

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:

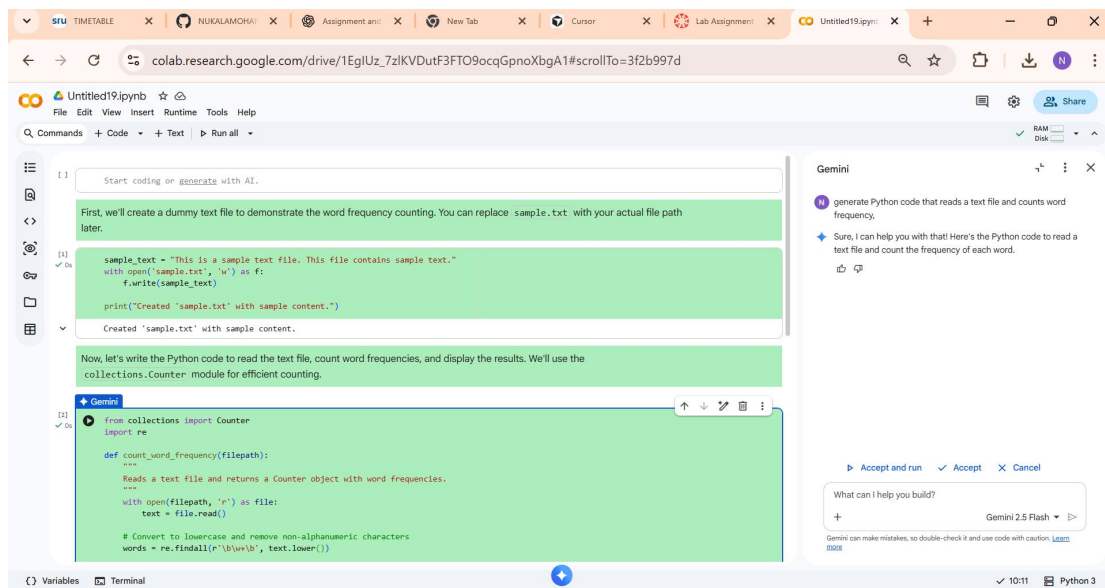
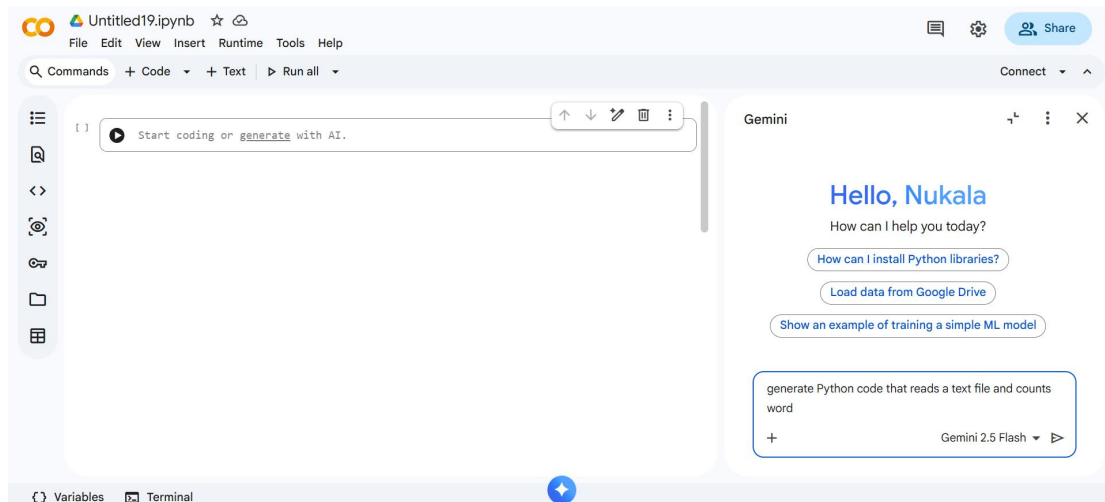
Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.

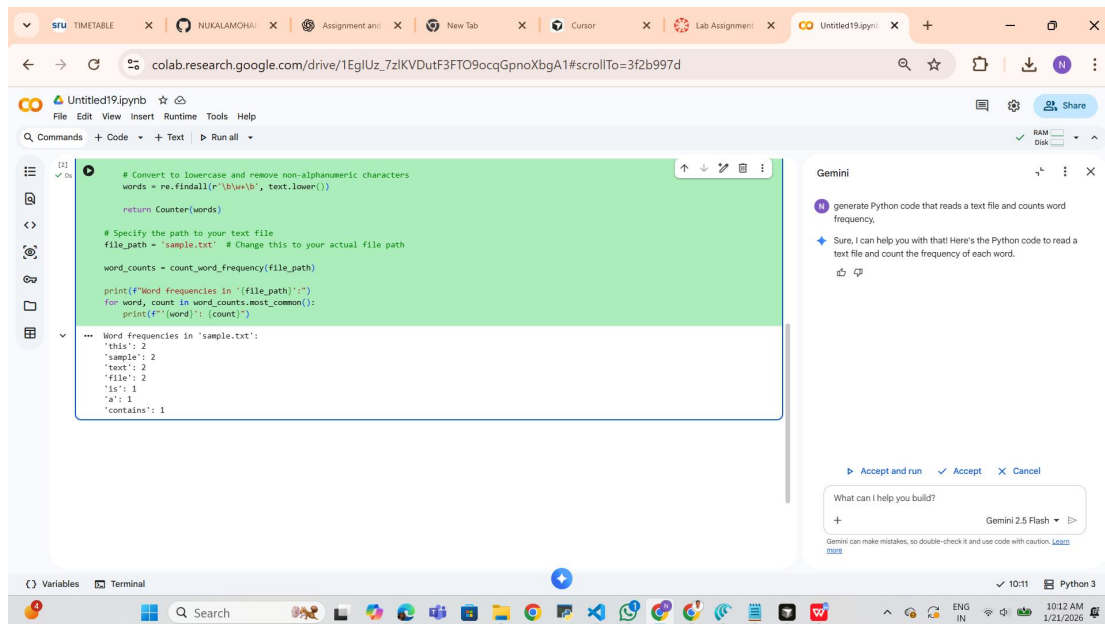
❖ Expected Output:

➤ Working code

➤ Explanation

➤ Screenshot





The 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.

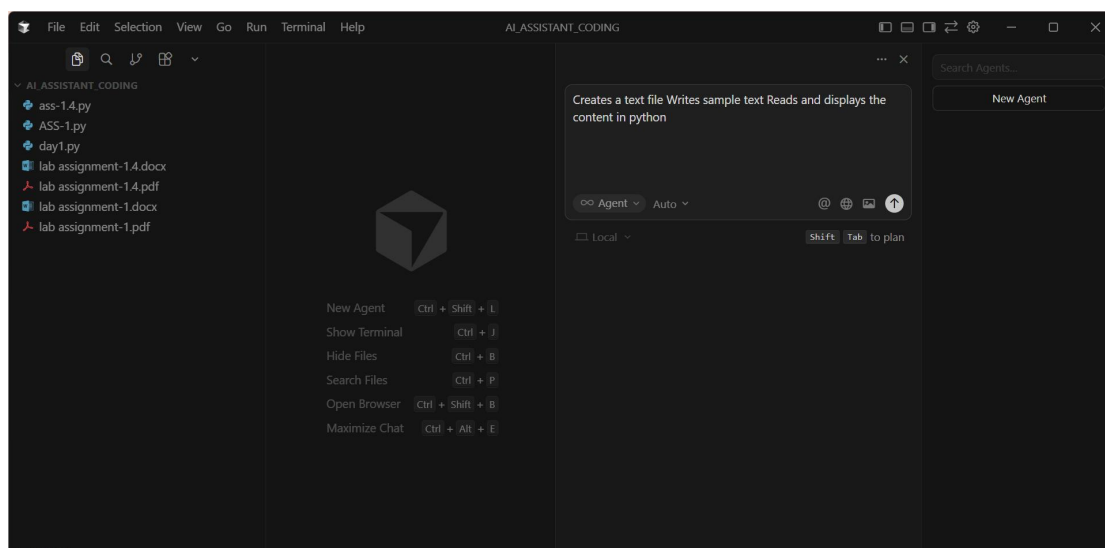
❖ Task:

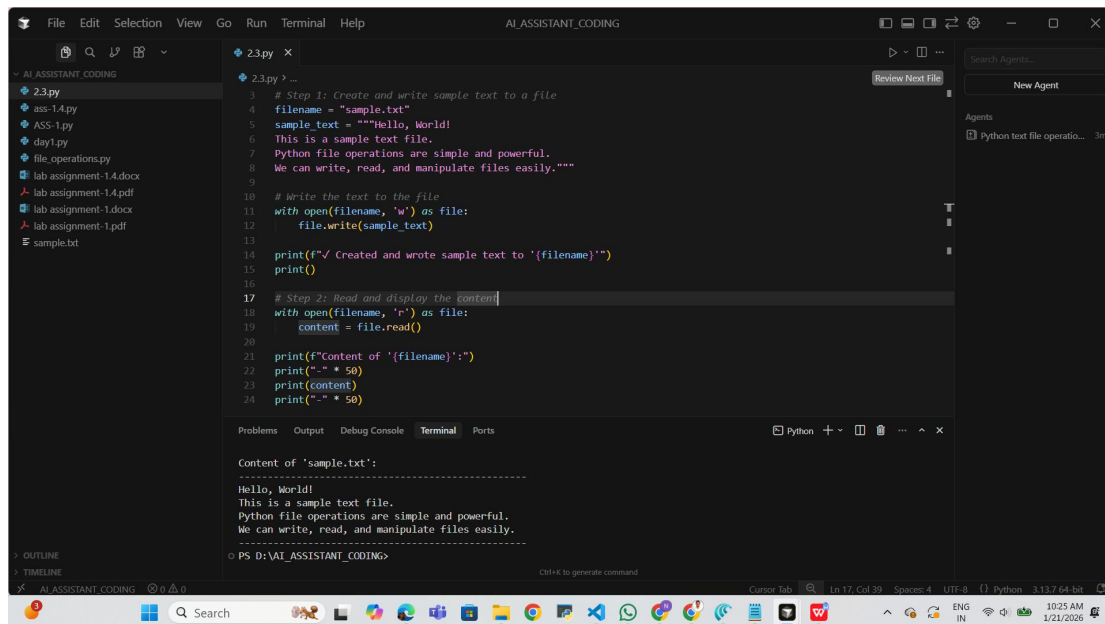
Use Cursor AI to generate a program that:

- Creates a text file
- Writes sample text
- Reads and displays the content

❖ Expected Output:

- Functional code
- Cursor AI screenshots





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

Share

RAM
Disk

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

ENG
IN

10:34 AM
1/21/2026

colab.research.google.com/drive/1EglUz_7zIKVDutF3FTO9ocqGpnoXbgA1#scrollTo=fcc15417

Untitled19.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

Files

- ..
- config
- sample_data
- machine-readable-business-em...
- sample.txt
- sample_data.csv

Disk 86.58 GB available

Start coding or generate with AI.

First, let's create a sample CSV file named `sample_data.csv` that we can use for demonstration. It will contain some numerical data.

```
[1] import pandas as pd

# Create a sample DataFrame
data = {'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C'],
        'Value': [10, 25, 15, 30, 20, 12, 35],
        'Quantity': [1, 5, 2, 8, 3, 2, 7]}
df_sample = pd.DataFrame(data)

# Save the DataFrame to a CSV file
df_sample.to_csv('sample_data.csv', index=False)

print(f"Created 'sample_data.csv' with sample content.")
print(df_sample)
```

	Category	Value	Quantity
0	A	10	1
1	B	25	5
2	A	15	2
3	C	30	8
4	B	20	3
5	A	12	2
6	C	35	7

Now, let's read the `sample_data.csv` file into a pandas DataFrame and calculate the mean, min, and max for a numerical column, for example, the 'Value' column.

Gemini

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

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.

Accept and run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

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

Variables Terminal

Search

10:35 AM 1/21/2026

colab.research.google.com/drive/1EglUz_7zIKVDutF3FTO9ocqGpnoXbgA1#scrollTo=fcc15417

Untitled19.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

Files

- ..
- config
- sample_data
- machine-readable-business-em...
- sample.txt
- sample_data.csv

Disk 86.58 GB available

Now, let's read the `sample_data.csv` file into a pandas DataFrame and calculate the mean, min, and max for a numerical column, for example, the 'Value' column.

```
[1] import pandas as pd

# Specify the path to your CSV file
csv_file_path = 'sample_data.csv' # Change this to your actual CSV file path

try:
    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(csv_file_path)

    print(f"Successfully loaded '{csv_file_path}'. Here's the first few rows:")
    display(df.head())

    # Select a numerical column to calculate statistics (e.g., 'Value')
    # You can change 'Value' to any numerical column in your CSV.
    numerical_column = 'Value'

    if numerical_column in df.columns and pd.api.types.is_numeric_dtype(df[numerical_column]):
        # Calculate mean, min, and max
        mean_value = df[numerical_column].mean()
        min_value = df[numerical_column].min()
        max_value = df[numerical_column].max()

        print(f"\nStatistics for column '{numerical_column}':")
        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("Available columns: ", df.columns.tolist())
```

Gemini

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

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.

Accept and run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

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

Variables Terminal

Search

10:35 AM 1/21/2026

The screenshot shows a Google Colab notebook with the following content:

```

import pandas as pd

def read_csv_file(csv_file_path):
    try:
        df = pd.read_csv(csv_file_path)
        print(f"Mean: {df['Value'].mean():.2f}")
        print(f"Min: {df['Value'].min():.2f}")
        print(f"Max: {df['Value'].max():.2f}")
    except:
        print(f"\nError: Column '{numerical_column}' not found or is not numeric in the CSV file.")
        print("Available columns: ", df.columns.tolist())

except FileNotFoundError:
    print(f"Error: The file '{csv_file_path}' was not found. Please check the path.")
except Exception as e:
    print(f"An 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

The Gemini chat interface on the right shows a conversation about reading a CSV file and calculating statistics.

Task 4: Sorting Lists – Manual vs Built-in

❖ Scenario:

You are reviewing algorithm choices for efficiency.

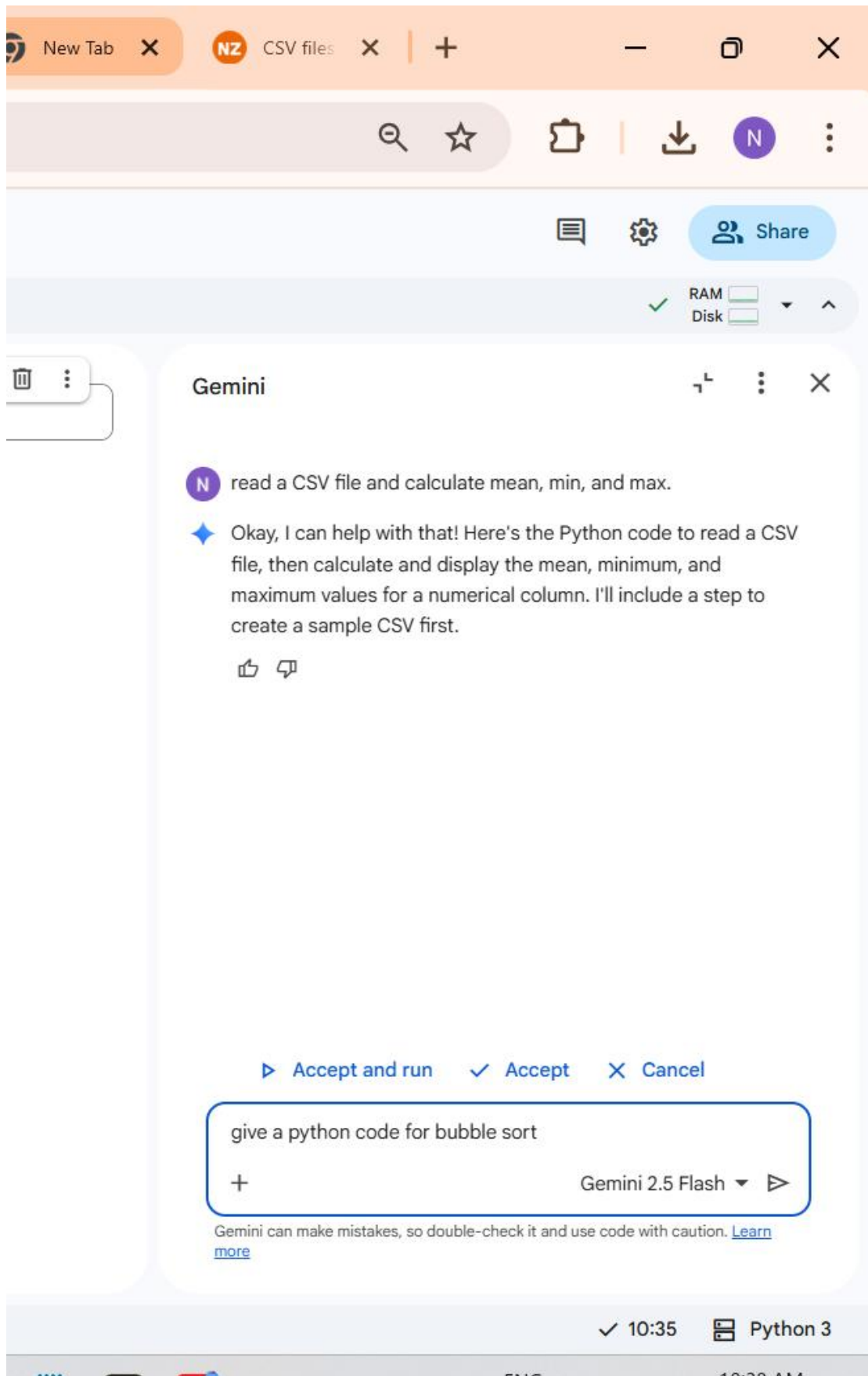
❖ Task:

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 Google Colab notebook titled 'Untitled19.ipynb'. The left sidebar displays a file explorer with folders like '.config', 'sample_data', and files like 'sample.txt' and 'sample_data.csv'. The main area contains a code cell with a custom bubble sort function. The output shows the original and sorted versions of two lists.

```
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]
```

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 Google Colab notebook with a code cell comparing `list.sort()` and `sorted()`. The output demonstrates that `list.sort()` modifies the list in-place, while `sorted()` returns a new sorted list without changing the original.

```
# 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.