AI ASSISTED CODING

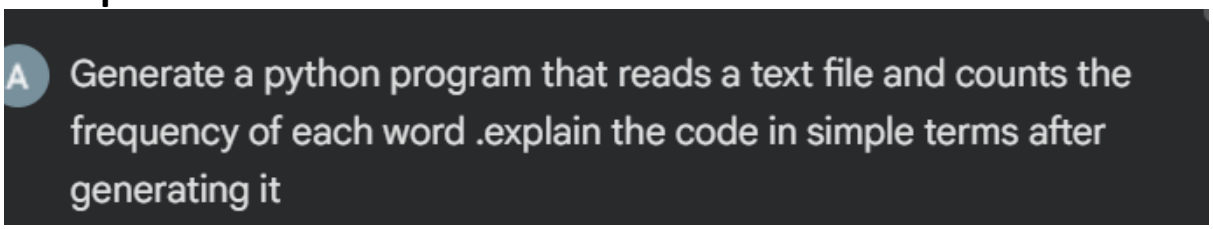
ASSIGNMENT-2.3
2303A52364
BATCH-32

TASK-1

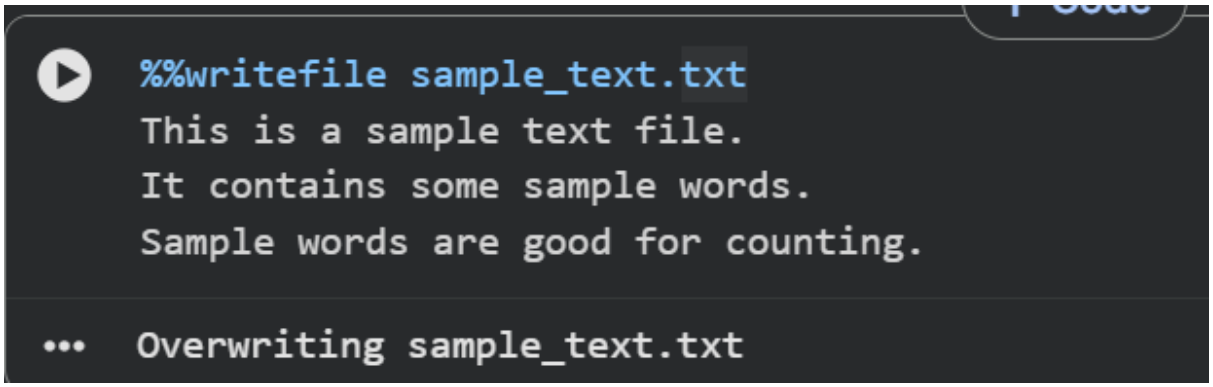
Task 1: Word Frequency from Text File
scenario:
You are analyzing log files for keyword frequency.
Task:
Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.
Expected Output:

- Working code
- Explanation
- Screenshot

Prompt:



Code:



```python
from collections import Counter
import re

def count_word_frequency(filepath):
    """
    Reads a text file, counts the frequency of each word,
    and returns a dictionary of word counts.
    """
    word_counts = Counter()
    try:
        with open(filepath, 'r', encoding='utf-8') as file:
            text = file.read()
            # Convert to lowercase and find all words using regex
            words = re.findall(r'\b\w+\b', text.lower())
            word_counts.update(words)
    except FileNotFoundError:
        print(f"Error: File not found at {filepath}")
    return word_counts

# Specify the path to your text file
file_path = 'sample_text.txt'

# Call the function to count word frequencies
frequencies = count_word_frequency(file_path)

# Print the results
print("Word Frequencies:")
for word, count in frequencies.most_common():
    print(f"{word}: {count}")
```

Output:

```
Word Frequencies:
sample: 3
words: 2
this: 1
is: 1
a: 1
text: 1
file: 1
it: 1
contains: 1
some: 1
are: 1
good: 1
for: 1
counting: 1
```

Explanation:

This Python program is used to analyze a log file and count the frequency of each word. It opens the text file and reads it line by line to handle large files efficiently. Each line is converted to lowercase and split into individual words so that the same words with different cases are treated equally. Punctuation marks are removed to ensure accurate counting. A dictionary is used to store each word as a key and its occurrence count as the value. If a word appears again, its count is increased. Finally, the program prints all the words along with their frequencies, helping to identify commonly occurring keywords in the log file.

Task 2 :

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
Expected Output:

- Functional code
- Cursor AI screenshots

Prompt:

A code editor showing Python file operations code with a newly created sample.txt file

Code:

```
%%writefile sample.txt
Hello, this is the initial content of sample.txt.
This is the second line.
```

```python
# 1. Reading from a file
print("--- Reading 'sample.txt' ---")
with open('sample.txt', 'r') as file:
    content = file.read()
    print(content)

# 2. Writing to a file (this overwrites existing content)
print("\n--- Writing new content to 'sample.txt' (overwrites) ---")
with open('sample.txt', 'w') as file:
    file.write("New content has been written.\n")
    file.write("This replaced the old content.")

# Verify by reading again
print("--- Reading 'sample.txt' after writing ---")
with open('sample.txt', 'r') as file:
    content = file.read()
    print(content)

# 3. Appending to a file (adds to the end)
print("\n--- Appending content to 'sample.txt' ---")
with open('sample.txt', 'a') as file:
    file.write("\nThis line was appended.\n")
    file.write("Another appended line.")

# Verify by reading again
print("--- Reading 'sample.txt' after appending ---")
with open('sample.txt', 'r') as file:
    content = file.read()
    print(content)
```

Output:

```
--- Reading 'sample.txt' ---
Hello, this is the initial content of sample.txt.
This is the second line.


--- Writing new content to 'sample.txt' (overwrites) ---
--- Reading 'sample.txt' after writing ---
New content has been written.
This replaced the old content.

--- Appending content to 'sample.txt' ---
--- Reading 'sample.txt' after appending ---
New content has been written.
This replaced the old content.
This line was appended.
Another appended line.
```

Explanation:

This program demonstrates basic file operations in Python. First, a text file named sample.txt is created in write mode and sample text is written into it. The file is then closed to save the data. Next, the same file is opened in read mode, and its content is read and displayed on the screen. This helps in understanding how files are created, written, and read using Python.

Task3:

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

Prompt:

Python code cell in Google Colab calculating mean, minimum, and maximum from a CSV file.

Code:

```
%%writefile sample_text.txt
This is a sample text file.
It contains some sample words.
Sample words are good for counting.
```

```
Overwriting sample_text.txt
```

```
%%writefile sample_data.csv
Name,Age,Score
Alice,25,85
Bob,30,92
Charlie,22,78
David,28,95
Eve,35,88
```

```
Writing sample_data.csv
```

```python
import pandas as pd

# Load the CSV file into a pandas DataFrame
try:
    df = pd.read_csv('sample_data.csv')

    # Display the DataFrame to see its content
    print("--- DataFrame Content ---")
    print(df)
    print("\n")

    # Calculate mean, minimum, and maximum for a specific numeric column, e.g., 'Score'
    if 'Score' in df.columns:
        mean_score = df['Score'].mean()
        min_score = df['Score'].min()
        max_score = df['Score'].max()

        print(f"Mean Score: {mean_score:.2f}")
        print(f"Minimum Score: {min_score}")
        print(f"Maximum Score: {max_score}")
    else:
        print("Error: 'Score' column not found in the CSV file.")

except FileNotFoundError:
    print("Error: 'sample_data.csv' not found. Please ensure it has been created.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Output:

```
--- DataFrame Content ---
      Name  Age  Score
0    Alice   25     85
1      Bob   30     92
2  Charlie   22     78
3    David   28     95
4      Eve   35     88


Mean Score: 87.60
Minimum Score: 78
Maximum Score: 95
```

Explanation:

This program uses the Pandas library to read a CSV file in Google Colab. The CSV file contains numeric data in a column named “Value”. After reading the file, the program calculates the mean, minimum, and maximum values using built-in Pandas functions. Finally, the calculated results are printed, making it easy to analyze structured data.

Task 4:

Prompt:

Gemini generating Python code for bubble sort algorithm in a code editor.

Code:

```
%%writefile sample_text.txt
This is a sample text file.
It contains some sample words.
Sample words are good for counting.
```

```
Overwriting sample_text.txt
```

```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):
            # 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.copy()) # Use .copy() to avoid modifying original list in place if you want to keep it
print("Sorted list (Bubble Sort):", sorted_list)
```

Output:

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

Explanation:

Bubble sort is a manual sorting algorithm that compares adjacent elements and swaps them repeatedly, which makes it easy to understand but slow for large lists. Python's built-in sort() method is highly optimized and much faster because it uses an efficient algorithm internally. Therefore, bubble sort is useful for learning concepts, while the built-in sort is preferred for real applications.