

AI ASSISTED CODING

ASSIGNMENT-2.3

Name: A.Deepthi

Ht.No: 2303A52390

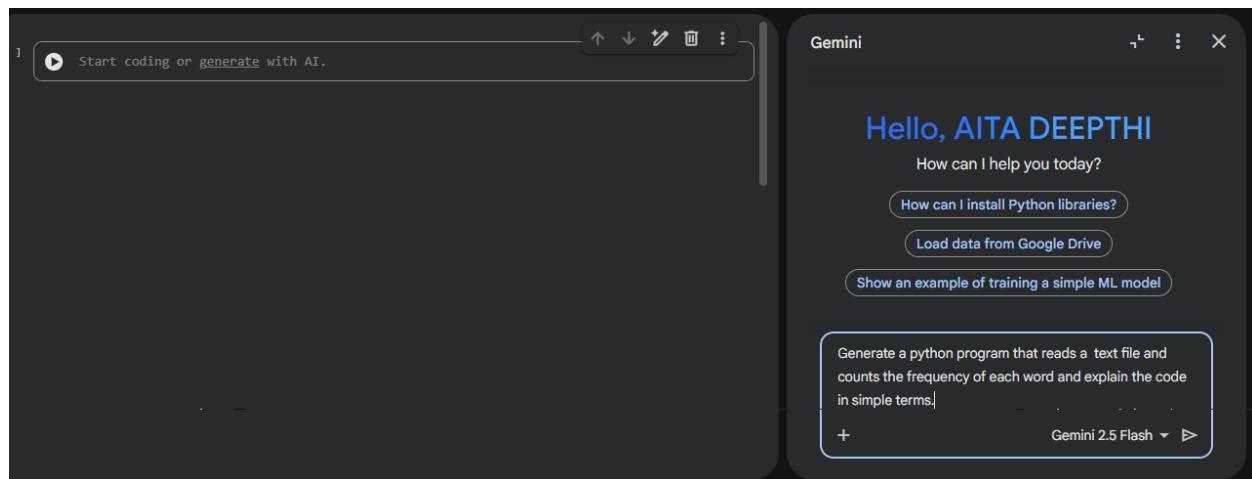
Batch: 32

Task 1: Word Frequency from Text File

You are analyzing log files for keyword frequency.

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

Prompt:



Generate a python program that reads a text file and counts the frequency of each word and explain the code in simple terms.

The screenshot shows a Jupyter Notebook interface with a dark theme. The left sidebar displays a 'Files' tree with a 'sample_data' folder containing a 'sample_text.txt' file. The main code cell contains the following Python code:

```
import collections
import re

def count_word_frequency(file_path):
    """
    Reads a text file, counts the frequency of each word, and returns the counts.
    """
    word_counts = collections.Counter()
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                # Convert to lowercase and remove punctuation
                words = re.findall(r'\b\w+\b', line.lower())
                word_counts.update(words)
    except FileNotFoundError:
        print(f"Error: The file at '{file_path}' was not found.")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None
    return word_counts

# --- Example Usage ---
# First, let's create a dummy text file for demonstration
```

The screenshot shows the continuation of the Jupyter Notebook interface. The code cell now includes the creation of a sample text file and the use of the function to count word frequencies:

```
print(f"Error: The file at '{file_path}' was not found.")
return None
except Exception as e:
    print(f"An error occurred: {e}")
    return None
return word_counts

# --- Example Usage ---
# First, let's create a dummy text file for demonstration
file_content = """This is a sample text file. \nThis file has sample text, \nand this text is for demonstration purposes.

with open('sample_text.txt', 'w', encoding='utf-8') as f:
    f.write(file_content)

# Now, use the function to count word frequencies
file_to_analyze = 'sample_text.txt'
frequencies = count_word_frequency(file_to_analyze)

if frequencies:
    print(f"Word Frequencies in '{file_to_analyze}':")
    # Display the top 10 most common words, or all if less than 10
    for word, count in frequencies.most_common(10):
        print(f"'{word}': {count}'")
```

Output:

```
... Word frequencies in '/content/sample.txt':  
'hello': 1  
'this': 1  
'is': 1  
'deepthi': 1
```

Explanation:

This code reads a text file and splits the content in the file into words. And each word is stored in a dictionary where the keys are words and the values are the count of the word.

The number of times the word repeats the count increases.

In the last the program gives an output of how many times the words appeared in the file.

Task 2: File Operations Using Cursor AI

You are automating basic file operations.

Task: Use Cursor AI to generate a program that:

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

Prompt:

Generate a python program that creates a text file. Write some sample text into it, and then read the file to display the content



Gemini 2.5 Flash ▾ ➤

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

Code:

```
[7] 0s # Define the filename and some sample text
file_name = 'my_sample_file.txt'
sample_text = """Hello, this is a sample text file.
It contains multiple lines of text.
We will write this content to a file and then read it back.
"""

# --- Step 1 & 2: Create and Write to the file ---
try:
    # Open the file in 'write' mode ('w'). If the file doesn't exist, it will be created.
    # If it exists, its content will be truncated (emptied) before writing.
    with open(file_name, 'w', encoding='utf-8') as file:
        file.write(sample_text)
    print(f"Successfully wrote content to '{file_name}'")
except Exception as e:
    print(f"Error writing to file: {e}")

# --- Step 3: Read the file and display its content ---
try:
    # Open the file in 'read' mode ('r')
    with open(file_name, 'r', encoding='utf-8') as file:
        read_content = file.read()
    print(f"\nContent of '{file_name}':\n")
    print(read_content)
except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found for reading.")
except Exception as e:
    print(f"Error reading file: {e}")
```

Output:

```
*** Successfully wrote content to 'my_sample_file.txt'

Content of 'my_sample_file.txt':

Hello, this is a sample text file.
It contains multiple lines of text.
We will write this content to a file and then read it back.
```

Explanation:

This program demonstrates basic file handling in Python using Cursor.

First, a text file is created and sample text is written into it.

Then, the same file is opened in read mode and its contents are displayed on the screen.

It shows how Python can be used to create, write, and read files easily.

Such operations are useful in automation and data storage tasks.

Task 3: CSV Data Analysis

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 python program to read csv file that contains numerical values and calculate the mean, minimum and maximum values

+ Gemini 2.5 Flash ▶

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

Generate a python program to read csv file that contains numerical values and calculate the mean, minimum and maximum values.

Code:

```
[4]  import pandas as pd
✓ os   import io

# --- Create a dummy CSV file for demonstration --- #
csv_data = """id,value1,value2
1,10,100
2,20,150
3,30,120
4,40,180
5,50,130
"""

# Save the dummy data to a CSV file
with open('sample_numbers.csv', 'w') as f:
    f.write(csv_data)

# --- Read the CSV file and calculate statistics --- #
def analyze_numerical_csv(file_path, column_name):
    try:
        df = pd.read_csv(file_path)
        if column_name not in df.columns:
            print(f"Error: Column '{column_name}' not found in the CSV file.")
            return None

        # Convert the column to numeric, coercing errors to NaN
        # This handles cases where some values might not be purely numerical
        df[column_name] = pd.to_numeric(df[column_name], errors='coerce')

        # Drop rows where the target column became NaN after conversion
        df.dropna(subset=[column_name], inplace=True)

        if df[column_name].empty:
            print(f"No valid numerical data found in column '{column_name}'.")
            return None
```

```
[4]  ✓  os
        df[column_name] = pd.to_numeric(df[column_name], errors='coerce')

        # Drop rows where the target column became NaN after conversion
        df.dropna(subset=[column_name], inplace=True)

        if df[column_name].empty:
            print(f"No valid numerical data found in column '{column_name}' .")
            return None

        mean_val = df[column_name].mean()
        min_val = df[column_name].min()
        max_val = df[column_name].max()

        return mean_val, min_val, max_val

    except FileNotFoundError:
        print(f"Error: The file at '{file_path}' was not found.")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

# Example Usage:
file_to_analyze = 'sample_numbers.csv'
column_to_analyze = 'value1' # Change this to 'value2' or any other numerical column name

results = analyze_numerical_csv(file_to_analyze, column_to_analyze)

if results:
    mean_val, min_val, max_val = results
    print(f"\nStatistics for column '{column_to_analyze}' in '{file_to_analyze}':")
    print(f"Mean: {mean_val:.2f}")
    print(f"Minimum: {min_val:.2f}")
    print(f"Maximum: {max_val:.2f}")
```

Output:

```
*** Statistics for column 'value1' in 'sample_numbers.csv':
Mean: 30.00
Minimum: 10.00
Maximum: 50.00
```

Explanation:

This program reads data from a CSV and it extracts numerical values from the file and calculates their mean, minimum and maximum.

CSV data analysis is used in data processing and analytics application.

Task 4: Sorting Lists – Manual vs Built-in

You are reviewing algorithm choices for efficiency.

- ❖ Task: Use Gemini to generate:
 - Bubble sort
 - Python's built-in sort()
 - Compare both implementations.

Prompt:

Generate a python program for bubble sort

- Certainly! Here's a Python program that implements the bubble sort algorithm, along with a step-by-step explanation.

Working...

Code:

Bubble sort

```
[5]  ✓ 0s
▶ 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(f"Original list: {my_list}")

sorted_list = bubble_sort(my_list)
print(f"Sorted list: {sorted_list}")

my_list_2 = [5, 1, 4, 2, 8]
print(f"Original list: {my_list_2}")
sorted_list_2 = bubble_sort(my_list_2)
print(f"Sorted list: {sorted_list_2}")
```

Built-in sort:

```
[5]  ✓ 0s
▶ # --- Using list.sort() method --- #
# This method sorts the list in-place (modifies the original list) and returns None.

my_list_sort = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list (for .sort()): {my_list_sort}")
my_list_sort.sort()
print(f"Sorted list (using .sort()): {my_list_sort}")

print("\n---")

# --- Using sorted() function --- #
# This function returns a new sorted list, leaving the original list unchanged.

my_list_sorted = [5, 1, 4, 2, 8]
print(f"Original list (for sorted()): {my_list_sorted}")
sorted_list_new = sorted(my_list_sorted)
print(f"Sorted list (using sorted()): {sorted_list_new}")
print(f"Original list remains unchanged: {my_list_sorted}")

# Example with custom sorting (descending order)
print("\n---")
my_list_desc = [10, 50, 20, 40, 30]
print(f"Original list (descending example): {my_list_desc}")
sorted_desc = sorted(my_list_desc, reverse=True)
print(f"Sorted list (descending): {sorted_desc}")
```

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]
```

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

---
Original list (for sorted()): [5, 1, 4, 2, 8]
Sorted list (using sorted()): [1, 2, 4, 5, 8]
Original list remains unchanged: [5, 1, 4, 2, 8]

---
Original list (descending example): [10, 50, 20, 40, 30]
Sorted list (descending): [50, 40, 30, 20, 10]
```

Explanation:

Bubble sort is a simple sorting algorithm that repeatedly compares and swaps adjacent elements. It is easy to understand but inefficient for large data sets. Python's built-in sort function is shorter, optimized and much faster. The built-in method should be preferred in real-world applications