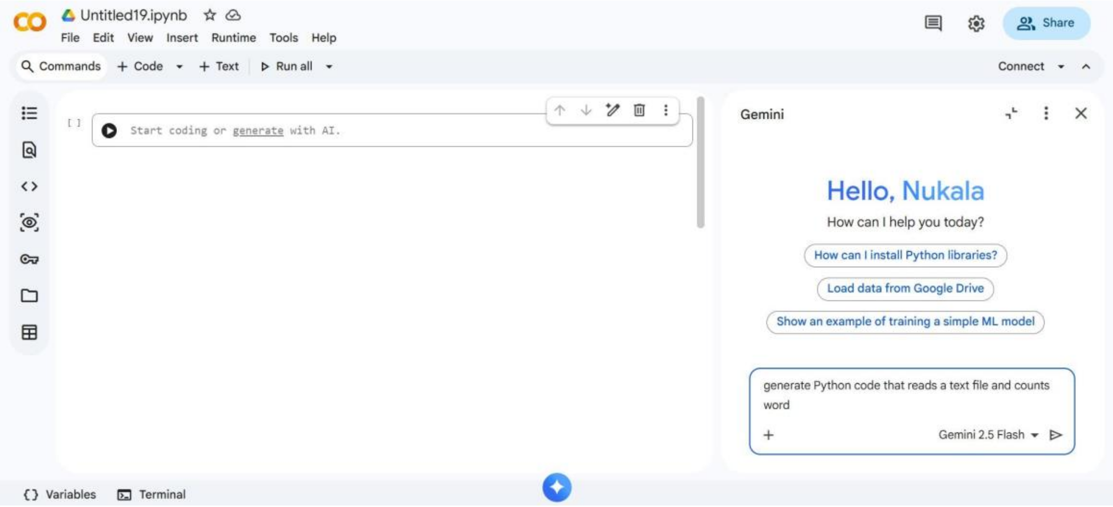
Gangula Navyasri
2303A51739
TASK-1

## Task 1: Word Frequency from Text File

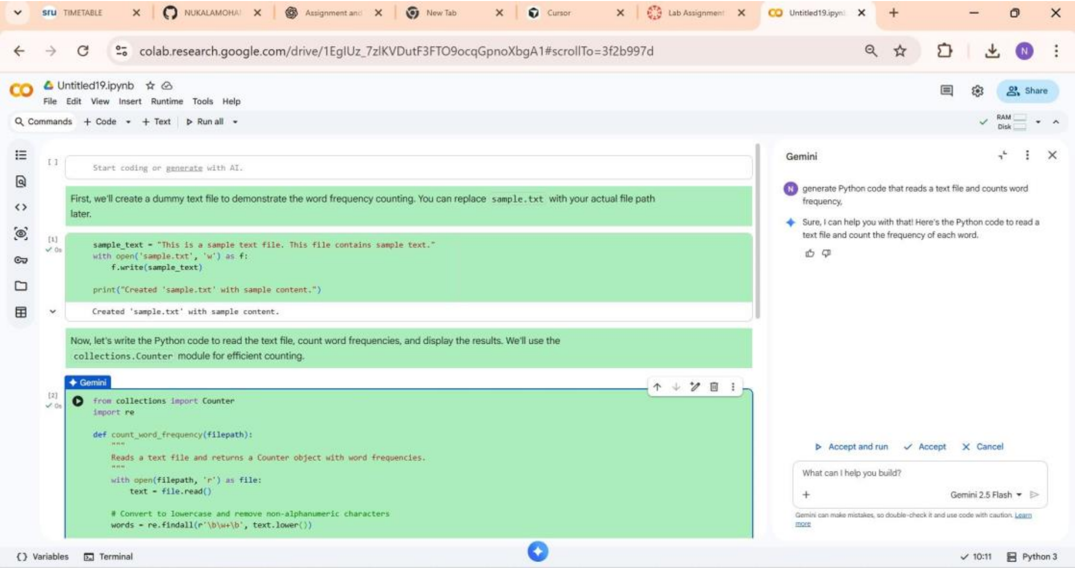
❖ Scenario:
You are analyzing log files for keyword frequency.
❖ Task:
Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.
❖ Expected Output:
➢ Working code
➢ Explanation
➢ Screenshot


Untitled19.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Run all
Share
Connect
Gemini
How can I install Python libraries?
Load data from Google Drive
Show an example of training a simple ML model
Gemini 2.5 Flash
Variables
Terminal

Start coding or generate with AI.

Hello, Nukala
How can I help you today?

generate Python code that reads a text file and counts word



colab.research.google.com/drive/1EglUz_7zlKVDutF3FTO9ocqGpnoXbgA1#scrollTo=3f2b997d

Start coding or generate with AI.

First, we'll create a dummy text file to demonstrate the word frequency counting. You can replace `sample.txt` with your actual file path later.

```
sample_text = "This is a sample text file. This file contains sample text."
with open('sample.txt', 'w') as f:
    f.write(sample_text)

print("Created 'sample.txt' with sample content.")
```

```
Created 'sample.txt' with sample content.
```

Now, let's write the Python code to read the text file, count word frequencies, and display the results. We'll use the `collections.Counter` module for efficient counting.

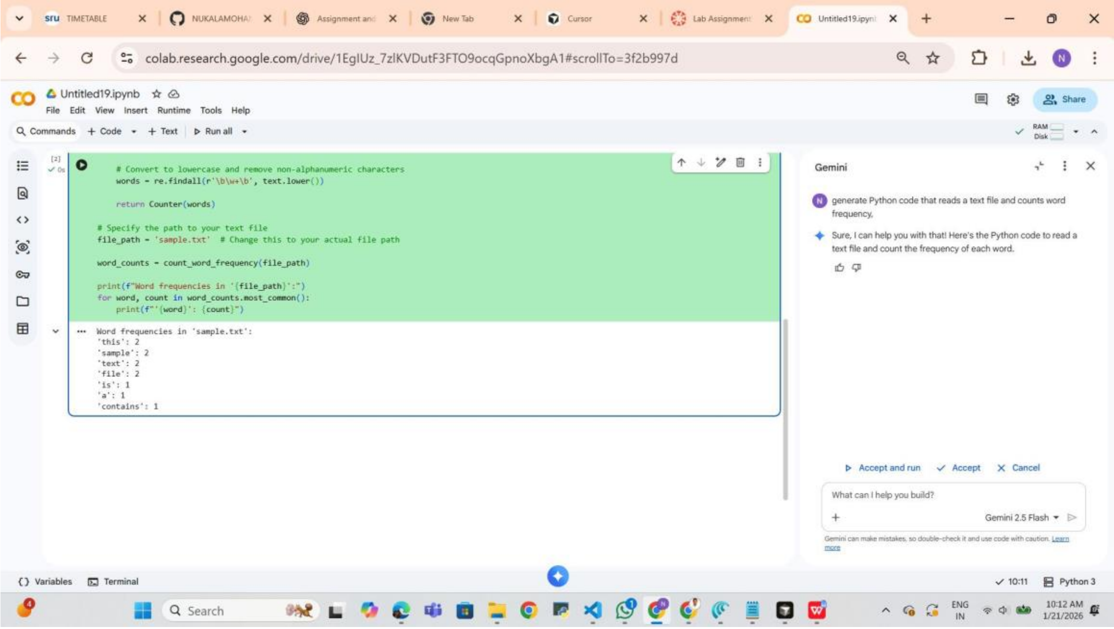
```
from collections import Counter
import re

def count_word_frequency(filepath):
    """
    Reads a text file and returns a Counter object with word frequencies.
    """
    with open(filepath, 'r') as file:
        text = file.read()

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

generate Python code that reads a text file and counts word frequency.

Sure, I can help you with that! Here's the Python code to read a text file and count the frequency of each word.

What can I help you build?

Gemini can make mistakes, so double-check it and use code with caution. Learn more



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

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

generate Python code that reads a text file and counts word frequency.

Sure, I can help you with that! Here's the Python code to read a text file and count the frequency of each word.

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.

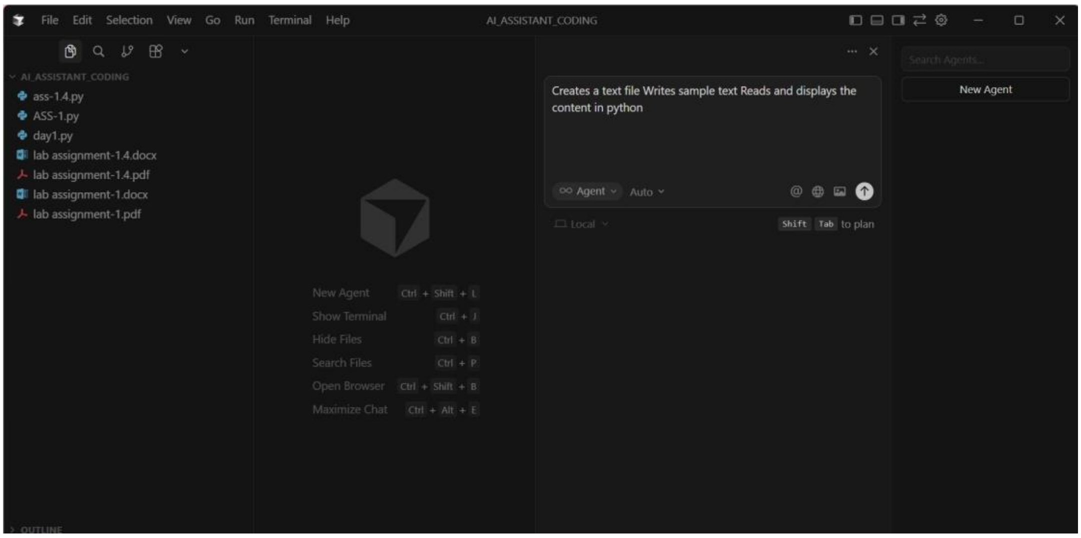
❖ Task:

Use Cursor AI to generate a program that:

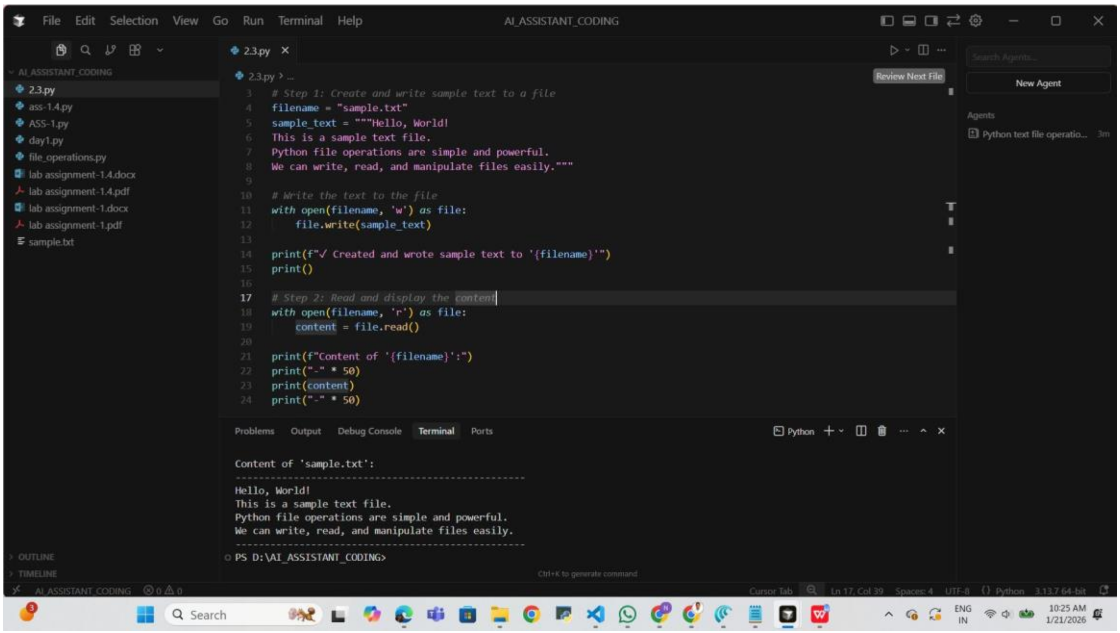
➢ Creates a text file
➢ Writes sample text
➢ Reads and displays the content

❖ Expected Output:

➢ Functional code
➢ Cursor AI screenshots



Creates a text file Writes sample text Reads and displays the content in python



```python
# Step 1: Create and write sample text to a file
filename = "sample.txt"
sample_text = """Hello, World!
This is a sample text file.
Python file operations are simple and powerful.
We can write, read, and manipulate files easily."""

# Write the text to the file
with open(filename, 'w') as file:
    file.write(sample_text)

print(f"✓ Created and wrote sample text to '{filename}'")
print()

# Step 2: Read and display the content
with open(filename, 'r') as file:
    content = file.read()

print(f"Content of '{filename}':")
print("-" * 50)
print(content)
print("-" * 50)
```

```
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>
```

Task 3: CSV Data Analysis

❖ Scenario:

You are processing structured data from a CSV file.

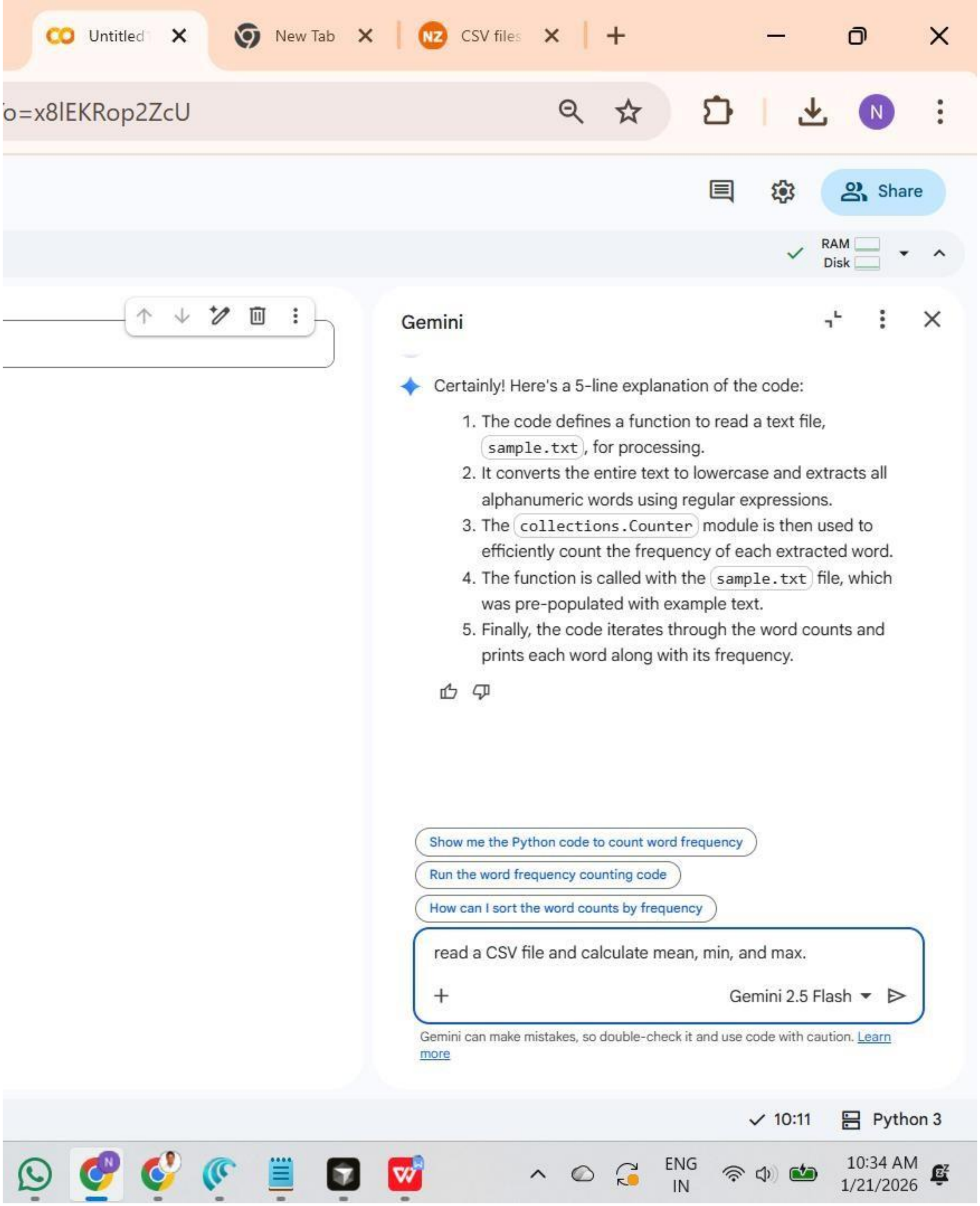
❖ Task:

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

❖ Expected Output:

➢ Correct output

➢ Screenshot



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

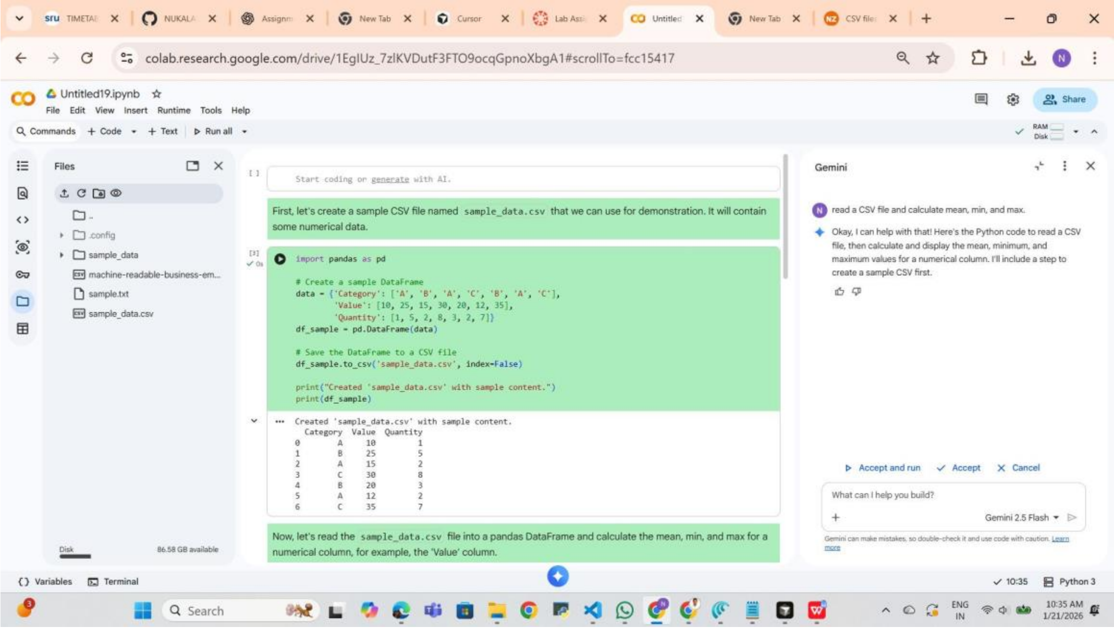
Run the word frequency counting code

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


colab.research.google.com/drive/1EglUz_7zlKVDutF3FTO9ocqGpnoXbgA1#scrollTo=fcc15417
Untitled19.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Files
..
.config
sample_data
machine-readable-business-em...
sample.txt
sample_data.csv
Disk 86.58 GB available
Variables Terminal
10:35 Python 3
Search
ENG IN 10:35 AM 1/21/2026

```
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.

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

```
Created 'sample_data.csv' with sample content.
  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
```

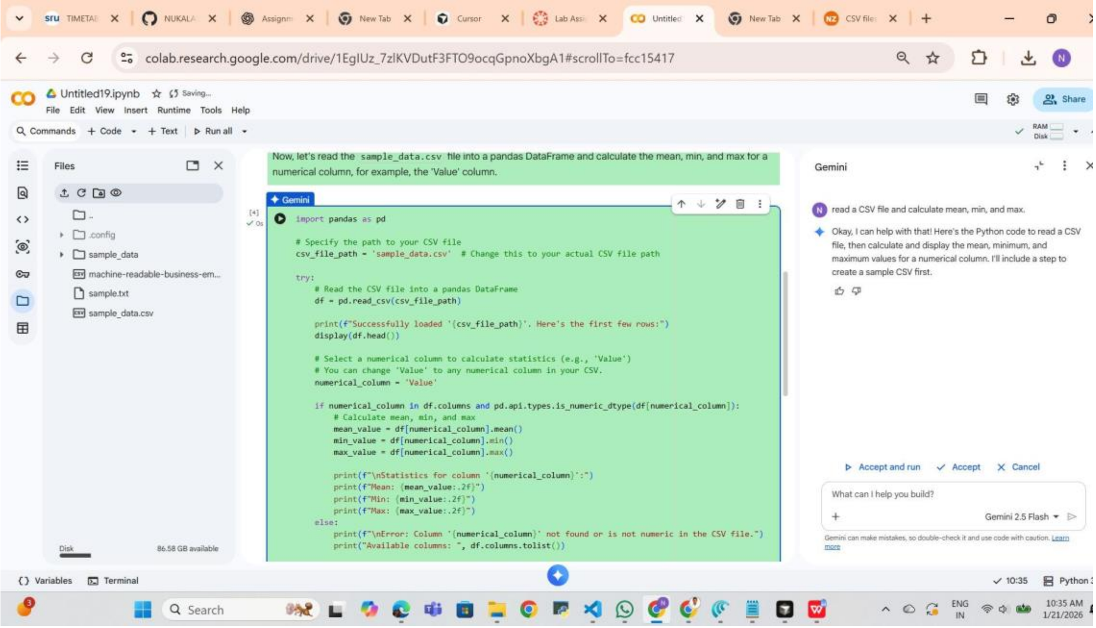
Now, let's read the `sample_data.csv` file into a pandas DataFrame and calculate the mean, min, and max for a numerical column, for example, the 'Value' column.

**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


colab.research.google.com/drive/1EglUz_7zlKVDutF3FTO9ocqGpnoXbgA1#scrollTo=fcc15417
Untitled19.ipynb Saving...
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Files
..
.config
sample_data
machine-readable-business-em...
sample.txt
sample_data.csv
Disk 86.58 GB available
Gemini
Variables Terminal
10:35 Python

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.

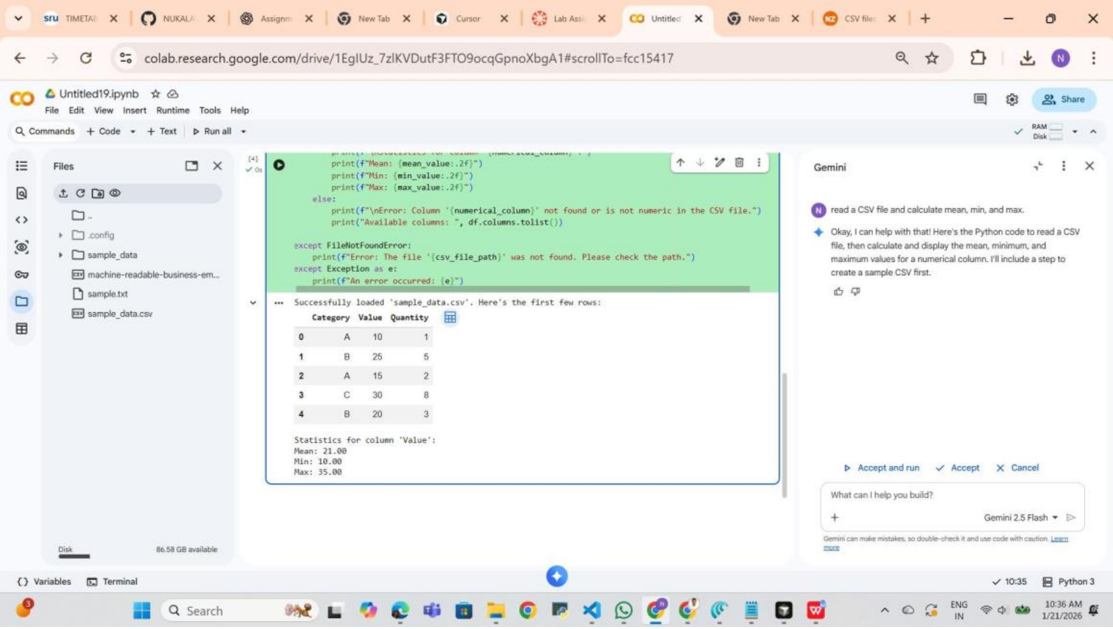
```
import pandas as pd

# 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



```
        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())

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
```

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.

Task 4: Sorting Lists – Manual vs Built-in

❖ Scenario:

You are reviewing algorithm choices for efficiency.

❖ Task:

Use Gemini to generate:

➢ Bubble sort

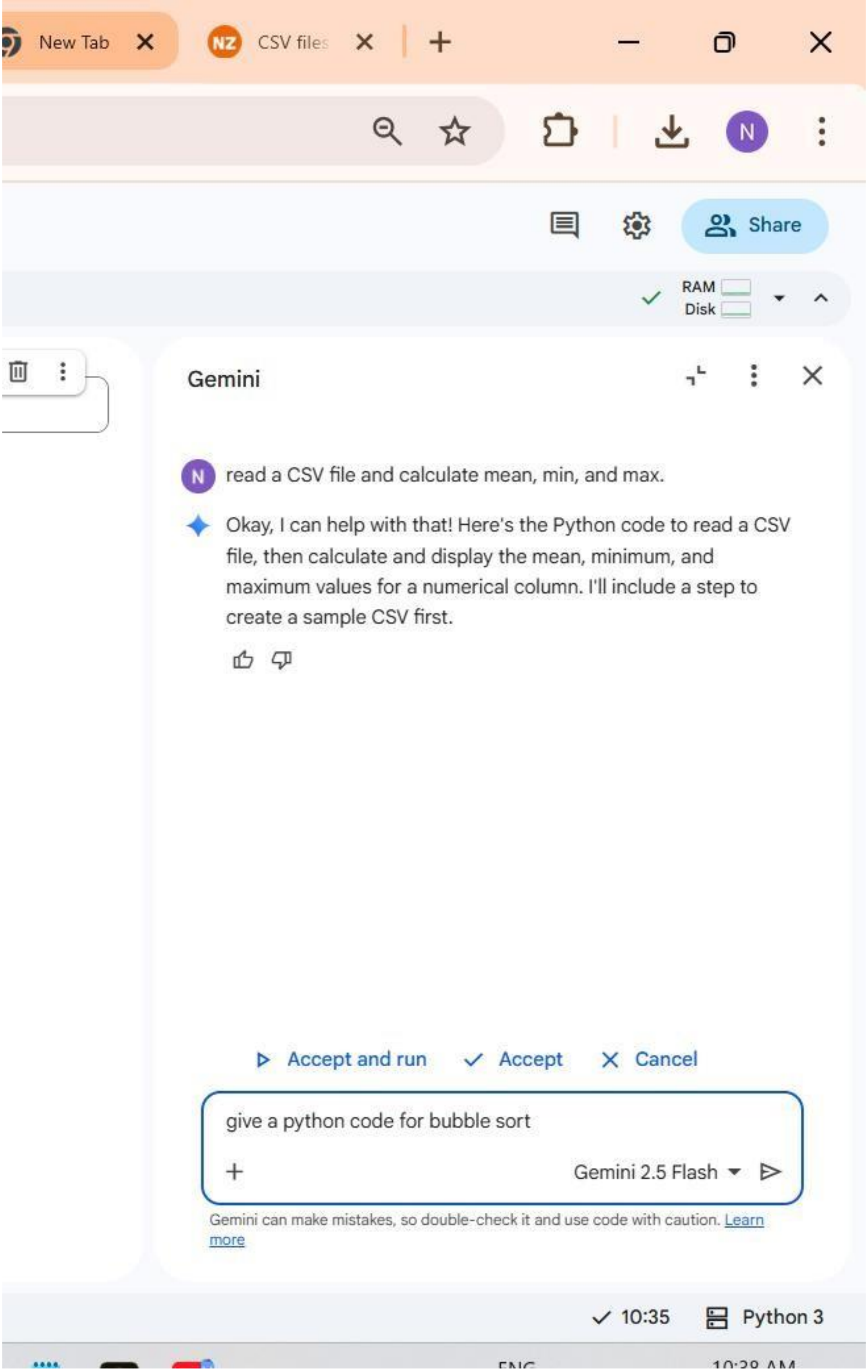
➢ Python's built-in sort()

➢ Compare both implementations.

❖ Expected Output:

➢ Two versions of code

➢ Short comparison

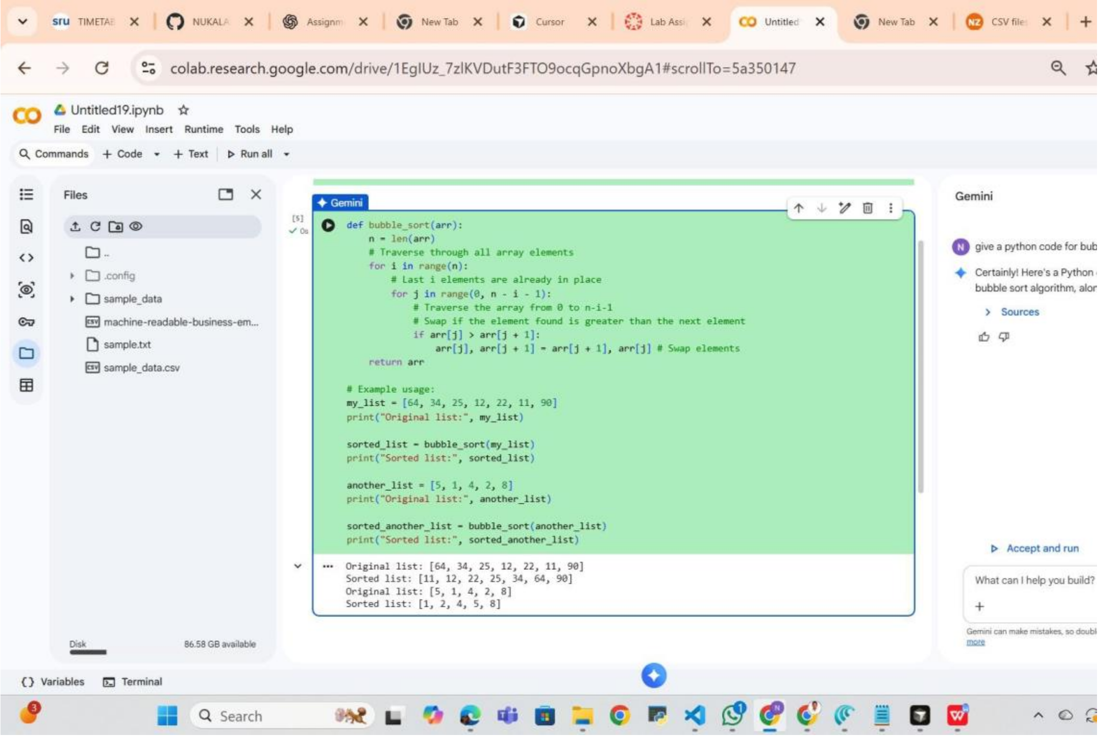


read a CSV file and calculate mean, min, and max.

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.

give a python code for bubble sort

Gemini can make mistakes, so double-check it and use code with caution. Learn more


colab.research.google.com/drive/1EglUz_7zlKVDutF3FTO9ocqGpnoXbgA1#scrollTo=5a350147
Untitled19.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Files
sample_data
machine-readable-business-em...
sample.txt
sample_data.csv
Disk 86.58 GB available
Gemini
Variables Terminal

```
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)
```

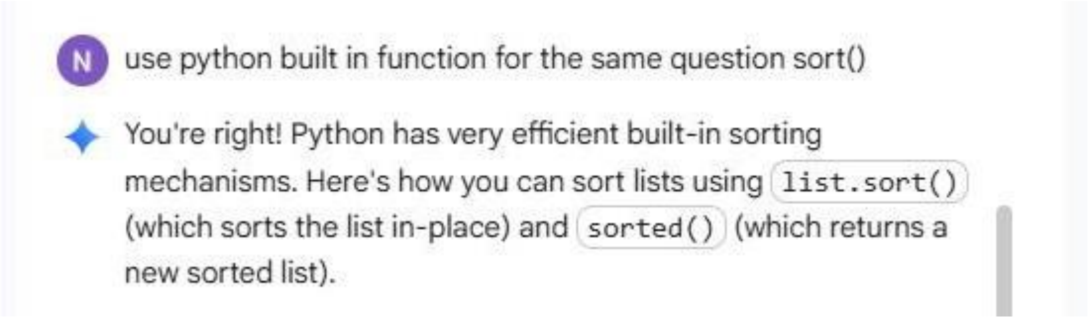
```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list: [11, 12, 22, 25, 34, 64, 90]
Original list: [5, 1, 4, 2, 8]
Sorted list: [1, 2, 4, 5, 8]
```

Gemini

give a python code for bub

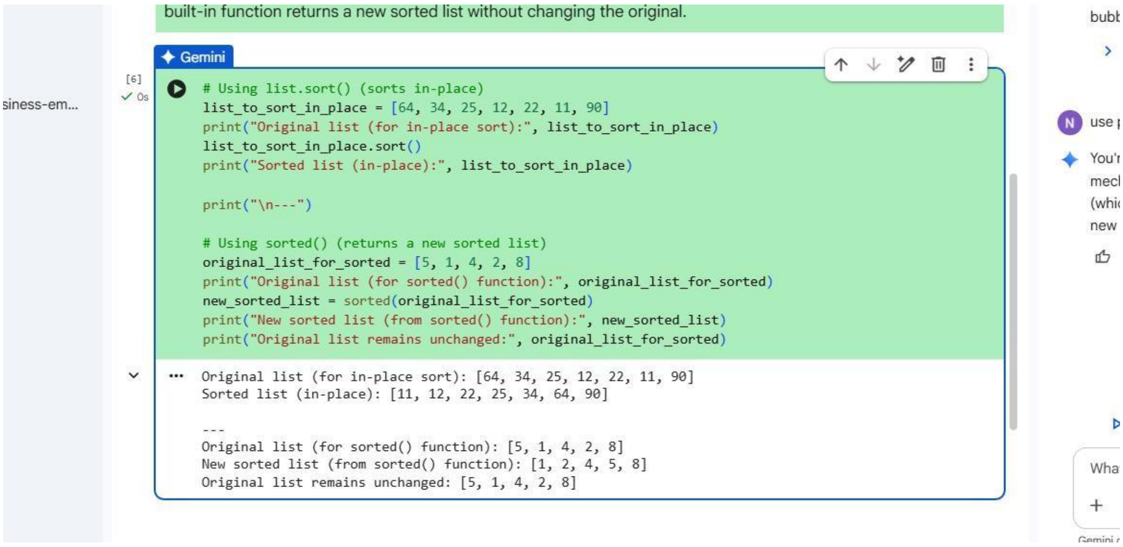
Certainly! Here's a Python
bubble sort algorithm, alor

Sources
Accept and run
What can I help you build?
Gemini can make mistakes, so doubl
more



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).



built-in function 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)
```

```
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 comparision

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.