Assignment-2.3
K.Akshay
2303A51817
Batch-26

Assignment Number:2.3(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to comple***
1	Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI **Lab Objectives:** ❖ To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. ❖ To understand and use Cursor AI for code generation, explanation, and refactoring. ❖ To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. ❖ To perform code optimization and documentation using AI tools. **Lab Outcomes (LOs):** After completing this lab, students will be able to: ❖ Generate Python code using Google Gemini in Google Colab. ❖ Analyze the effectiveness of code explanations and suggestions by Gemini. ❖ Set up and use Cursor AI for AI-powered coding assistance. ❖ Evaluate and refactor code using Cursor AI features. ❖ Compare AI tool behavior and code quality across different platforms.	Week1 - Monday
	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 :

By analyzing log files for keyword frequency , generate Python code that reads a text file and counts word frequency

Code:

```python
import os

# Create a dummy log file for demonstration
file_content = """
INFO: User logged in from IP 192.168.1.100.
ERROR: Failed attempt to access /admin from 192.168.1.101.
INFO: Data sync started.
WARNING: Disk space low on /dev/sda1.
INFO: User logged in from IP 192.168.1.102.
ERROR: Database connection lost.
INFO: Data sync completed.
"""

with open('sample_log.txt', 'w') as f:
    f.write(file_content)

print("Created 'sample_log.txt' for demonstration.")
```

```
Created 'sample_log.txt' for demonstration.
```

```python
from collections import Counter
import re
```

```python
def count_word_frequency(filepath):
    """
    Reads a text file, counts the frequency of each word, and returns the counts.
    """
    word_list = []
    try:
        with open(filepath, 'r') as f:
            for line in f:
                # Convert to lowercase and find all words (alphanumeric sequences)
                words_in_line = re.findall(r'\b\w+\b', line.lower())
                word_list.extend(words_in_line)
    except FileNotFoundError:
        print(f"Error: The file '{filepath}' was not found.")
        return None

    word_counts = Counter(word_list)
    return word_counts

# Specify the path to your log file
log_file_path = 'sample_log.txt'

# Get word frequencies
frequencies = count_word_frequency(log_file_path)
```

```python
if frequencies:
    print(f"Word frequencies in '{log_file_path}':")
    # Display the top 10 most common words
    for word, count in frequencies.most_common(10):
        print(f"'{word}': {count}")
```

Output :

```
••• Word frequencies in 'sample_log.txt':
    'info': 4
    'from': 3
    '192': 3
    '168': 3
    '1': 3
    'user': 2
    'logged': 2
    'in': 2
    'ip': 2
    'error': 2
```

Explanation :

This code is just a simple way to read a file and count how many times each word appears. First, it opens the file you give it and reads each line. It changes all the words to lowercase so that “Error” and “error” are treated the same. Then it uses a small pattern to pick out words, ignoring punctuation, and adds them to a list. It shows the ten most common words in the file with their counts. In short, the program reads your log file, counts the words, and tells you which ones appear most often.

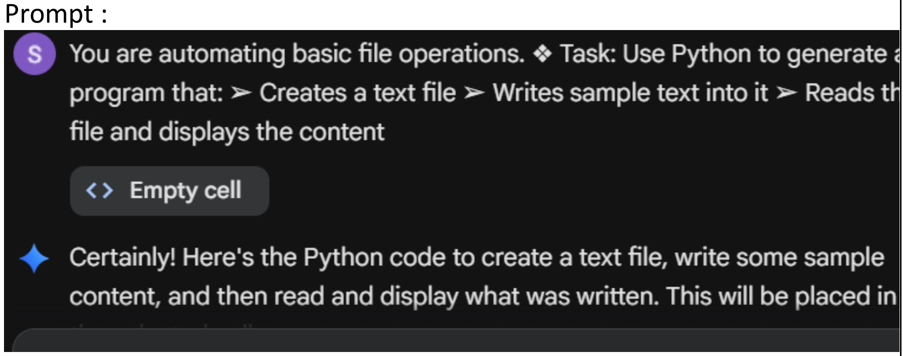
Task 2: File Operations Using Cursor AI

❖ **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 :



Code :

```python
# 1. Define the filename and sample content
file_name = 'my_sample_file.txt'
sample_text = """This is line 1 of the sample file.
This is the second line.
And finally, the third line of text."""

# 2. Create the text file and write sample text into it
try:
    with open(file_name, 'w') as f:
        f.write(sample_text)
    print(f"Successfully created '{file_name}' and wrote content.\n")
except IOError as e:
    print(f"Error writing to file: {e}")

# 3. Read the file and display its content
try:
    with open(file_name, 'r') as f:
        read_content = f.read()
    print(f"Content of '{file_name}':\n---\n{read_content}\n---")
except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found.")
except IOError as e:
    print(f"Error reading file: {e}")
```

Output :

```
Successfully created 'my_sample_file.txt' and wrote content.

Content of 'my_sample_file.txt':
---
This is line 1 of the sample file.
This is the second line.
And finally, the third line of text.
---
```

Explanation : This code simply creates a text file, writes some sample lines into it, and then reads the file back to show what was written. First, it sets the file name and the text you want to save. Then it opens the file in write mode, adds the text, and confirms it was saved. After that, it opens the same file in read mode, reads all the content, and prints it out. If the file is missing or there's a problem while writing or reading, it shows an error message instead of stopping.

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 :

S Read a CSV file in Python, pick a numeric column, calculate its mean, minimum, and maximum, and print the results clearly.

Sure! Here's the Python code to read a CSV file, calculate the mean, minimum, and maximum of a numeric column, and display the results. start by creating a dummy CSV file for you to test with.

Code :

```python
import pandas as pd
import numpy as np
import os

# 1. Create a dummy CSV file for demonstration
csv_file_name = 'sample_data.csv'

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [24, 27, 22, 32, 29],
    'Score': [85, 91, 78, 95, 88],
    'City': ['NY', 'LA', 'CHI', 'NY', 'LA']
}
df_dummy = pd.DataFrame(data)
df_dummy.to_csv(csv_file_name, index=False)

print(f"Created '{csv_file_name}' with sample data:")
print(df_dummy.head())
```

```
Created 'sample_data.csv' with sample data:
      Name  Age  Score City
0    Alice   24     85   NY
1      Bob   27     91   LA
2  Charlie   22     78  CHI
3    David   32     95   NY
4      Eve   29     88   LA
```

```python
import pandas as pd

# 2. Read the CSV file
csv_file_name = 'sample_data.csv' # Make sure this matches the file you created or have
try:
    df = pd.read_csv(csv_file_name)
    print(f"\nSuccessfully read '{csv_file_name}'. First 5 rows:\n")
    display(df.head())

    # 3. Pick a numeric column for calculations (e.g., 'Score')
    numeric_column_name = 'Score' # You can change this to 'Age' or another numeric column

    if numeric_column_name in df.columns and pd.api.types.is_numeric_dtype(df[numeric_column_name]):
        # 4. Calculate mean, minimum, and maximum
        mean_value = df[numeric_column_name].mean()
        min_value = df[numeric_column_name].min()
        max_value = df[numeric_column_name].max()

        # 5. Print the results clearly
        print(f"\nStatistics for column '{numeric_column_name}':")
        print(f"  Mean: {mean_value:.2f}")
        print(f"  Minimum: {min_value}")
        print(f"  Maximum: {max_value}")
```

```
        print(f"  Maximum: {max_
    else:
        print(f"Error: Column '{numeric_column_name}' not found or is not numeric.")

except FileNotFoundError:
    print(f"Error: The file '{csv_file_name}' was not found. Please make sure it exists.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Output :

```
Successfully read 'sample_data.csv'. First 5 rows:
```

	Name	Age	Score	City
0	Alice	24	85	NY
1	Bob	27	91	LA
2	Charlie	22	78	CHI
3	David	32	95	NY
4	Eve	29	88	LA

```
Statistics for column 'Score':
  Mean: 87.40
  Minimum: 78
  Maximum: 95
```

Explanation : This code reads a CSV file, shows the first few rows, then checks if a chosen column (like "Score") is numeric. If it is, it calculates the mean, minimum, and maximum values and prints them. If the file is missing or the column isn't valid, it shows an error message instead.

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

Prompt :

Write Python code to implement bubble sort and also use the built-in sort() function, then compare their efficiency

Great! I've put together a plan to implement both bubble sort and Python's

Code :

```python
def bubble_sort(arr):
    """
    Implements the Bubble Sort algorithm to sort a list in ascending order.
    The list is sorted in-place.
    """
    n = len(arr)
    # Traverse through all array elements
    for i in range(n - 1):
        # Last i elements are already in place
        for j in range(n - 1 - i):
            # 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}")

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

Output :

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

```python
def bubble_sort(arr):
    """
    Implements the Bubble Sort algorithm to sort a list in ascending order.
    The list is sorted in-place.
    """
    n = len(arr)
    # Outer loop: iterate from n-1 down to 1
    # 'i' represents the last index of the unsorted portion
    for i in range(n - 1, 0, -1):
        # Inner loop: iterate from 0 up to i-1
        # 'j' goes through the unsorted part, comparing adjacent elements
        for j in range(i):
            # 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}")

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

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

```python
import random

# 1. Create a sample list of unsorted numbers
original_list = [random.randint(1, 100) for _ in range(10)]
print(f"Original list: {original_list}")

# 2. Create a copy of this list, as list.sort() sorts in-place
sorted_list = original_list.copy()

# 3. Apply the sort() method directly to the copied list to sort it in ascending order
sorted_list.sort()

# 4. Print both the original and the sorted list to demonstrate the effect of the built-in sort
print(f"Sorted list (using list.sort()): {sorted_list}")
```

```
Original list: [42, 67, 2, 8, 38, 3, 40, 70, 98, 44]
Sorted list (using list.sort()): [2, 3, 8, 38, 40, 42, 44, 67, 70, 98]
```

```python
import random

# 1. Define the desired number of elements for the test list
list_size = 5000  # You can change this to 1000, 5000, 10000, etc.

# 2. Populate 'test_data' with 'list_size' random integers
# Using random.randint(1, 10000) to generate numbers within a reasonable range
test_data = [random.randint(1, 10000) for _ in range(list_size)]

# 3. Print the first few elements and the length of the generated list to verify
print(f"Generated a list of {len(test_data)} random integers.")
print(f"First 10 elements: {test_data[:10]}...")
```

```
Generated a list of 5000 random integers.
First 10 elements: [617, 9925, 7006, 1656, 8090, 6684, 3950, 3242, 111, 2825]...
```

```python
import time

# 1. Create a copy of the test_data list for bubble sort
bubble_sort_data = test_data.copy()

# 2. Measure execution time for Bubble Sort
start_time_bubble = time.time()
bubble_sort(bubble_sort_data)
end_time_bubble = time.time()
time_taken_bubble = end_time_bubble - start_time_bubble

# 3. Create another copy of the test_data list for built-in sort
builtin_sort_data = test_data.copy()

# 4. Measure execution time for built-in list.sort()
start_time_builtin = time.time()
builtin_sort_data.sort()
end_time_builtin = time.time()
time_taken_builtin = end_time_builtin - start_time_builtin

# 5. Print the comparison results
print(f"Comparison of Sorting Algorithm Performance (List Size: {list_size})")
print("--------------------------------------------------")
print(f"Bubble Sort execution time: {time_taken_bubble:.6f} seconds")
print(f"Built-in sort() execution time: {time_taken_builtin:.6f} seconds")
print("--------------------------------------------------")

if time_taken_bubble > time_taken_builtin:
    print(f"The built-in sort() was approximately {time_taken_bubble / time_taken_builtin:.2f} times faster than Bubble Sort.")
elif time_taken_builtin > time_taken_bubble:
    print(f"Bubble Sort was approximately {time_taken_builtin / time_taken_bubble:.2f} times faster than the built-in sort() (highly unlikely for large lists).")
else:
    print("Both algorithms took approximately the same amount of time.")
```

Explanation :

This code measures how long Bubble Sort and Python's built-in sort() take on the same list, then prints the times and shows which method was faster. Bubble Sort is usually much slower because it's O(n^2), while the built-in sort is optimized at O(n\log n) .