

ASSIGNMENT-2.3

Name: T. Swetha

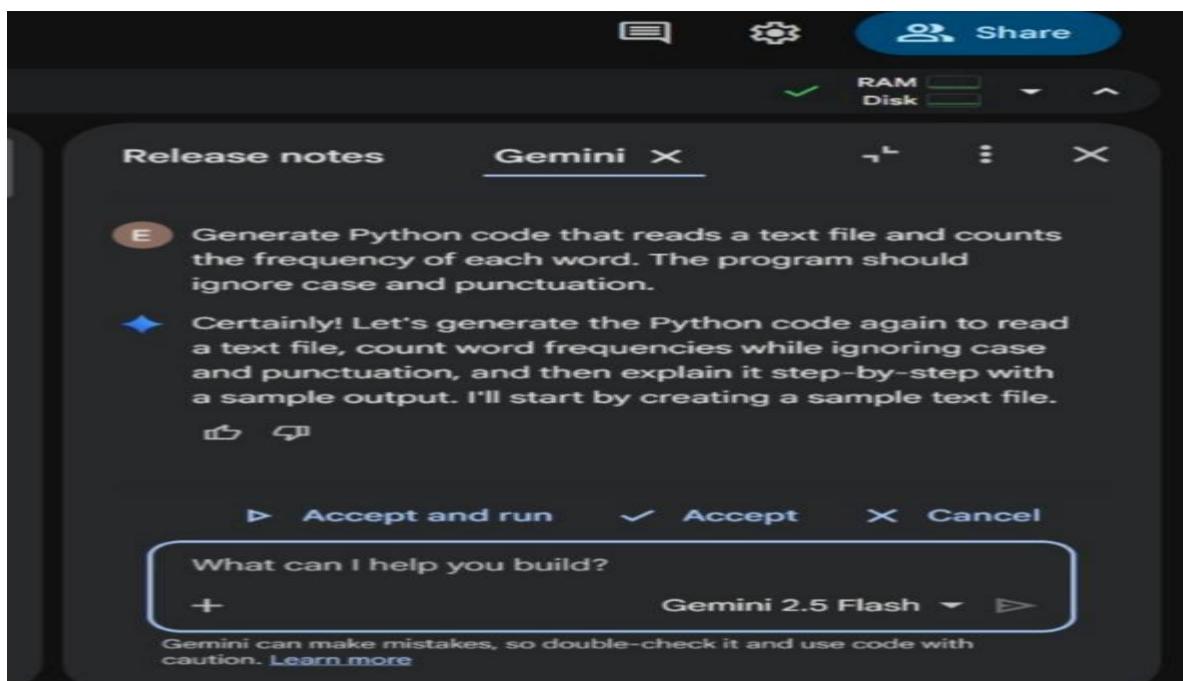
Ht.no:2303A51317

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 is a cell containing Python code to create a sample text file and another cell with a partially completed function to count word frequencies. A Gemini AI sidebar is open, providing step-by-step instructions to generate the code.

```
sample_text = """This is a sample text file. It contains some words, and it has punctuation like commas, periods, and question marks. I'll read this file and count the frequency of each word, ignoring case and punctuation."""
with open('sample.txt', 'w') as f:
    f.write(sample_text)

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

```
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()
```

Release notes 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

The screenshot shows the same Google Colab notebook. The code editor now contains the completed function to count word frequencies. The Gemini AI sidebar continues to provide step-by-step instructions and explanations.

```
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'[^\w\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}')
```

Release notes 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

Variables Terminal ✓ 09:36 Python 3

OUTPUT:

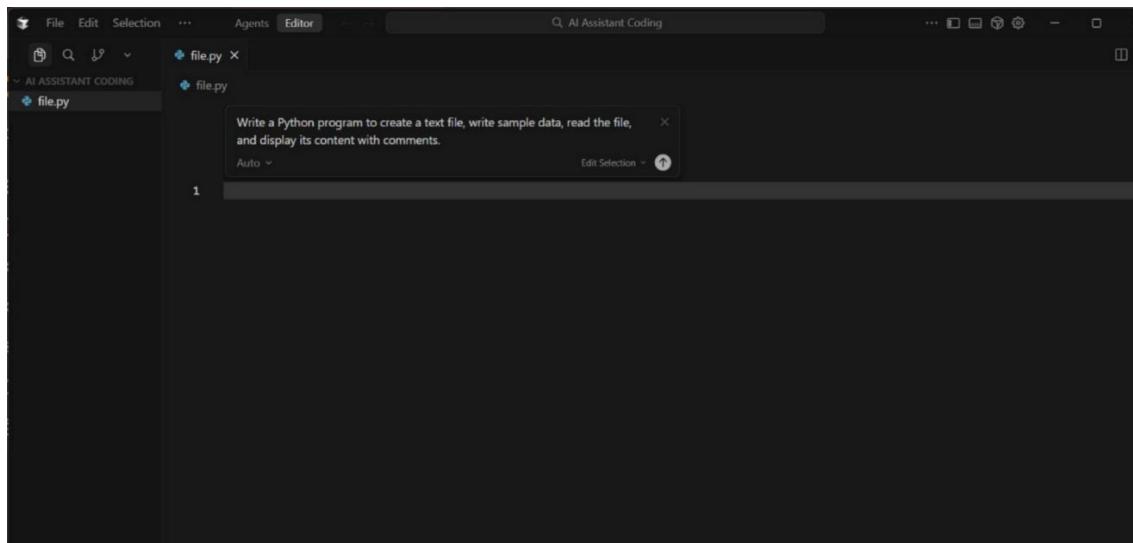
```
...  
Word Frequencies:  
'this': 2  
'is': 2  
'a': 2  
'sample': 1  
'text': 1  
'file': 1  
'it': 2  
'contains': 1  
'some': 1  
'words': 2  
'and': 2  
'has': 1  
'punctuation': 1  
'like': 1  
'commas': 1  
'periods': 1  
'lets': 1  
'count': 1  
'good': 1  
'example': 1
```

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:

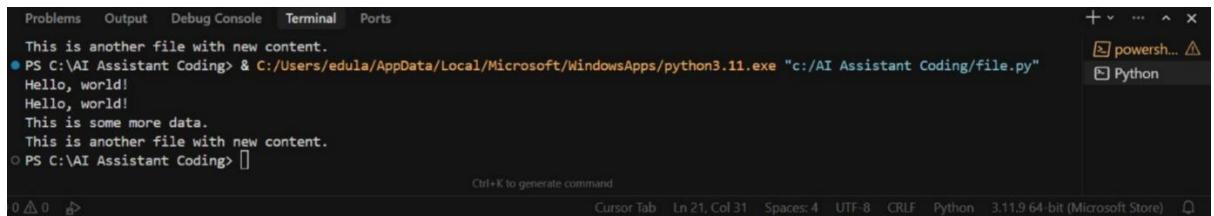
A screenshot of a code editor window titled "AI Assistant Coding". The menu bar includes File, Edit, Selection, Agents, and Editor. The title bar shows "file.py", "Extension: Python", and "sample.txt". The code editor displays a Python script named "file.py" with the following content:

```
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 the code editor 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:



```
Problems Output Debug Console Terminal Ports
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> []
Ctrl+K to generate command
Cursor Tab Ln 21, Col 31 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store) Q
```

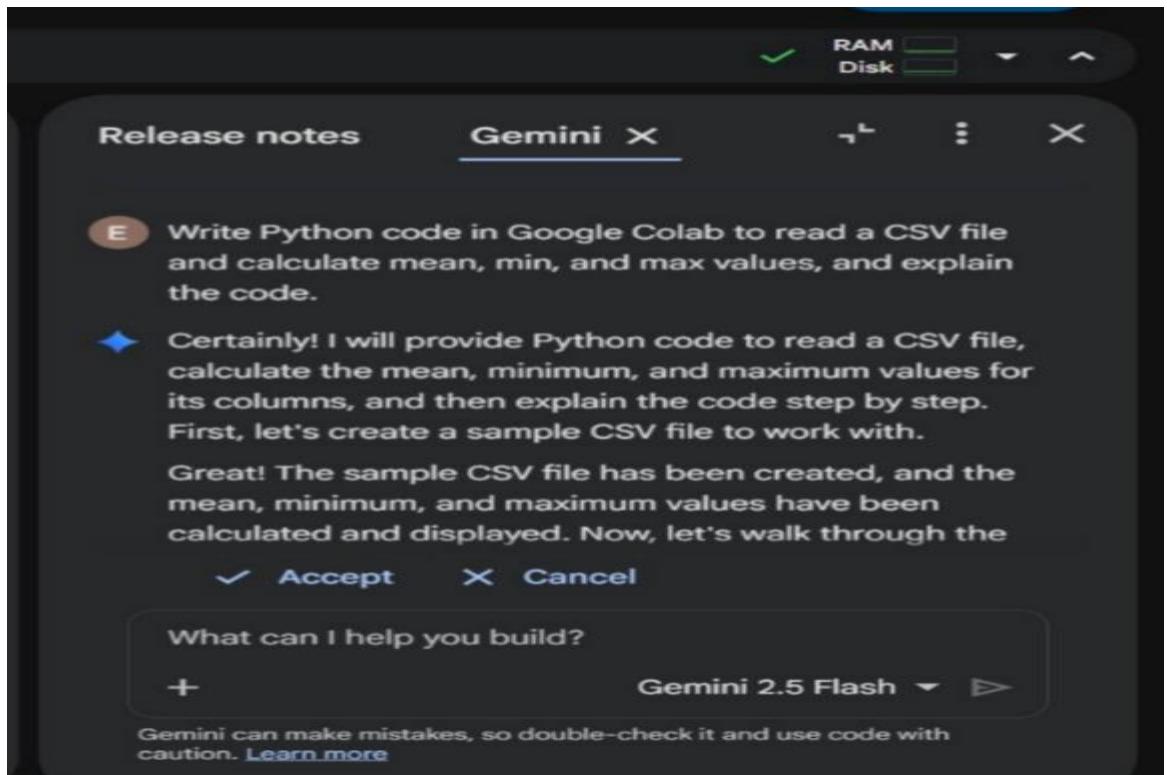
EXPLANATION:

- 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:

Google Colab - Yahoo India Search | week-2.3.ipynb - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[5] ✓ 0s
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())

```

... '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

Variables Terminal

Release notes Gemini X

E Write Python code in Google Colab to read a CSV file and calculate mean, min, and max values, and explain the code.

◆ Certainly! I will provide Python code to read a CSV file, calculate the mean, minimum, and maximum values for its columns, and then explain the code step by step. First, let's create a sample CSV file to work with.

Great! The sample CSV file has been created, and the mean, minimum, and maximum values have been calculated and displayed. Now, let's walk through the code.

✓ 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

✓ 10:24 Python 3

Google Colab - Yahoo India Search | week-2.3.ipynb - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[6] ✓ 0s
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}")

```

Variables Terminal

Release notes Gemini X

E Write Python code in Google Colab to read a CSV file and calculate mean, min, and max values, and explain the code.

◆ Certainly! I will provide Python code to read a CSV file, calculate the mean, minimum, and maximum values for its columns, and then explain the code step by step. First, let's create a sample CSV file to work with.

Great! The sample CSV file has been created, and the mean, minimum, and maximum values have been calculated and displayed. Now, let's walk through the code.

✓ 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

✓ 10:24 Python 3

OUTPUT:

The screenshot shows a Google Colab notebook titled "week-2.3.ipynb". On the left, there's an interactive table with columns: index, Name, Age, and Score. The data is as follows:

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

Below the table, the code used to calculate mean, minimum, and maximum values is shown:

```
Mean values:  
    0  
    Age 26.8  
    Score 87.1  
dtype: float64  
Minimum values:  
    0  
    Age 22.0  
    Score 78.9
```

On the right, a Gemini AI interface is open, showing a release note and a prompt for calculating statistics from a CSV file.

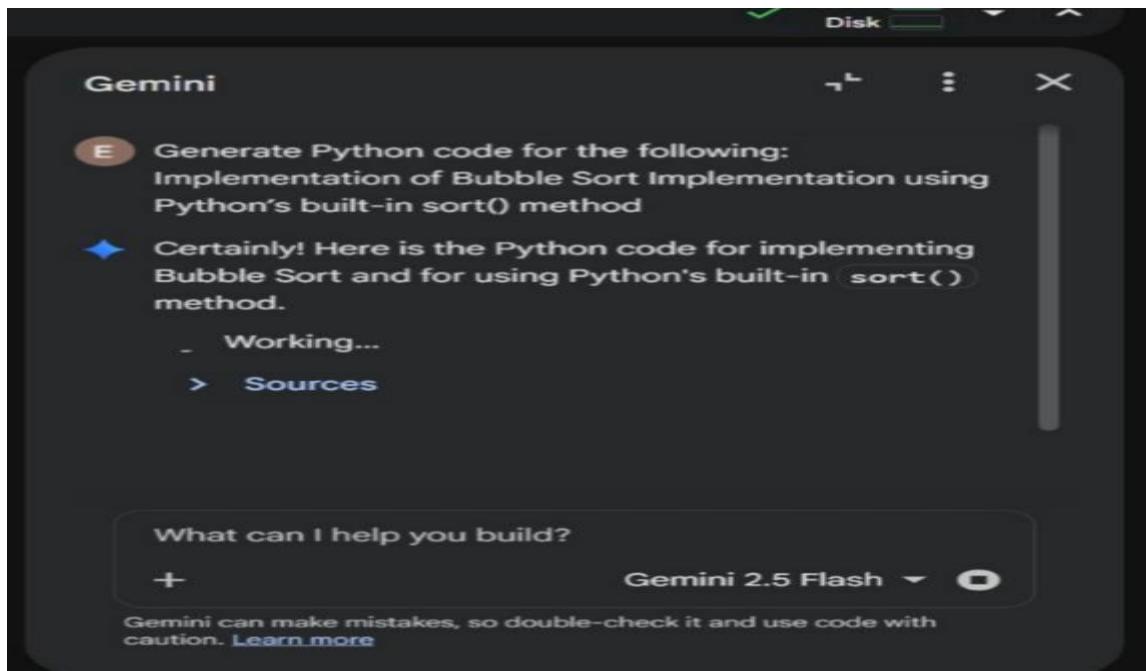
EXPLANATION:

- 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:

The screenshot shows a Google Colab notebook titled "Untitled15.ipynb". On the left, there's a code editor with the following Python code for Bubble Sort:

```
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)
```

On the right, the Gemini AI interface is visible, showing the same query and the generated code. The output of the code execution in Colab shows the original list [64, 34, 25, 12, 22, 11, 90] and the sorted list using Bubble Sort [11, 12, 22, 25, 34, 64, 90]. The status bar at the bottom right shows "10:36" and "Python 3".

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.