

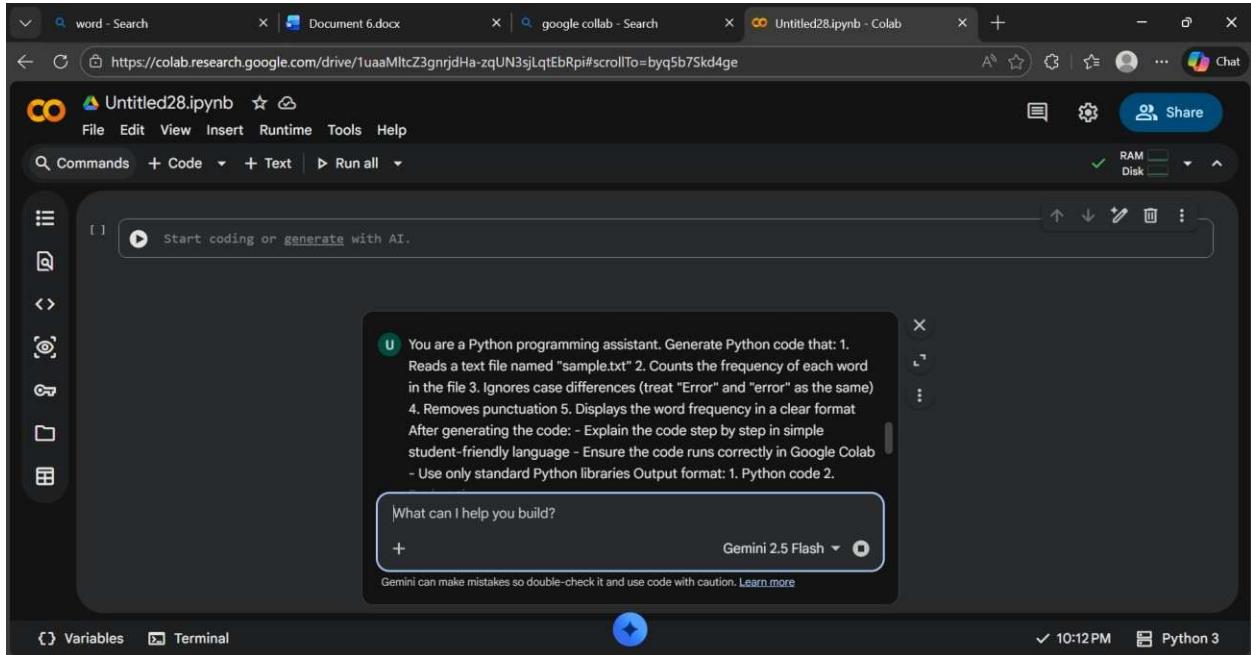
Assignment 2.3 AI ASSISTED CODING

HTNO:2303A510C1

BTNO:06

TASK 1:

Prompt:



Code:

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell contains the following Python script:

```
sample_text = """This is a sample text file. \nIt contains some words, some punctuation like commas, periods, and exclamation marks! \nWord with open('sample.txt', 'w') as f:  
    f.write(sample_text)  
  
print("sample.txt created successfully!")  
... sample.txt created successfully!
```

Below the code cell, a note reads: "Now, here's the Python code to read the `sample.txt` file, count the frequency of each word (ignoring case and punctuation), and display the results:"

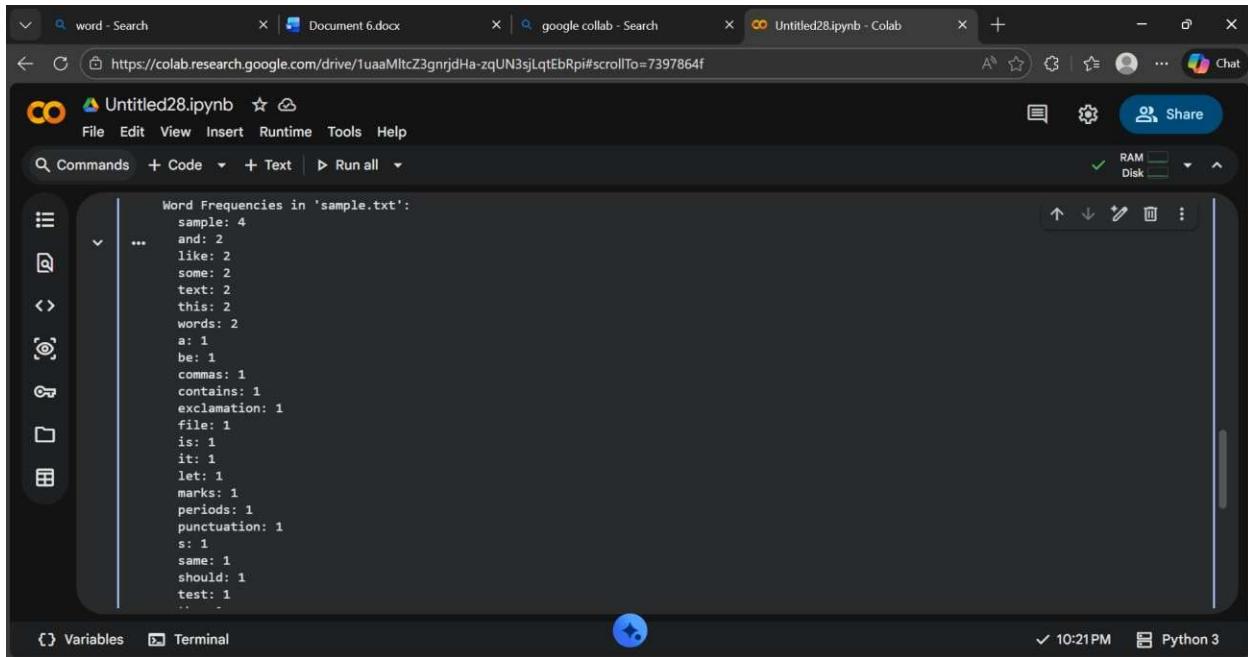
The sidebar on the left shows two files: "os" and "sample.txt". The status bar at the bottom right indicates "10:21PM" and "Python 3".

The screenshot shows the same Google Colab notebook "Untitled28.ipynb". The code cell now contains a more complex Python script:

```
import re  
from collections import Counter  
  
def count_word_frequency(filepath):  
    """  
    Counts the frequency of each word in a text file,  
    ignoring case and punctuation.  
    """  
  
    word_list = []  
    try:  
        with open(filepath, 'r', encoding='utf-8') as file:  
            text = file.read()  
            # Convert to lowercase and find all words (alphanumeric sequences)  
            # \b ensures whole words, \w+ matches one or more word characters  
            word_list = re.findall(r'\b\w+\b', text.lower())  
    except FileNotFoundError:  
        print(f"Error: The file '{filepath}' was not found.")  
        return {}  
    except Exception as e:  
        print(f"An error occurred: {e}")  
        return {}
```

The sidebar on the left shows the same files: "os" and "sample.txt". The status bar at the bottom right indicates "10:21PM" and "Python 3".

Output:



A screenshot of a Google Colab notebook titled "Untitled28.ipynb". The notebook interface includes a top bar with tabs for "word - Search", "Document 6.docx", "google colab - Search", and the current notebook. Below the tabs is a toolbar with icons for File, Edit, View, Insert, Runtime, Tools, Help, Commands, Code, Text, Run all, Share, and RAM/Disk status. The main workspace displays the output of a code cell: "Word Frequencies in 'sample.txt':". The output is a dictionary-like list of words and their frequencies:

```
sample: 4
and: 2
like: 2
some: 2
text: 2
this: 2
words: 2
a: 1
be: 1
commas: 1
contains: 1
exclamation: 1
file: 1
is: 1
it: 1
let: 1
marks: 1
periods: 1
punctuation: 1
s: 1
same: 1
should: 1
test: 1
```

At the bottom of the workspace are buttons for Variables, Terminal, and Python 3. The status bar shows the time as 10:21PM.

Explanation:

The program reads text from a file and converts all words to lowercase to avoid case differences.

It removes punctuation and splits the text into words, then counts how many times each word appears.

Finally, it prints each word along with its frequency in the file.

Task 2:

Prompt:



A screenshot of a code editor window titled "file.py". The editor interface includes a top bar with File, Edit, Selection, Agents, Editor, and AI Assistant Coding. The left sidebar shows a file tree with "file.py" selected. The main workspace contains a code editor with the following text:

```
Write a Python program to create a text file, write sample data, read the file, and display its content with comments.
```

The status bar at the bottom shows the file name "file.py" and the line number "1".

Code:

The screenshot shows a Google Colab interface. On the left, a code cell titled "Implementation of Bubble Sort" contains Python code for bubble sort. On the right, a "Gemini" sidebar provides information and prompts related to the code.

```
# Implementation of 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)
```

Gemini

- E Generate Python code for the following:
Implementation of Bubble Sort Implementation using Python's built-in `sort()` method
- ◆ Certainly! Here is the Python code for implementing Bubble Sort and for using Python's built-in `sort()` method.
The code for Bubble Sort successfully sorted [64, 34, 25, 12, 22, 11, 90] into [11, 12, 22, 25, 34, 64, 90].
Compare the performance of Bubble Sort against Python's built-in `sort()`.
- Explain the time complexity of the Bubble Sort algorithm
- Test Bubble Sort with a larger list of numbers

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

Output:

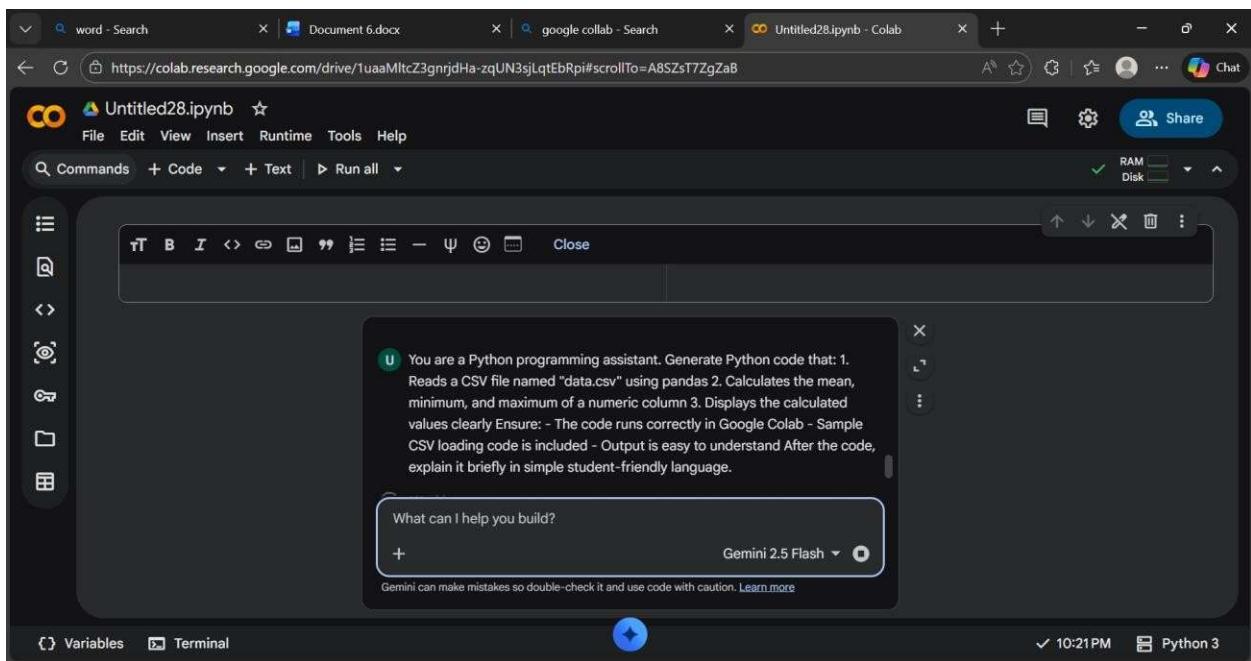
A terminal window shows the following session:

```
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> []
```

Explanation: The program creates a text file and writes sample text into it using file handling methods.

It then reads the file content and displays it on the screen.

Task 3: CSV Data Analysis Prompt:



Code:

A screenshot of a web browser window displaying Google Colab. The title bar shows multiple tabs: 'word - Search', 'Document 6.docx', 'google colab - Search', and 'Untitled28.ipynb - Colab'. The main area is titled 'Untitled28.ipynb' and contains Python code. The code imports pandas, creates a sample DataFrame, saves it to a CSV file, and prints the first few rows. The output shows the DataFrame creation message and the five rows of data.

```
[5]  Os
import pandas as pd

# Create a sample DataFrame
sample_data = {
    'Category': ['A', 'B', 'A', 'C', 'B', 'C', 'A', 'B', 'C', 'A'],
    'Value': [10, 15, 12, 18, 13, 20, 11, 16, 19, 14]
}
sample_df = pd.DataFrame(sample_data)

# Save the DataFrame to a CSV file named 'data.csv'
sample_df.to_csv('data.csv', index=False)

print("Sample 'data.csv' created successfully!")
print(sample_df.head())

... Sample 'data.csv' created successfully!
   Category  Value
0         A     10
1         B     15
2         A     12
3         C     18
4         B     13
```

A screenshot of a web browser window displaying Google Colab. The title bar shows multiple tabs: 'word - Search', 'Document 6.docx', 'google colab - Search', and 'Untitled28.ipynb - Colab'. The main area is titled 'Untitled28.ipynb' and contains Python code. The code reads a CSV file, prints the first five rows, specifies a numeric column for calculations, ensures the column is numeric, drops rows with NaN values, and calculates the mean of the column. The output shows the code execution steps and the final mean value.

```
[6]  Os
import pandas as pd

# 1. Reads a CSV file named 'data.csv' using pandas
try:
    df = pd.read_csv('data.csv')
    print("CSV file 'data.csv' loaded successfully.")
    print("\nFirst 5 rows of the DataFrame:")
    display(df.head())
    
    # Specify the numeric column for calculations
    numeric_column = 'Value'

    if numeric_column in df.columns:
        # Ensure the column is numeric (e.g., if it was read as object due to mixed types)
        df[numeric_column] = pd.to_numeric(df[numeric_column], errors='coerce')

        # Drop rows where the numeric_column became NaN due to coercion errors
        df.dropna(subset=[numeric_column], inplace=True)

        if not df[numeric_column].empty:
            # 2. Calculates the mean, minimum, and maximum of a numeric column
            mean_value = df[numeric_column].mean()
```

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell contains a script for calculating statistics from a CSV file. It includes error handling for empty files and missing columns, and it prints mean, minimum, and maximum values for numeric columns.

```
if not df[numeric_column].empty:
    # 2. Calculates the mean, minimum, and maximum of a numeric column
    mean_value = df[numeric_column].mean()
    min_value = df[numeric_column].min()
    max_value = df[numeric_column].max()

    # 3. Displays the calculated values clearly
    print(f"\nstatistics for column '{numeric_column}':")
    print(f" Mean: {mean_value:.2f}")
    print(f" Minimum: {min_value:.2f}")
    print(f" Maximum: {max_value:.2f}")

else:
    print(f"Error: Column '{numeric_column}' is empty or contains no valid numeric data after cleaning.")

else:
    print(f"Error: Column '{numeric_column}' not found in the CSV file.")

except FileNotFoundError:
    print("Error: 'data.csv' not found. Please make sure the file exists in the current directory.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")


```

Output:

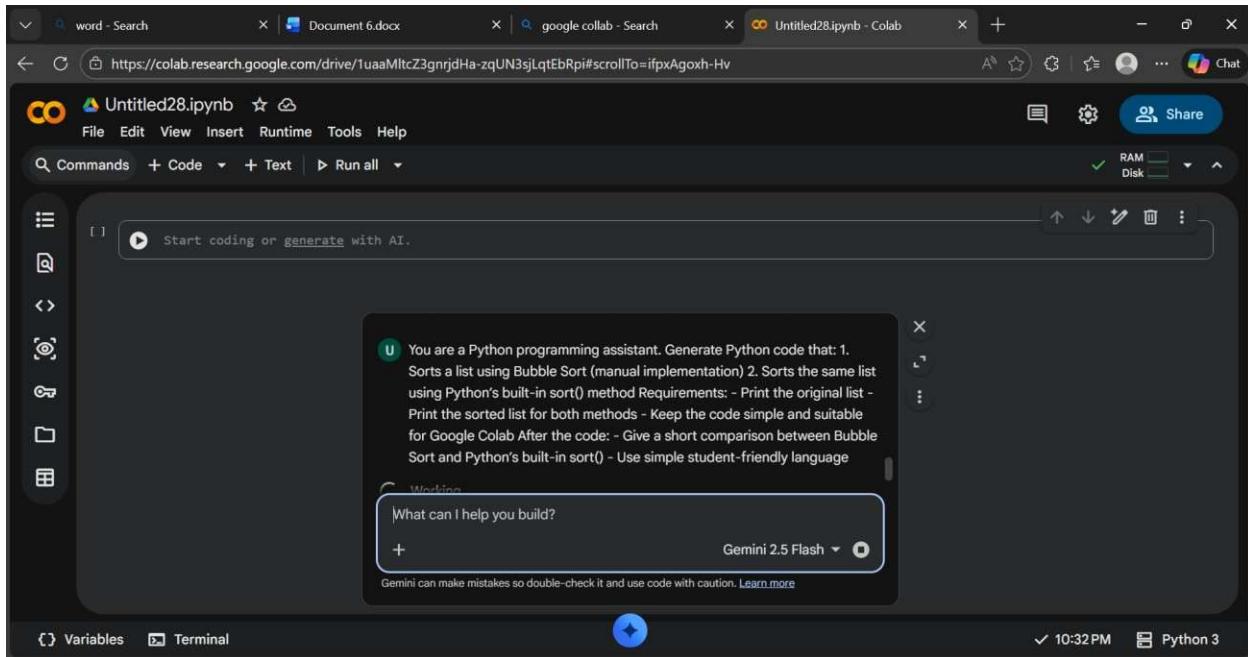
The screenshot shows the output of the script. It prints a message about an unexpected error, followed by a confirmation that the CSV file was loaded successfully. It then displays the first 5 rows of a DataFrame and provides statistics for the 'Value' column.

Category	Value	Mean
0	A	10
1	B	15
2	A	12
3	C	18
4	B	13

Statistics for column 'Value':
Mean: 14.80
Minimum: 10.00
Maximum: 20.00

Explanation: Manual sorting uses loops and comparisons to arrange elements step by step. Built-in sorting is faster and simpler as Python handles the sorting internally.

Task 4: Sorting Lists – Manual vs Built-in Prompt:



Code:

A screenshot of the Google Colab interface. At the top, there are tabs for 'word - Search', 'Document 6.docx', 'google colab - Search', and 'Untitled28.ipynb - Colab'. The main area shows a code cell with the following Python code:

```
# 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]

# Make a copy of the original list for Bubble Sort
bubble_sorted_list = my_list.copy()
bubble_sorted_list = bubble_sort(bubble_sorted_list)
print("\nList after Bubble Sort:", bubble_sorted_list)

# --- 2. Python's built-in sort() method ---
# Make another copy for the built-in sort to keep the original untouched
builtin_sorted_list = my_list.copy()
builtin_sorted_list.sort()
print("List after Python's built-in sort():", builtin_sorted_list)
```

The bottom status bar shows '10:39 PM' and 'Python 3'.

Output:

Explanation: Bubble sort repeatedly compares and swaps elements and is slow for large lists.

Python's built-in `sort()` is faster and more efficient because it uses optimized algorithms.

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell displays three lists:

- Original List: [64, 34, 25, 12, 22, 11, 90, 75, 5]
- List after Bubble Sort: [5, 11, 12, 22, 25, 34, 64, 75, 90]
- List after Python's built-in sort(): [5, 11, 12, 22, 25, 34, 64, 75, 90]

The Colab interface includes a sidebar with "Variables" and "Terminal" tabs, and a status bar at the bottom indicating "10:39 PM" and "Python 3".