

ASSIGNMENT-2.3

Name:Ch.Jyothika

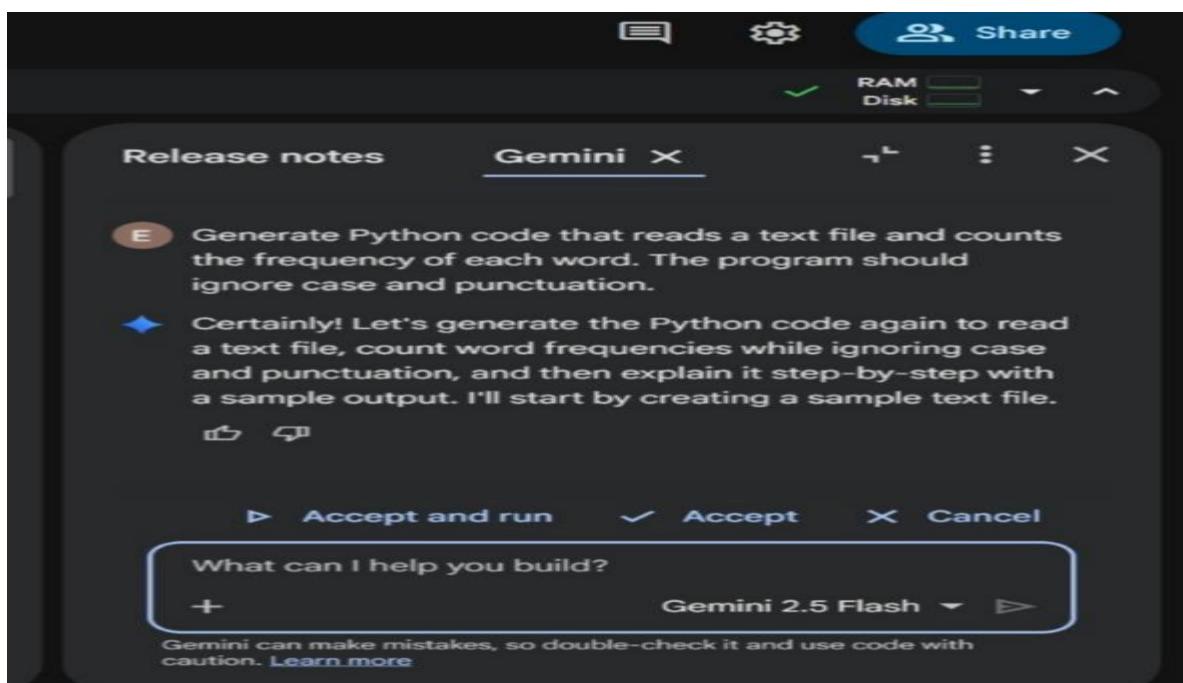
Ht.no:2303A51280

Batch:05

TASK:01

Word Frequency from Text File

PROMPT:



CODE:

The screenshot shows a Google Colab notebook titled "week-2.3.ipynb". In the code editor, there are two cells. The first cell contains code to create a sample text file and print its content:

```

sample_text = """This is a sample text file. It contains some words, and it has punctuation like commas, periods, and exclamation marks.

Created 'sample.txt' with the following content:
This is a sample text file. It contains some words, and it has punctuation like commas, periods, and exclamation marks."""
with open('sample.txt', 'w') as f:
    f.write(sample_text)

print("Created 'sample.txt' with the following content:")
print(sample_text)

```

The second cell contains code to count word frequencies while ignoring case and punctuation:

```

import collections
import re

def count_word_frequency(filepath):
    """
    Reads a text file, counts the frequency of each word,
    ignoring case and punctuation.
    """
    with open(filepath, 'r') as file:
        text = file.read()

    # Convert to lowercase
    text = text.lower()

```

A sidebar on the right shows the "Release notes" tab selected, displaying a conversation with Gemini:

- E Generate Python code that reads a text file and counts the frequency of each word. The program should ignore case and punctuation.
- ◆ Certainly! Let's generate the Python code again to read a text file, count word frequencies while ignoring case and punctuation, and then explain it step-by-step with a sample output. I'll start by creating a sample text file.
- Great! The sample file has been created, and the word frequencies have been calculated and displayed. Now, let's go through the code step-by-step to explain how

Accept Cancel

What can I help you build?

Gemini 2.5 Flash

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

The screenshot shows the same Google Colab notebook. The code editor now contains the completed function:

```

with open(filepath, 'r') as file:
    text = file.read()

# Convert to lowercase
text = text.lower()

# Remove punctuation using a regular expression
# This replaces anything that is not a letter, number, or space with an empty string
text = re.sub(r'[^a-zA-Z\s]', '', text)

# Split the text into words
words = text.split()

# Count word frequencies
word_counts = collections.Counter(words)

return word_counts

# Example usage with the 'sample.txt' file
file_path = 'sample.txt'
frequency_map = count_word_frequency(file_path)

print("\nWord Frequencies:")
for word, count in frequency_map.items():
    print(f"{word}: {count}")

```

The sidebar shows the same conversation with Gemini, indicating the code has been generated and explained.

Variables Terminal 09:36 Python 3

OUTPUT:

The screenshot shows the terminal output of the generated Python code. The output is a dictionary of word frequencies:

```

Word Frequencies:
'this': 2
'is': 2
'a': 2
'sample': 1
'text': 1
'file': 1
'it': 2
'contains': 1
'some': 1
'words': 2
'and': 2
'has': 1
'punctuation': 1
'like': 1
'commas': 1
'periods': 1
'lets': 1
'count': 1
'good': 1
'example': 1

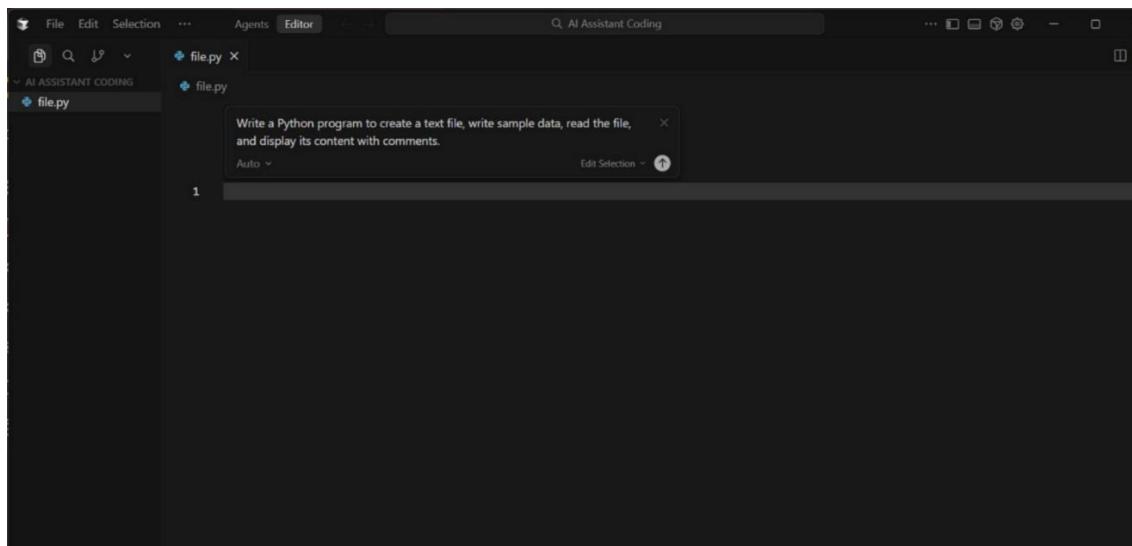
```

EXPLANATION:

- The program reads the contents of a text file.
- It converts all text to lowercase to avoid case mismatch.
- Punctuation marks are removed to ensure accurate word counting.
- Each word is counted using a dictionary.
- The final output displays each word along with its frequency.

TASK-02:

PROMPT:



CODE:

The screenshot shows the Visual Studio Code interface. The left sidebar displays files: file.py, another_file.txt, and sample.txt. The main editor tab is file.py, containing the following Python code:

```
1 # Create and write data to the file
2 filename = 'sample.txt'
3 with open(filename, 'w') as file:
4     file.write('Hello, world!')
5
6 # Read and display the content of the file
7 with open(filename, 'r') as file:
8     content = file.read()
9     print(content)
10
11 # Append more data to the file
12 with open(filename, 'a') as file:
13     file.write('\nThis is some more data.')
14
15 # Read and display the updated content of the file
16 with open(filename, 'r') as file:
17     content = file.read()
18     print(content)
19
20 # Create another file and write data to it
21 another_filename = 'another_file.txt'
22 with open(another_filename, 'w') as file:
23     file.write('This is another file with new content.')
24
```

The terminal below shows the output of running the script:

```
This is another file with new content.
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/file.py"
Hello, world!
Hello, world!
This is some more data.
This is another file with new content.
PS C:\AI Assistant Coding>
```

OUTPUT:

The screenshot shows the Visual Studio Code interface with the terminal tab selected. It displays the same output as the previous screenshot:

```
This is another file with new content.
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/file.py"
Hello, world!
Hello, world!
This is some more data.
This is another file with new content.
PS C:\AI Assistant Coding>
```

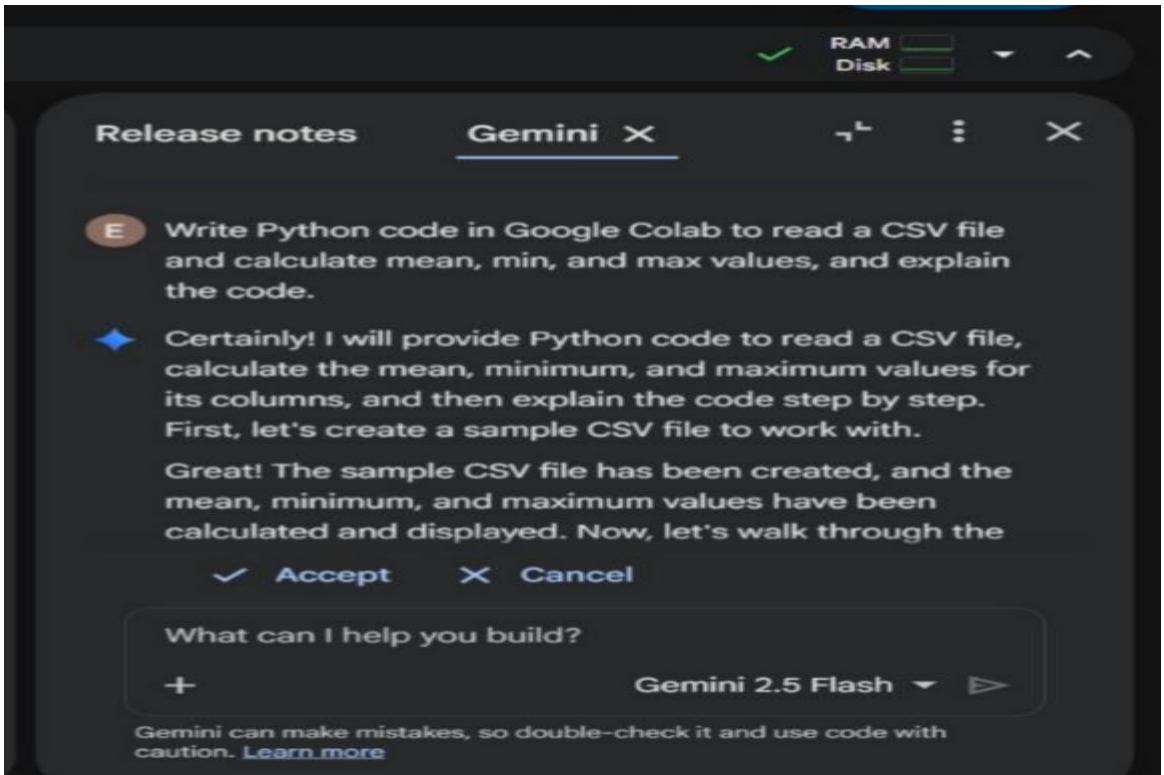
EXPLANATION:

- The program creates a new text file using write mode.
- Sample text is written into the file.
- The file is then opened in read mode.
- The program reads the content of the file.
- Finally, the file content is displayed on the screen.

TASK-03

CSV Data Analysis

PROMPT:



CODE:

The screenshot shows a Google Colab notebook titled "week-2.3.ipynb". The code cell contains the following Python code:

```
import pandas as pd
import numpy as np

# Create a sample DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [24, 27, 22, 32, 29],
    'Score': [85.5, 90.1, 78.9, 92.3, 88.7],
    'Grade': ['A', 'A', 'B', 'A', 'B']
}
df_sample = pd.DataFrame(data)

# Save the DataFrame to a CSV file
csv_filename = 'sample_data.csv'
df_sample.to_csv(csv_filename, index=False)

print(f"'{csv_filename}' created successfully with the following content:")
print(df_sample.to_string())
```

The output of the code cell shows the contents of the CSV file:

```
'sample_data.csv' created successfully with the following content:
   Name  Age  Score Grade
0  Alice   24    85.5     A
1    Bob   27    90.1     A
2  Charlie   22    78.9     B
3   David   32    92.3     A
4    Eve   29    88.7     B
```

To the right of the code cell, the Gemini AI interface is visible again, showing the same conversation and code generation process.

The screenshot shows a Google Colab notebook titled "week-2.3.ipynb". The code cell contains Python code to read a CSV file named "sample_data.csv" and calculate mean, minimum, and maximum values for numerical columns. The output of the code is displayed in the terminal.

```

[6]: csv_file_path = 'sample_data.csv'
try:
    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(csv_file_path)
    print(f"\nSuccessfully read '{csv_file_path}'. First 5 rows:")
    display(df.head())
    # Calculate mean, min, and max for numerical columns
    # Select only numerical columns for these operations
    numerical_df = df.select_dtypes(include=np.number)
    if not numerical_df.empty:
        mean_values = numerical_df.mean()
        min_values = numerical_df.min()
        max_values = numerical_df.max()
        print("\nMean values:")
        display(mean_values)
        print("\nMinimum values:")
        display(min_values)
        print("\nMaximum values:")
        display(max_values)
    else:
        print("No numerical columns found to calculate statistics.")

except FileNotFoundError:
    print(f"Error: The file '{csv_file_path}' was not found. Please ensure it exists")
except Exception as e:
    print(f>An error occurred: {e}")

```

The right side of the interface shows a "Gemini" AI panel with a task card asking to write Python code to read a CSV file and calculate statistics. The AI has responded with the provided code.

OUTPUT:

The screenshot shows the same Google Colab notebook. The code cell has been run, and the resulting DataFrame is displayed in the terminal. The AI panel on the right has generated statistics for the dataset.

index	Name	Age	Score
0	Alice	24	85.5 A
1	Bob	27	90.1 A
2	Charlie	22	78.9 B
3	David	32	92.3 A
4	Eve	29	88.7 B

Show 25 ⏮ per page

Like what you see? Visit the [data table notebook](#) to learn more about interactive tables.

Mean values:
Age 26.8
Score 87.1

dtype: float64

Minimum values:
Age 22.0
Score 78.9

The AI panel on the right has accepted the task and provided the generated code and statistics.

EXPLANATION:

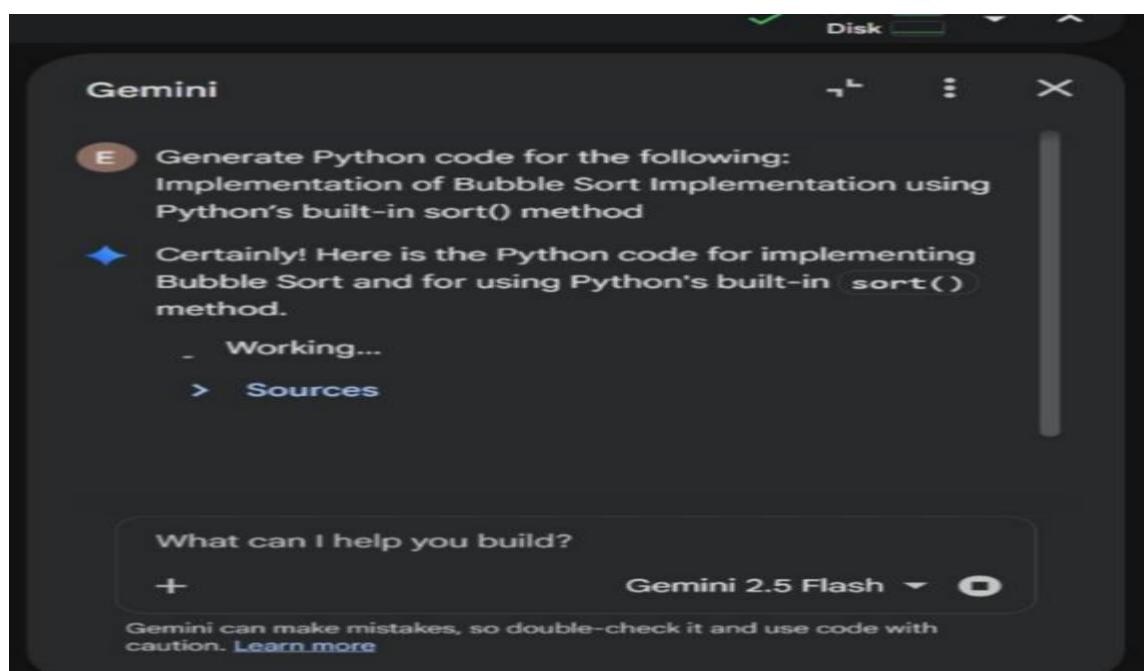
- The program imports the pandas library to work with CSV data.
- The CSV file is uploaded and read into a DataFrame.
- The dataset is displayed to understand its structure.

- The program calculates the mean of numeric columns.
- It finds the minimum value in each numeric column.
- It finds the maximum value in each numeric column.
- The results are displayed as output.

TASK-04

Sorting Lists – Manual vs Built-in

PROMPT:



CODE:

```
def bubble_sort(arr):
    n = len(arr)
    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 of Bubble Sort
my_list = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", my_list)
sorted_list = bubble_sort(list(my_list)) # Pass a copy to preserve original if needed
print("Sorted list using Bubble Sort:", sorted_list)
```

OUTPUT:

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

EXPLANATION:

1. Bubble Sort

- Bubble sort repeatedly compares adjacent elements.
- If the elements are in the wrong order, they are swapped.
- This process continues until the list is completely sorted.
- It is easy to understand but inefficient for large datasets.

2. Python Built-in sort()

- The `sort()` method sorts the list directly using an optimized algorithm.
- It is faster and more efficient than bubble sort.
- It requires less code and is suitable for large datasets.

Comparison

- Bubble sort has higher time complexity and is slower.
- Python's `sort()` is optimized and much faster.
- Bubble sort is mainly used for learning, while `sort()` is used in real applications.