```

Output:

```

Original list (for in-place sort): [64, 34, 25, 12, 22, 11, 90]
Sorted list (in-place): [11, 12, 22, 25, 34, 64, 90]

---
Original list (for sorted() function): [5, 1, 4, 2, 8]
New sorted list (from sorted() function): [1, 2, 4, 5, 8]
Original list remains unchanged: [5, 1, 4, 2, 8]

```

Short comparison

Bubble Sort: It's a simple sorting algorithm, but generally inefficient for large datasets ($O(n^2)$ complexity). It sorts by repeatedly stepping through the list, comparing adjacent elements and swapping them.

Python's `list.sort()` and `sorted()`: These are highly optimized and implemented in C, using Timsort (a hybrid stable sorting algorithm with $O(n \log n)$ average and worst-case complexity). They are significantly faster and more memory-efficient for most use cases. `list.sort()` sorts in-place, while `sorted()` returns a new sorted list.