# AI Assisted Co

ASSIGNMENT 2.3
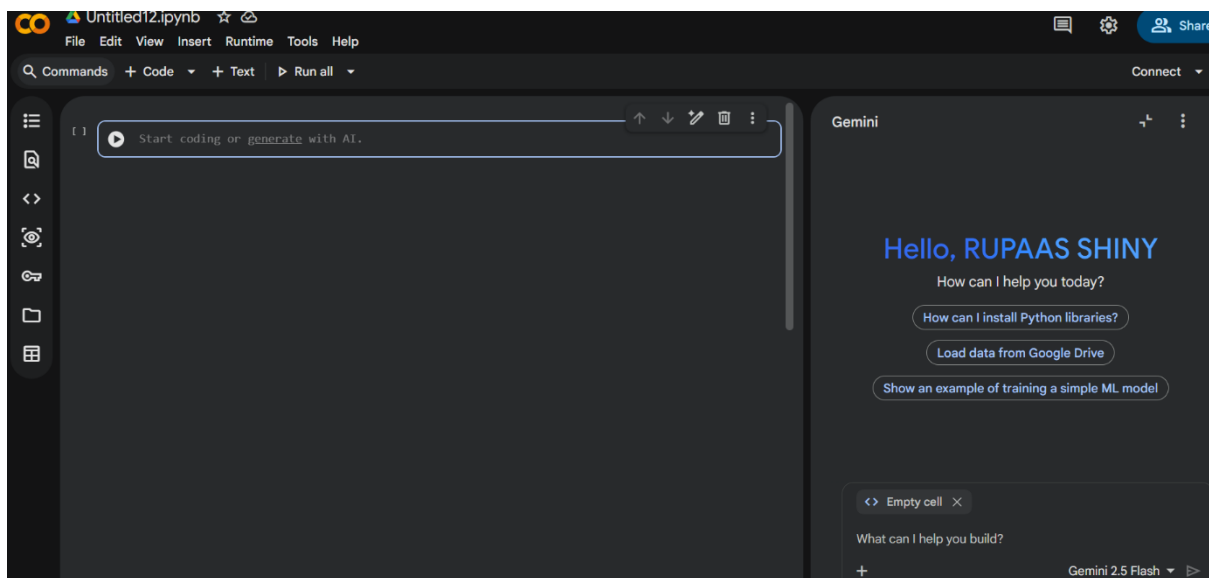
Name:K.Rupaas shyni

2303a52417

Batch:32

Question

Task 1:
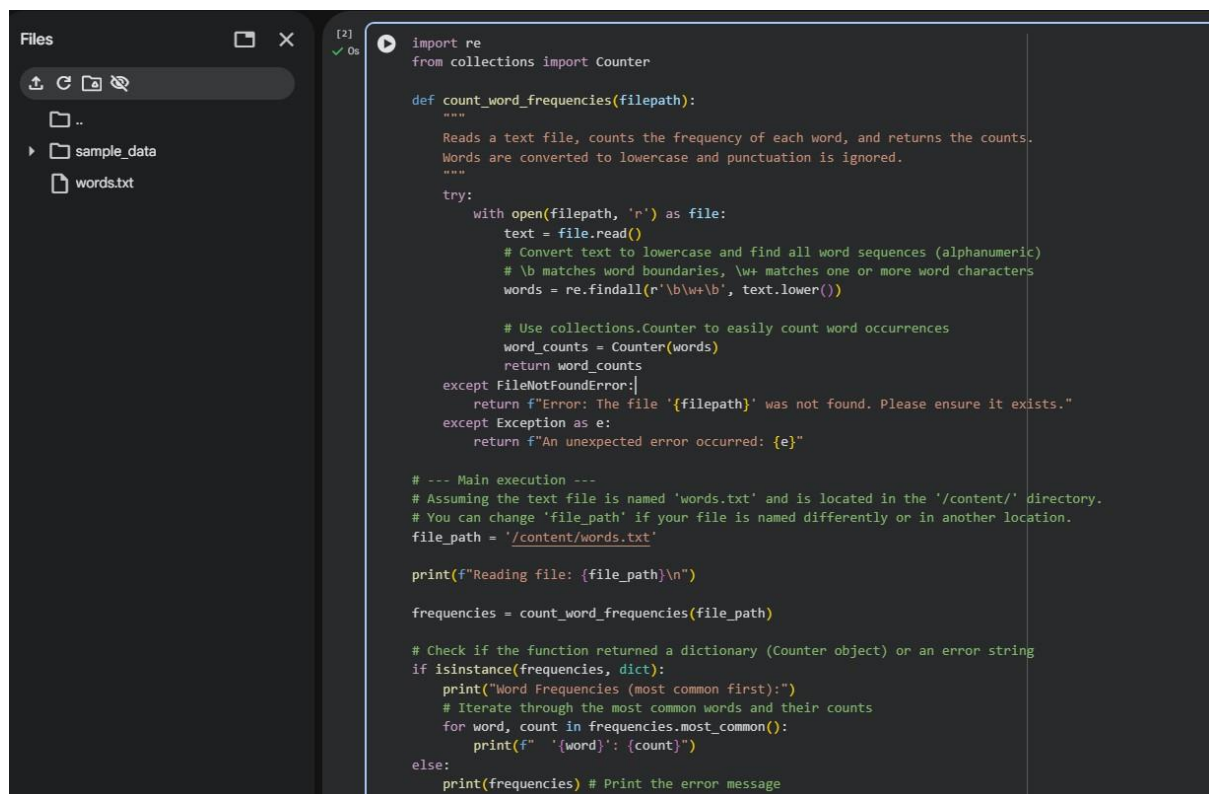
Word Frequency from Text File

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



Create a Python program that creates a text file, writes sample text into it, then reads the file and displays its contents.

Add clear comments in the code.

```
import re
from collections import Counter

def count_word_frequencies(filepath):
    """
    Reads a text file, counts the frequency of each word, and returns the counts.
    Words are converted to lowercase and punctuation is ignored.
    """
    try:
        with open(filepath, 'r') as file:
            text = file.read()
            # Convert text to lowercase and find all word sequences (alphanumeric)
            # \b matches word boundaries, \w+ matches one or more word characters
            words = re.findall(r'\b\w+\b', text.lower())

            # Use collections.Counter to easily count word occurrences
            word_counts = Counter(words)
            return word_counts
    except FileNotFoundError:
        return f"Error: The file '{filepath}' was not found. Please ensure it exists."
    except Exception as e:
        return f"An unexpected error occurred: {e}"

# --- Main execution ---
# Assuming the text file is named 'words.txt' and is located in the '/content/' directory.
# You can change 'file_path' if your file is named differently or in another location.
file_path = '/content/words.txt'

print(f"Reading file: {file_path}\n")

frequencies = count_word_frequencies(file_path)

# Check if the function returned a dictionary (Counter object) or an error string
if isinstance(frequencies, dict):
    print("Word Frequencies (most common first):")
    # Iterate through the most common words and their counts
    for word, count in frequencies.most_common():
        print(f"   '{word}': {count}")
else:
    print(frequencies) # Print the error message
```
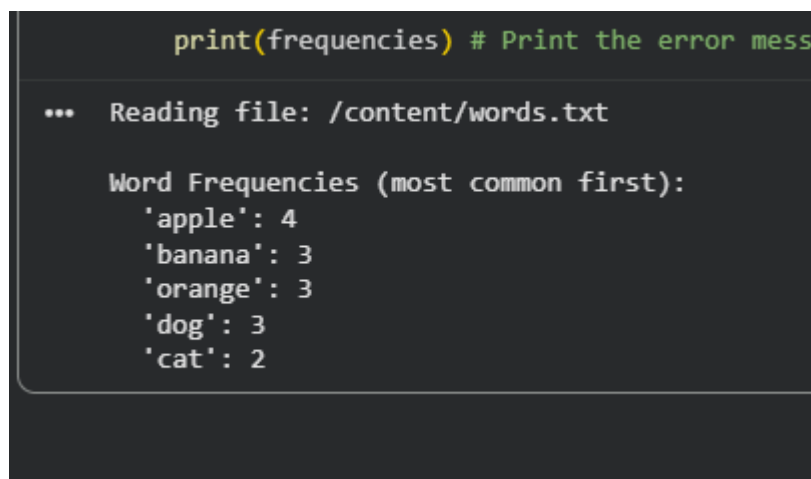
Output:



```
        print(frequencies) # Print the error mess

...  Reading file: /content/words.txt

     Word Frequencies (most common first):
        'apple': 4
        'banana': 3
        'orange': 3
        'dog': 3
        'cat': 2
```

Explaination:

This program reads a text file and splits the content into words.

Each word is stored in a dictionary where the key is the word and the value is its count. If a word appears again, its count is increased.

Finally, the program prints how many times each word appears in the file. This helps in analyzing text data and finding frequently used words.

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 python program, which creates a text file, write sample txt eg.fruits in that, read and display that content.

```python
# Python program to create, write, read, and display a text file
filename = "fruits.txt"

# Write sample text to the file
with open(filename, 'w') as file:
    file.write("Fruits\n")
    file.write("Apple\n")
    file.write("Banana\n")
    file.write("Orange\n")
    file.write("Grape\n")
    file.write("Mango\n")

print(f"File '{filename}' has been created and written to.\n")

# Read and display the content
print("Reading and displaying the content:")
print("-" * 40)
with open(filename, 'r') as file:
    content = file.read()
    print(content)
print("-" * 40)

# Alternative: Read line by line
print("\nReading line by line:")
print("-" * 40)
with open(filename, 'r') as file:
    for line_num, line in enumerate(file, 1):
        print(f"Line {line_num}: {line.strip()}")
```

Explaination:

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.

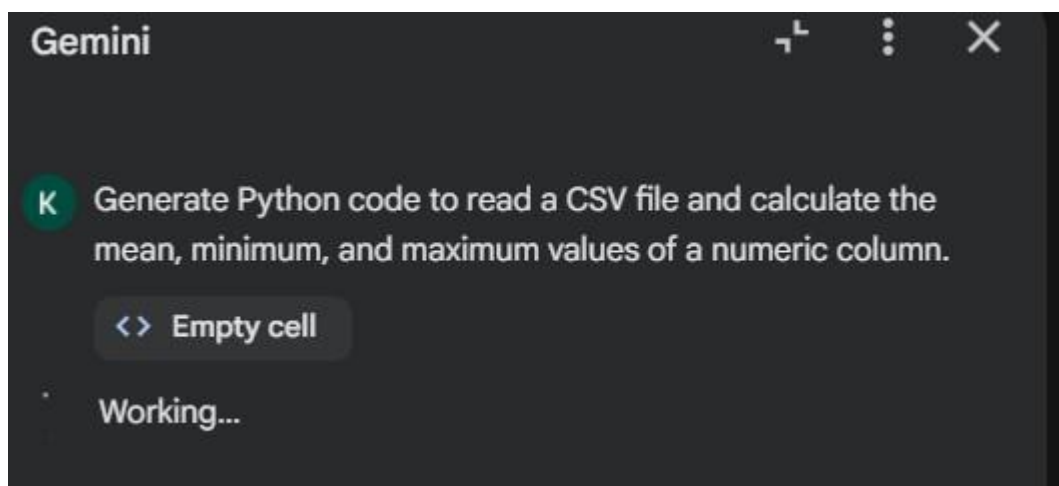Such operations are useful in automation and data storage tasks.

Task 3: CSV Data Analysis

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 Python code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.

```python
import pandas as pd
import io

# Uncomment the lines below and replace 'your_file.csv' with your CSV file path
try:
    df = pd.read_csv('scores.csv')
except FileNotFoundError:
    print("Error: CSV file not found. Please check the path.")
    exit()

# Specify the numeric column you want to analyze
column_name = 'score'

# Check if the column exists in the DataFrame
if column_name in df.columns:
    # Ensure the column is numeric (e.g., float or int)
    # pd.to_numeric will convert values to numeric, coercing errors to NaN
    numeric_column = pd.to_numeric(df[column_name], errors='coerce')

    # Drop rows where the numeric conversion resulted in NaN (non-numeric values)
    numeric_column = numeric_column.dropna()

    if not numeric_column.empty:
        # Calculate mean, minimum, and maximum
        mean_value = numeric_column.mean()
        min_value = numeric_column.min()
        max_value = numeric_column.max()

        print(f"Statistics 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"Column '{column_name}' contains no valid numeric data after cleaning.")
else:
    print(f"Error: Column '{column_name}' not found in the CSV file. Available columns: {df.columns.tolist()}")
```

Output:



This program reads data from a CSV file using Python.

It extracts numerical values from a column and calculates the mean, minimum, and maximum.

CSV analysis is used in data processing and analytics applications.


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:

```python
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):
            # 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 using Bubble Sort:", sorted_list)

my_list_2 = [5, 1, 4, 2, 8]
print("\nOriginal list 2:", my_list_2)

sorted_list_2 = bubble_sort(my_list_2)
print("Sorted list 2 using Bubble Sort:", sorted_list_2)
```

```
op/AIAC/assig2.py
riginal list: [64, 34, 25, 12, 22, 11, 90]
orted list using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]

riginal list 2: [5, 1, 4, 2, 8]
orted list 2 using Bubble Sort: [1, 2, 4, 5, 8]
S C:\Users\rupaa\OneDrive\Desktop\AIAC> |
                                                        Ln 90 C
```

Sorting using sort() function:

```python
# --- Using the list.sort() method ---
# This method sorts the list in-place (modifies the original list) and returns None.

my_list_1 = [64, 34, 25, 12, 22, 11, 90]
print("Original list (list.sort()):", my_list_1)

my_list_1.sort()
print("Sorted list (list.sort()):", my_list_1)

# You can also sort in descending order
my_list_2 = ['banana', 'apple', 'cherry', 'date']
print("\nOriginal list (descending):", my_list_2)

my_list_2.sort(reverse=True)
print("Sorted list (descending):", my_list_2)
# --- Using the sorted() function ---
```

```
op/AIAC/assig2.py
Original list (list.sort()): [64, 34, 25, 12, 22, 11, 90]
Sorted list (list.sort()): [11, 12, 22, 25, 34, 64, 90]

Original list (descending): ['banana', 'apple', 'cherry', 'date']
Sorted list (descending): ['date', 'cherry', 'banana', 'apple']
S C:\Users\rupaa\OneDrive\Desktop\AIAC> |
                                                        Ln 107 Col 1   Sna
```

Explaination:

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.