AI Assisted Coding

ASSIGNMENT 2.3

Name: B. Siri Chandana

HT_No: 2303A52004

Batch: 31

Question:

Task 1:

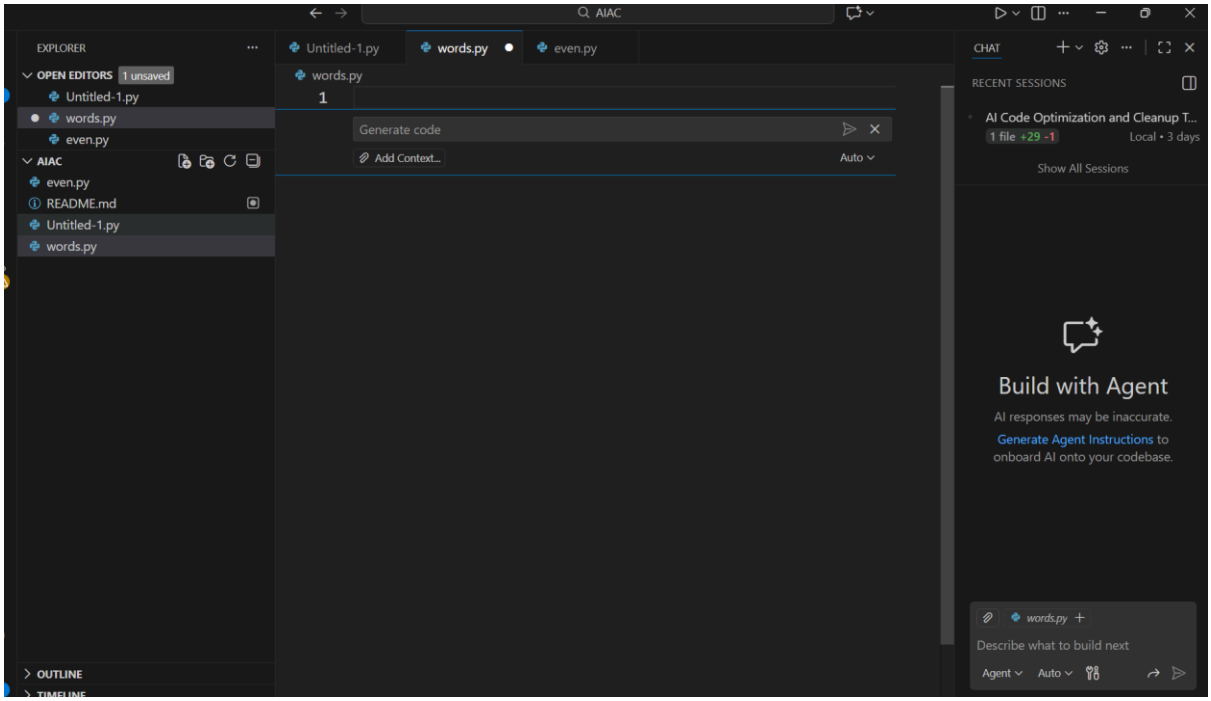
Word Frequency from Text File

Scenario: You are analyzing log files for keyword frequency.

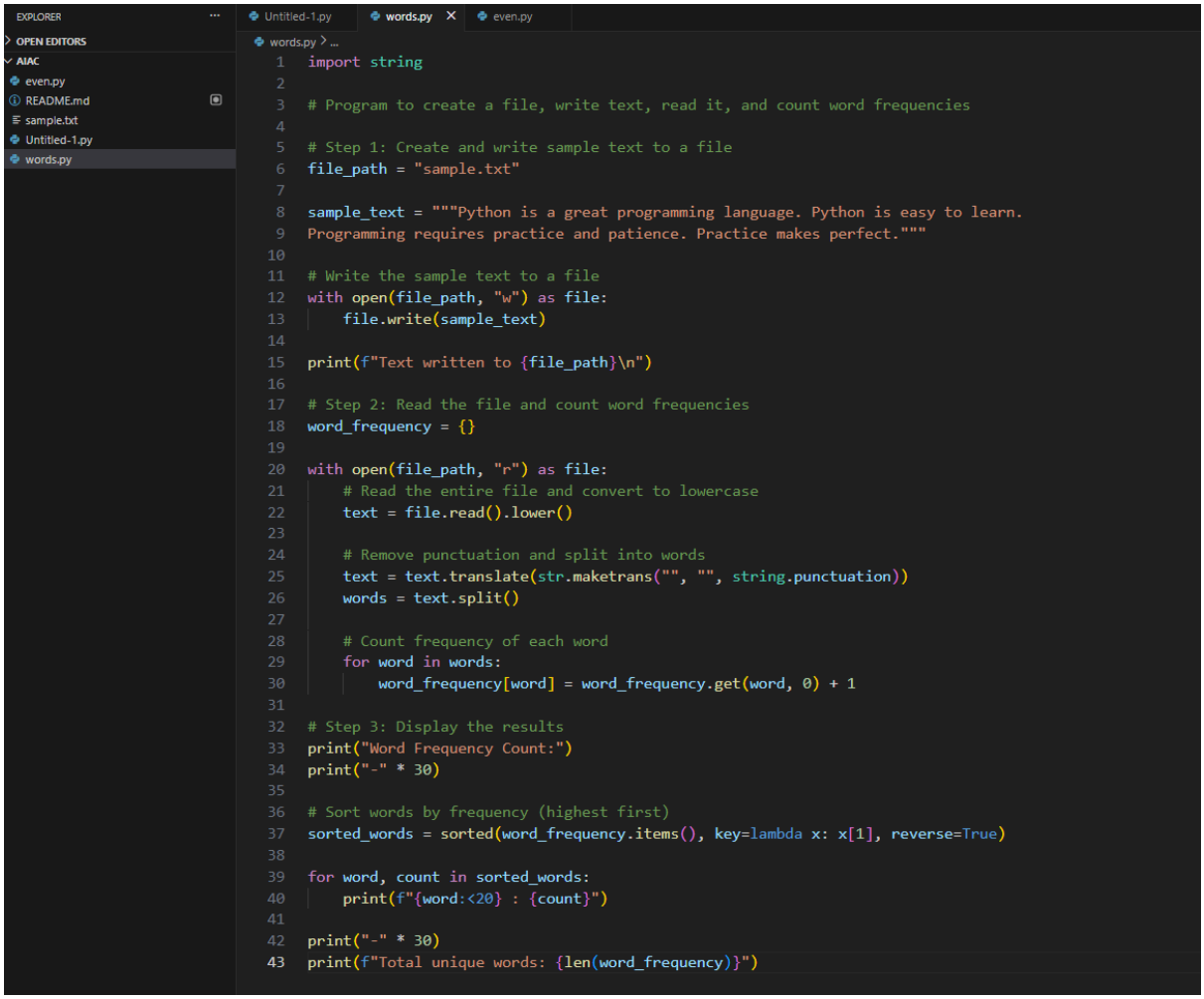
Task: Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code

Prompt:

Create a Python program that creates a text file, writes sample text into it, then reads the file and count the frequency of each word and displays it with its count. Add clear comments in the code.



Code:



Output:

```
PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\words.py"
Text written to sample.txt

Word Frequency Count:
------------------------------
python               : 2
is                   : 2
programming          : 2
practice             : 2
a                    : 1
great                : 1
language             : 1
easy                 : 1
to                   : 1
learn                : 1
requires             : 1
and                  : 1
patience             : 1
makes                : 1
perfect              : 1
------------------------------
Total unique words: 15
PS C:\Users\bogas\OneDrive\Desktop\AIAC>
```

Explanation:

This program first creates a text file and writes sample content into it.
It then reads the file, removes punctuation, and separates the text into individual words.
A dictionary is used to keep track of how many times each word occurs.
Finally, the program displays each word along with its frequency to identify commonly used words.

Question:

Task2:

File Operations Using Cursor AI

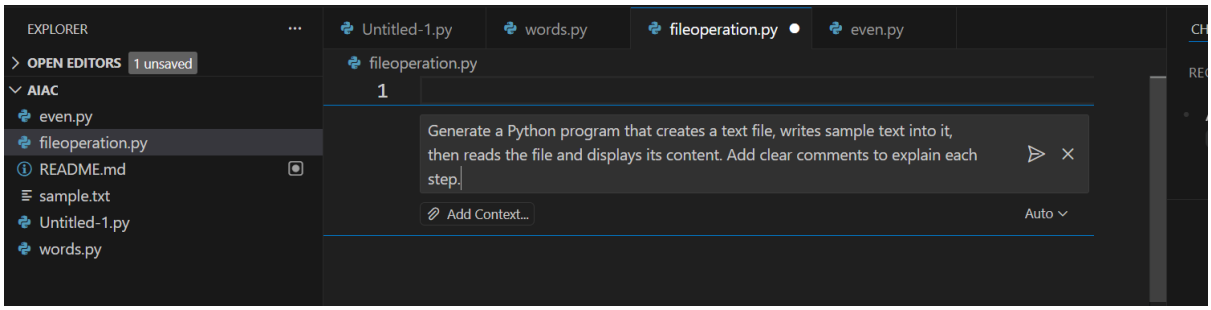
Scenario: You are automating basic file operations.

Task: Use Cursor AI to generate a program that:

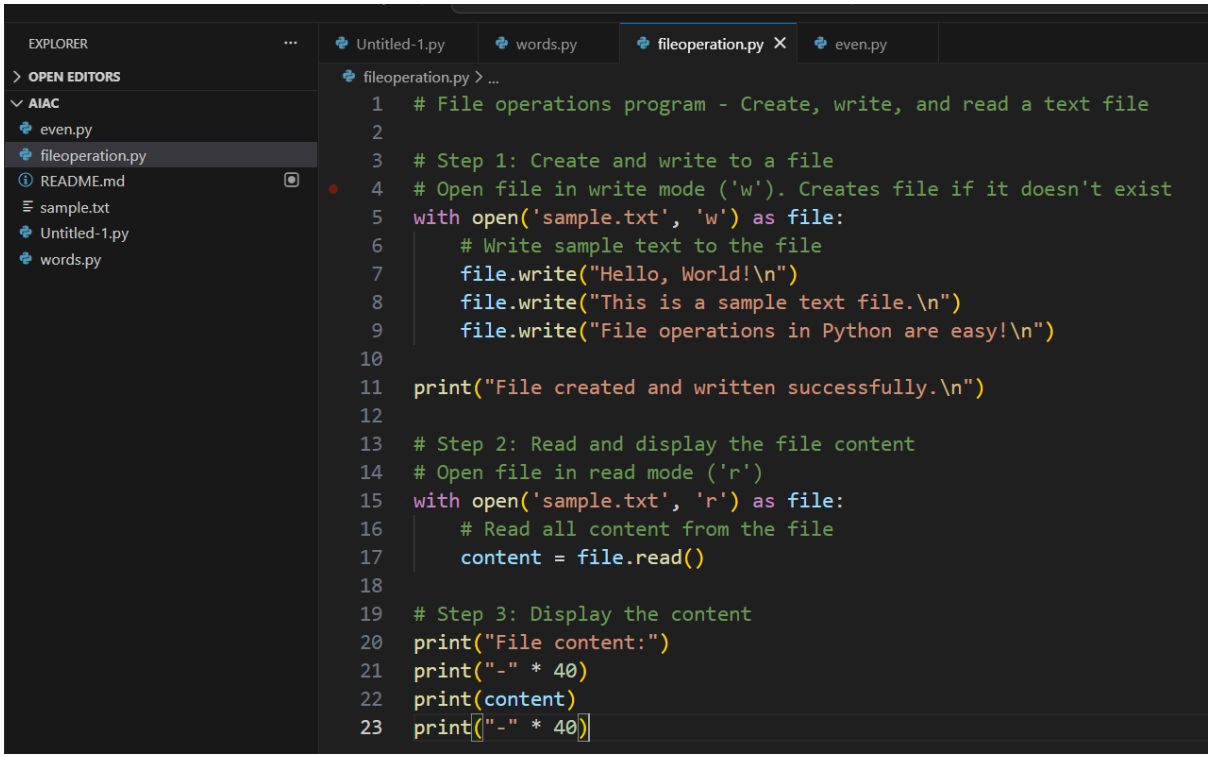
Creates a text file Writes sample text Reads and displays the content

Prompt:

Generate a Python program that creates a text file, writes sample text into it, then reads the file and displays its content. Add clear comments to explain each step.



Code:



Output:

```
PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\fileoperation.py"
File created and written successfully.

File content:
----------------------------------------
Hello, World!
This is a sample text file.
File operations in Python are easy!

----------------------------------------
PS C:\Users\bogas\OneDrive\Desktop\AIAC>
```

Explanation:

This program demonstrates basic file handling in Python.
It creates a text file, writes sample text into it, and then reads and displays the content.
Such file operations are useful for automation and data storage tasks.

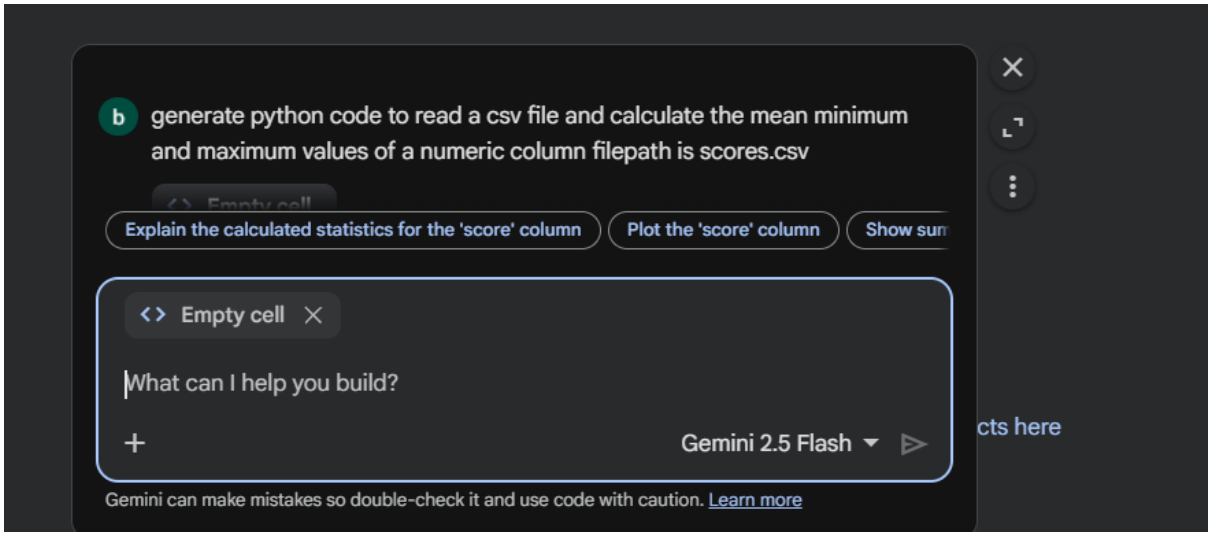
Question:

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:



Code:

```python
import pandas as pd
# --- Configuration --- #
file_path = '/content/scores.csv.csv'  # <--- IMPORTANT: This is now correctly set for your file.
column_name = 'score'  # <--- IMPORTANT: This is now correctly set for your column.
# --- Read CSV and Calculate Statistics --- #
try:
    df = pd.read_csv(file_path)
    print(f"Successfully loaded '{file_path}'.")
    if column_name in df.columns:
        numeric_column = df[column_name]
        # Ensure the column contains numeric data, coercing errors to NaN
        numeric_column = pd.to_numeric(numeric_column, errors='coerce')
        # Drop NaN values that resulted from non-numeric entries
        numeric_column = numeric_column.dropna()
        if not numeric_column.empty:
            mean_value = numeric_column.mean()
            min_value = numeric_column.min()
            max_value = numeric_column.max()

            print(f"\nStatistics for column '{column_name}':")
            print(f"Mean: {mean_value:.2f}")
            print(f"Minimum: {min_value:.2f}")
            print(f"Maximum: {max_value:.2f}")
        else:
            print(f"Error: Column '{column_name}' is either empty or contains no valid numeric data after cleaning.")

    else:
        print(f"Error: Column '{column_name}' not found in the CSV file.")

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

Output:

```
Successfully loaded '/content/scores.csv.csv'.

Statistics for column 'score':
Mean: 82.00
Minimum: 67.00
Maximum: 92.00
```

Explanation:

This Python program reads data from a CSV file and retrieves numerical values from a column. It then calculates the mean, minimum, and maximum. CSV data analysis is commonly used in data processing and analytics applications.

Question:

Task 4: Sorting Lists Manual vs Built-in

Scenario: You are reviewing algorithm choices for efficiency.

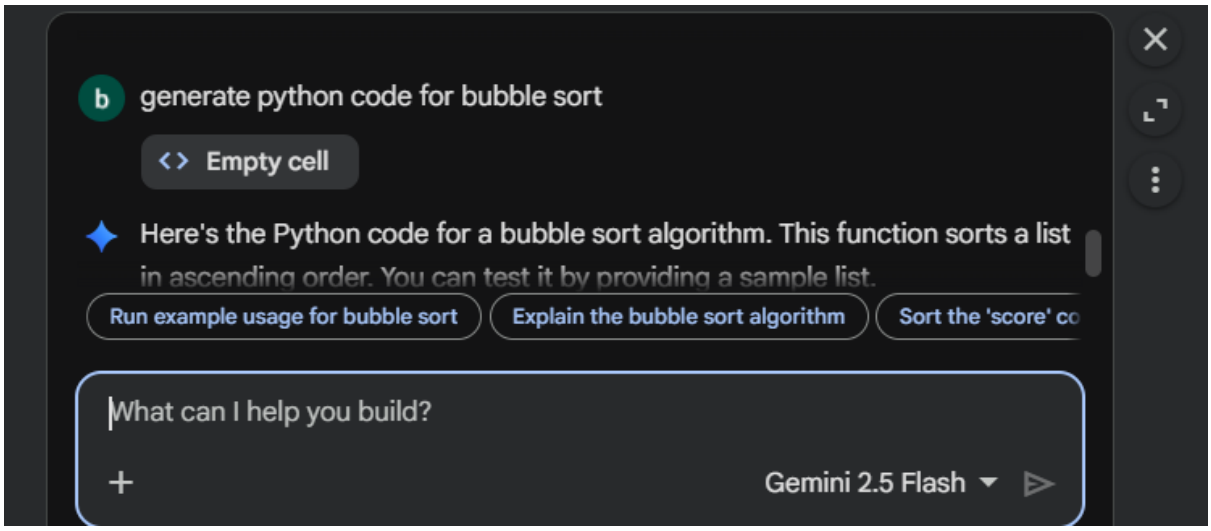
Task: Use Gemini to generate:

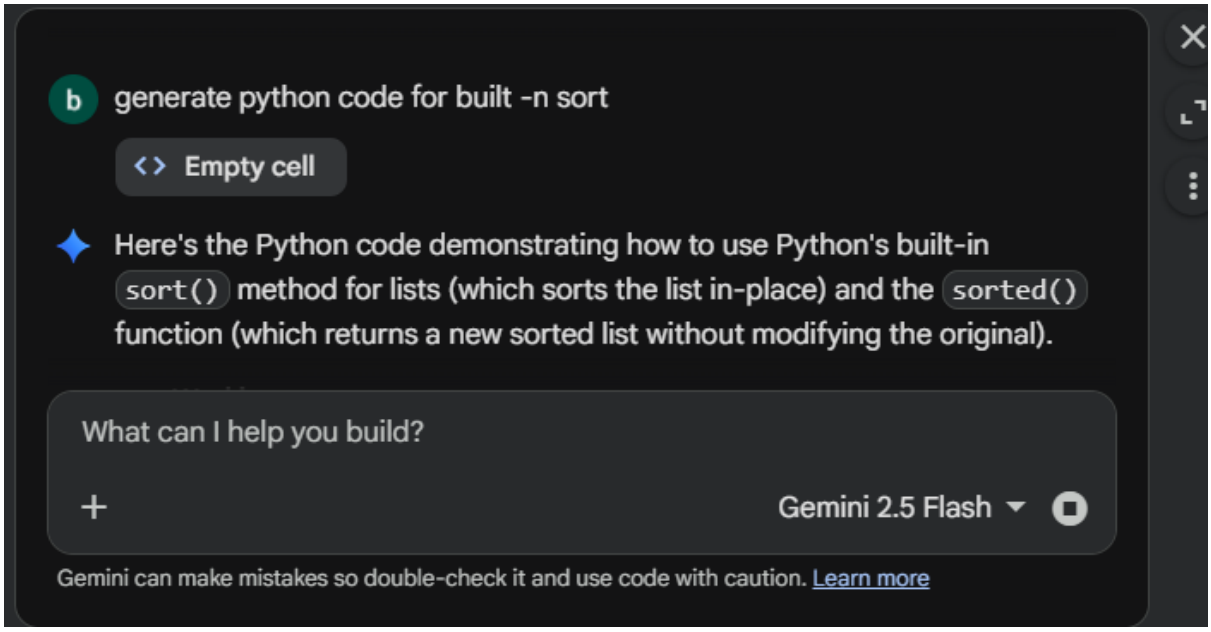
Bubble sort

Python's built-in sort()

Compare both implementations.

Prompt:





Code:

Manual Bubble Sort:

```
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]
    return arr

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list: {my_list}")
sorted_list = bubble_sort(my_list)
print(f"Sorted list: {sorted_list}")
```

Sorting using sort() function:

```python
# Example using the list.sort() method (sorts in-place)
my_list_sort = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list (sort method): {my_list_sort}")
my_list_sort.sort()
print(f"Sorted list (sort method): {my_list_sort}\n")

# Example using the sorted() function (returns a new sorted list)
my_list_sorted_func = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list (sorted function): {my_list_sorted_func}")
sorted_list_new = sorted(my_list_sorted_func)
print(f"New sorted list (sorted function): {sorted_list_new}")
print(f"Original list remains unchanged (sorted function): {my_list_sorted_func}")
```

Output:

Bubble Sort:

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list: [11, 12, 22, 25, 34, 64, 90]
```

Sort Function:

```
Original list (sort method): [64, 34, 25, 12, 22, 11, 90]
Sorted list (sort method): [11, 12, 22, 25, 34, 64, 90]

Original list (sorted function): [64, 34, 25, 12, 22, 11, 90]
New sorted list (sorted function): [11, 12, 22, 25, 34, 64, 90]
Original list remains unchanged (sorted function): [64, 34, 25, 12, 22, 11, 90]
```

Explanation:

Bubble sort is a simple way to sort numbers by comparing and swapping neighbors until everything is in order. It is easy to learn but slow for large lists. Python's built-in sort is much faster and better to use in real applications.