AI ASSISTED CODING

LAB Assignment 2.3

A.Srichala

BT-31

CSE-AIML

2303A52435

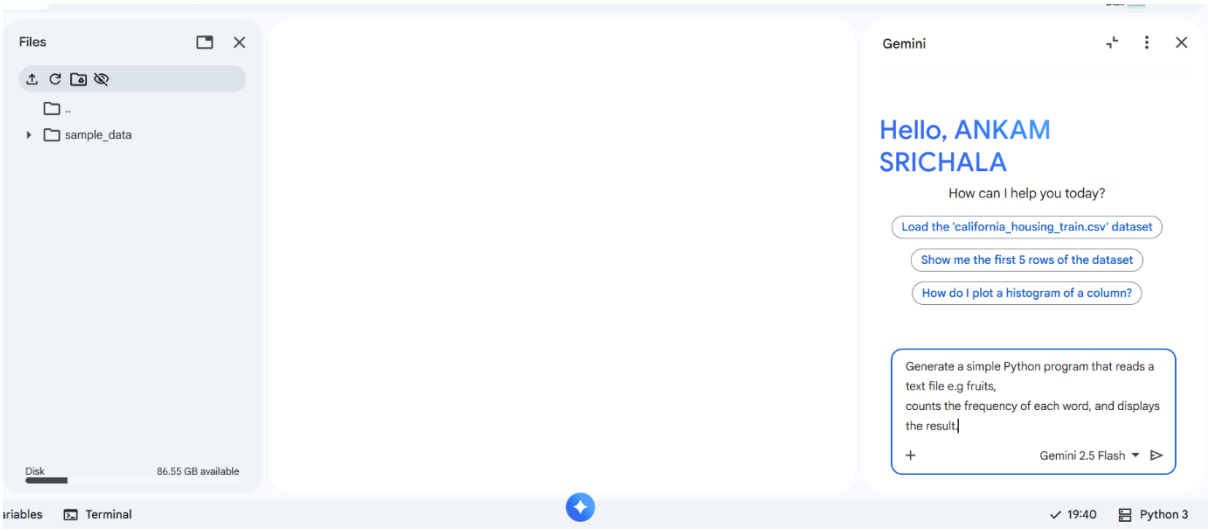
Task 1: Word Frequency from Text File

❖ Scenario:

You are analyzing log files for keyword frequency.

❖ Task:

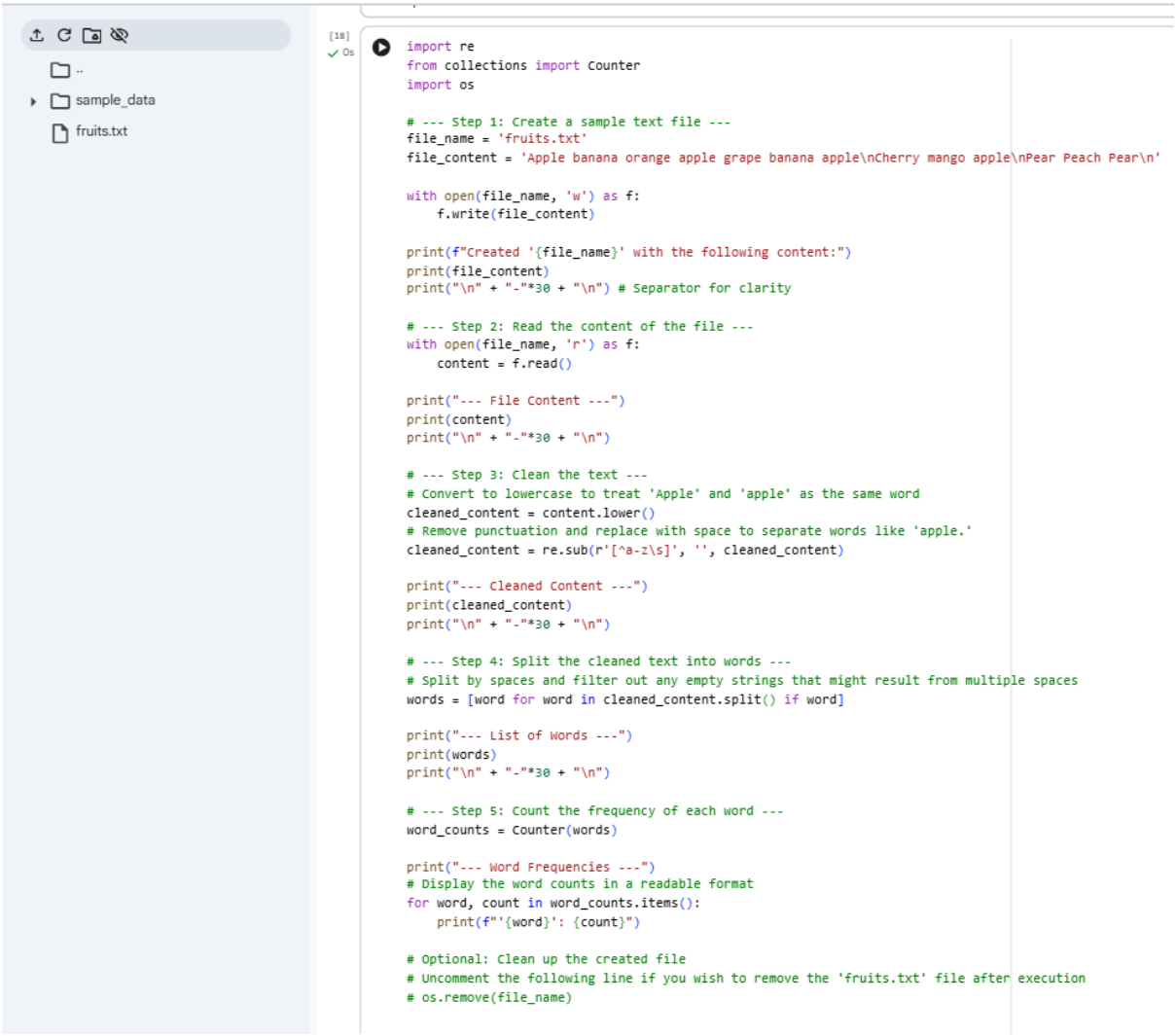
Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.



Prompt:-

Generate a Simple python program that reads a text file e.g, fruits counts the frequency of each word, and displays the result

Code:-


..
sample_data
fruits.txt
[18]
0s
import re
from collections import Counter
import os
# --- Step 1: Create a sample text file ---
file_name = 'fruits.txt'
file_content = 'Apple banana orange apple grape banana apple\nCherry mango apple\nPear Peach Pear\n'
with open(file_name, 'w') as f:
f.write(file_content)
print(f"Created '{file_name}' with the following content:")
print(file_content)
print("\n" + "-"*30 + "\n") # Separator for clarity
--- Step 2: Read the content of the file ---
with open(file_name, 'r') as f:
content = f.read()
print("--- File Content ---")
print(content)
print("\n" + "-"*30 + "\n")
# --- Step 3: Clean the text ---
# Convert to lowercase to treat 'Apple' and 'apple' as the same word
cleaned_content = content.lower()
# Remove punctuation and replace with space to separate words like 'apple.'
cleaned_content = re.sub(r'[^a-z\s]', '', cleaned_content)
print("--- Cleaned Content ---")
print(cleaned_content)
print("\n" + "-"*30 + "\n")
# --- Step 4: Split the cleaned text into words ---
# Split by spaces and filter out any empty strings that might result from multiple spaces
words = [word for word in cleaned_content.split() if word]
print("--- List of Words ---")
print(words)
print("\n" + "-"*30 + "\n")
# --- Step 5: Count the frequency of each word ---
word_counts = Counter(words)
print("--- Word Frequencies ---")
# Display the word counts in a readable format
for word, count in word_counts.items():
print(f"'{word}': {count}")
# Optional: Clean up the created file
# Uncomment the following line if you wish to remove the 'fruits.txt' file after execution
# os.remove(file_name)

Output:-

```
Created 'fruits.txt' with the following content:
Apple banana orange apple grape banana apple
Cherry mango apple
Pear Peach Pear


------------------------------

--- File Content ---
Apple banana orange apple grape banana apple
Cherry mango apple
Pear Peach Pear


------------------------------

--- Cleaned Content ---
apple banana orange apple grape banana apple
cherry mango apple
pear peach pear


------------------------------

--- List of Words ---
['apple', 'banana', 'orange', 'apple', 'grape', 'banana', 'apple', 'cherry', 'mango', 'apple', 'pear', 'peach', 'pear']

------------------------------

--- Word Frequencies ---
'apple': 4
'banana': 2
'orange': 1
'grape': 1
'cherry': 1
'mango': 1
'pear': 2
'peach': 1
```

Explanation:-

We created a sample fruits.txt file with text content.

The program read this file, then converted its text to lowercase and removed punctuation.

It then split the cleaned text into individual words.

Using collections. Counter, it calculated the frequency of each unique word.

Finally, it displayed the count for every word found in the file.

Task 2: File Operations Using Cursor AI

❖ Scenario:

You are automating basic file operations.

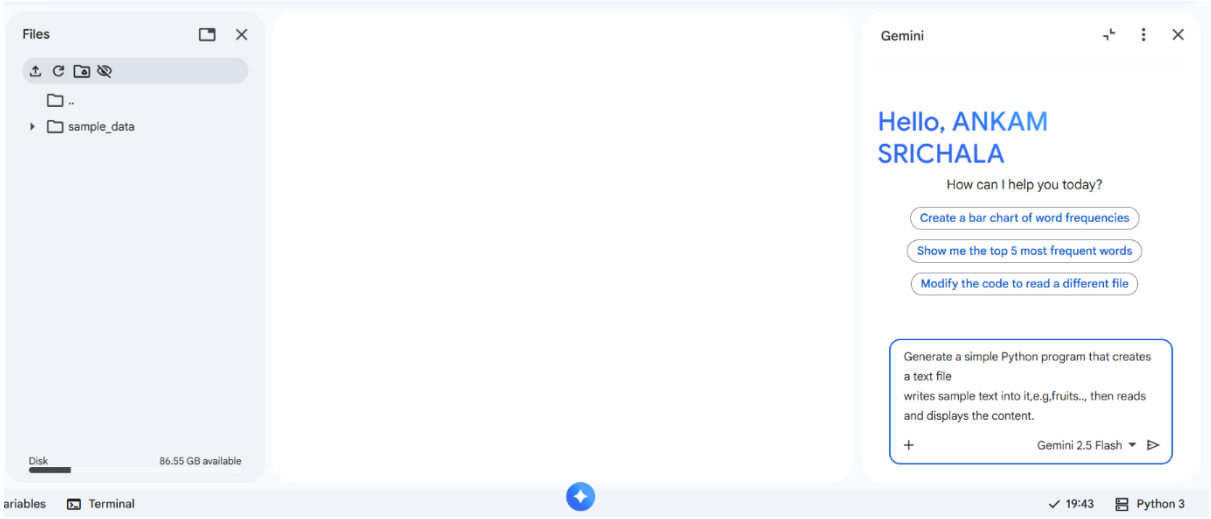
❖ Task:

Use Cursor AI to generate a program that:

➢ Creates a text file

➢ Writes sample text

➢ Reads and displays the content



Prompt:-

Generate a simple python that creates a text file write sample text into it e.g., fruits, then reads and display the content.

Code:-

Output:-

```
Successfully created and wrote to 'fruits.txt'.

Content of 'fruits.txt':

Apple banana orange apple grape banana apple
Cherry mango apple
Pear Peach Pear
```

Explanation:- The program first defines the file name (fruits.txt) and the sample content.
It then opens the file in write mode ('w').
The sample content is written into the fruits.txt file.
Next, the file is opened again in read mode ('r').
Finally, the file content is read and printed on the console.

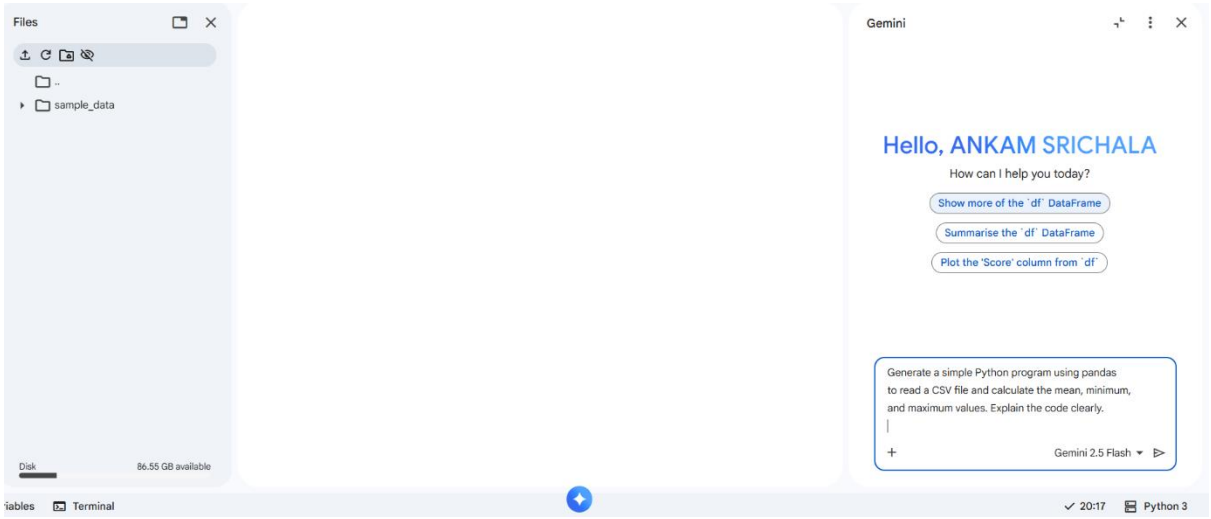
Task 3: CSV Data Analysis

❖ 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.



Prompt:-

Generate a simple python program using pandas to read a CSV file and calculate the mean, minimum and maximum values. Explain the code clearly.

Code:-

```
import pandas as pd
import io

# Define the simple numerical data as a string
simple_numbers_data = """10
20
30
40
50"""

# Read the string data into a pandas DataFrame
df_simple = pd.read_csv(io.StringIO(simple_numbers_data), header=None, names=['Value'])

print("Simple Numerical DataFrame:")
display(df_simple)

# Calculate the mean, minimum, and maximum of the 'Value' column
mean_simple = df_simple['Value'].mean()
min_simple = df_simple['Value'].min()
max_simple = df_simple['Value'].max()

print(f"\nMean of the numbers: {mean_simple}")
print(f"Minimum of the numbers: {min_simple}")
print(f"Maximum of the numbers: {max_simple}")
```

Output:-

```
Simple Numerical DataFrame:
```

	Value
0	10
1	20
2	30
3	40
4	50

```
Mean of the numbers: 30.0
Minimum of the numbers: 10
Maximum of the numbers: 50
```

Explanation:-

1. The program imports the pandas library.
2. It reads the CSV file using read_csv().
3. The mean, minimum, and maximum values are calculated.
4. These calculations are done on the Values column.
5. The results are printed to the console.

Task 4: Sorting Lists – Manual vs Built-in

❖ Scenario:

You are reviewing algorithm choices for efficiency.

❖ Task:

Use Gemini to generate:

➢ Bubble sort

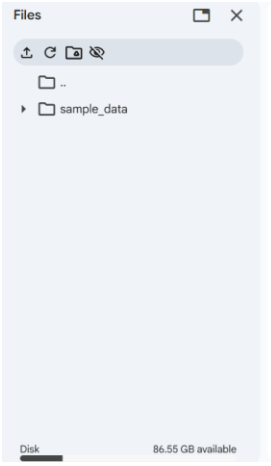
➢ Python's built-in sort()

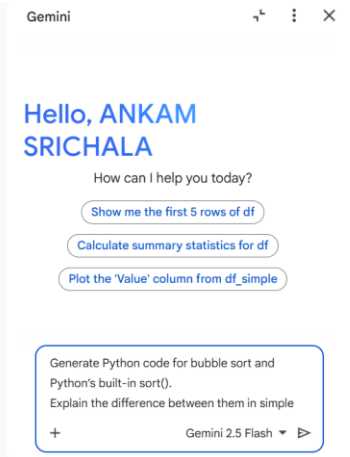
➢ Compare both implementations.

Prompt:-

Generate Python code for bubble sort and Python's built-in sort().

Explain the difference between them in simple terms.





Code:-

Manual sort-

```
my_list = [64, 34, 25, 12, 22, 11, 90]

def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j] # Swap elements

bubble_sort(my_list)
sorted_list_bubble = my_list
print("Sorted list using Bubble Sort:", sorted_list_bubble)
```

Output:-

```
Sorted list using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
```

Built-in Sort:-

```
my_list_2 = [64, 34, 25, 12, 22, 11, 90]
my_list_2.sort() # Sorts the list in-place
sorted_list_builtin_func = my_list_2
print("Sorted list using Python's built-in sort():", sorted_list_builtin_func)
```

Output:-

```
Sorted list using Python's built-in sort(): [11, 12, 22, 25, 34, 64, 90]
```

Explanation :-Bubble Sort is a basic sorting algorithm that compares and swaps adjacent elements.
It is easy to understand and implement.
However, it is very slow for large datasets with time complexity $O(n^2)$.
Python's built-in sort() uses an optimized algorithm called Timsort.
It is much faster and efficient with average time complexity O(n log n).