

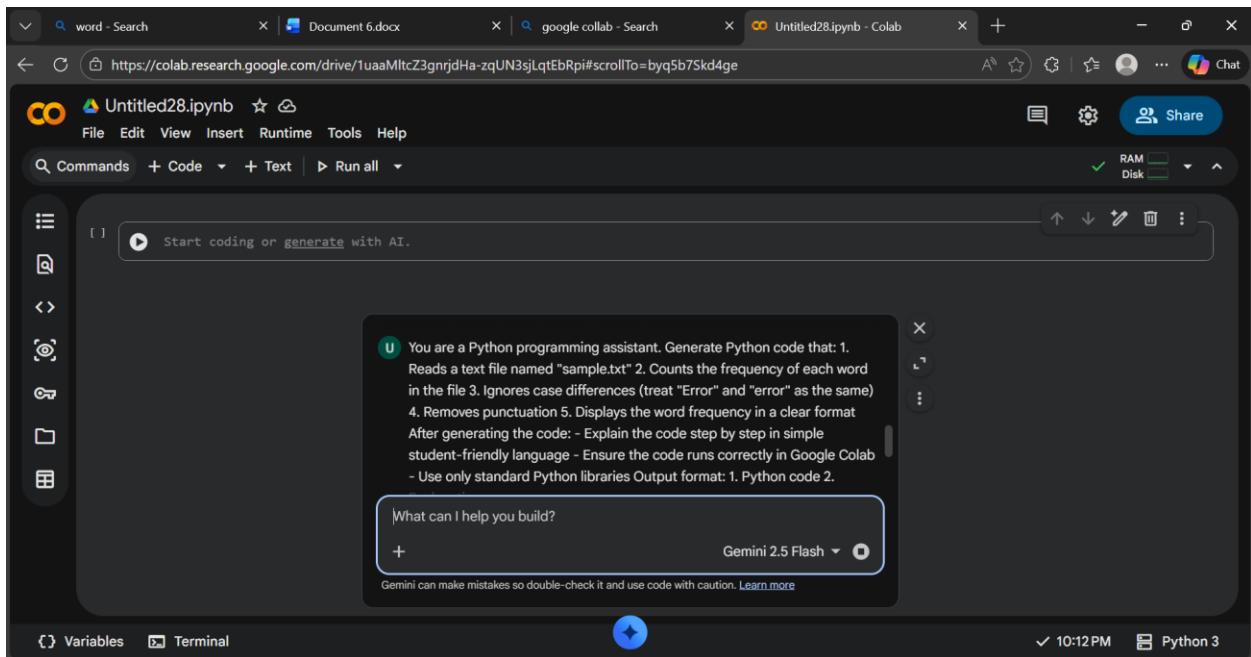
Assignment 2.3 AI ASSISTED CODING

HTNO:2303A51305

BTNO:05

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 text block says:

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 code cell below defines a function `count_word_frequency`:

```
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.  
    """
```

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell contains the following 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 {}
```

Output:

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

```
Word Frequencies in 'sample.txt':  
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
```

The output of the code is displayed in the main pane, showing the word frequencies. The sidebar on the left shows various file and notebook icons. The bottom navigation bar includes "Variables", "Terminal", and "Python 3". The status bar at the bottom right 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 the "AI Assistant Coding" interface. The top menu bar includes "File", "Edit", "Selection", "...", "Agents", and "Editor". The sidebar on the left shows a file tree with two "file.py" entries under "AI ASSISTANT CODING". The main workspace has a text input field containing the following prompt:

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

Code:

The screenshot shows a Google Colab interface with a notebook titled "Implementation of Bubble Sort". The code in the cell is:

```

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)

```

The output of the code is:

```

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

```

A Gemini sidebar is open, asking for Python code to implement Bubble Sort using the built-in `sort()` method. It provides a response with the code and notes on its performance compared to Python's built-in `sort()`.

Output:

The screenshot shows a terminal window in VS Code with the following text:

```

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> []

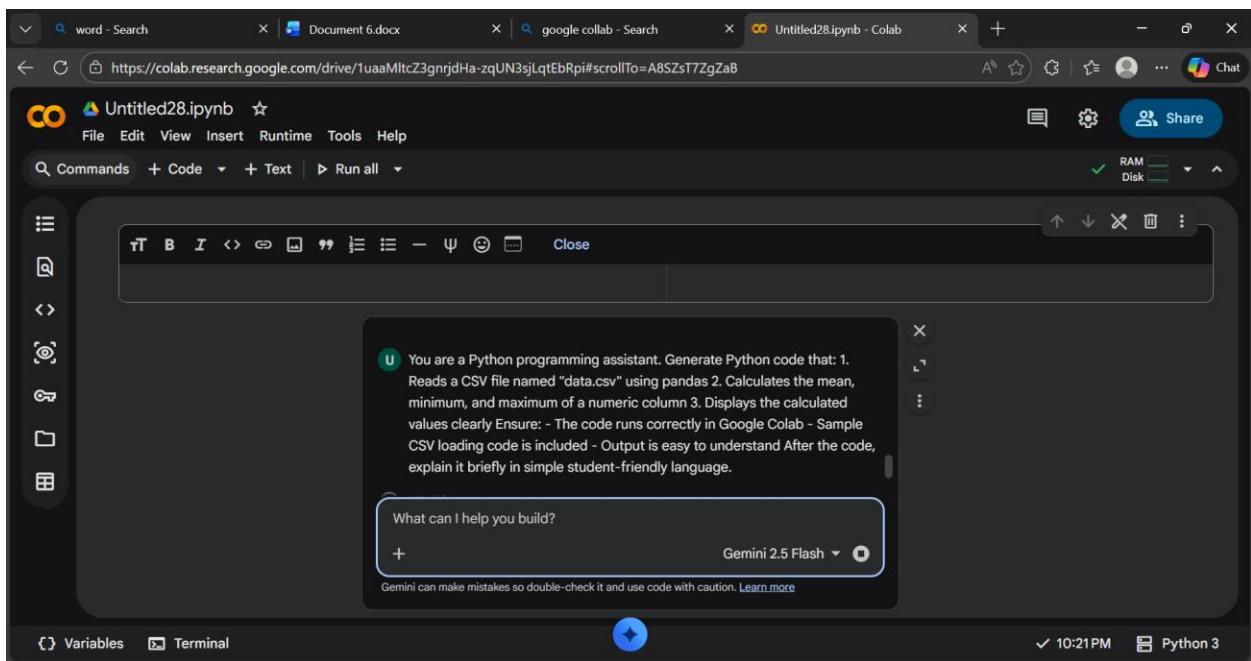
```

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:

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell [5] contains Python code to create a sample DataFrame and save it to a CSV file. The output shows the DataFrame creation and its head.

```
import pandas as pd

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())
```

Output:

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

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell [6] contains Python code to read a CSV file, handle mixed types, drop rows with NaN values, and calculate statistics for the 'Value' column.

```
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 the following Python script:

```
def calculate_statistics(dataframe, numeric_column):
    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 code execution in Google Colab. The output includes:

- A message: "... CSV file 'data.csv' loaded successfully."
- A preview of the first 5 rows of the DataFrame:

Category	Value
0	A
1	B
2	A
3	C
4	B

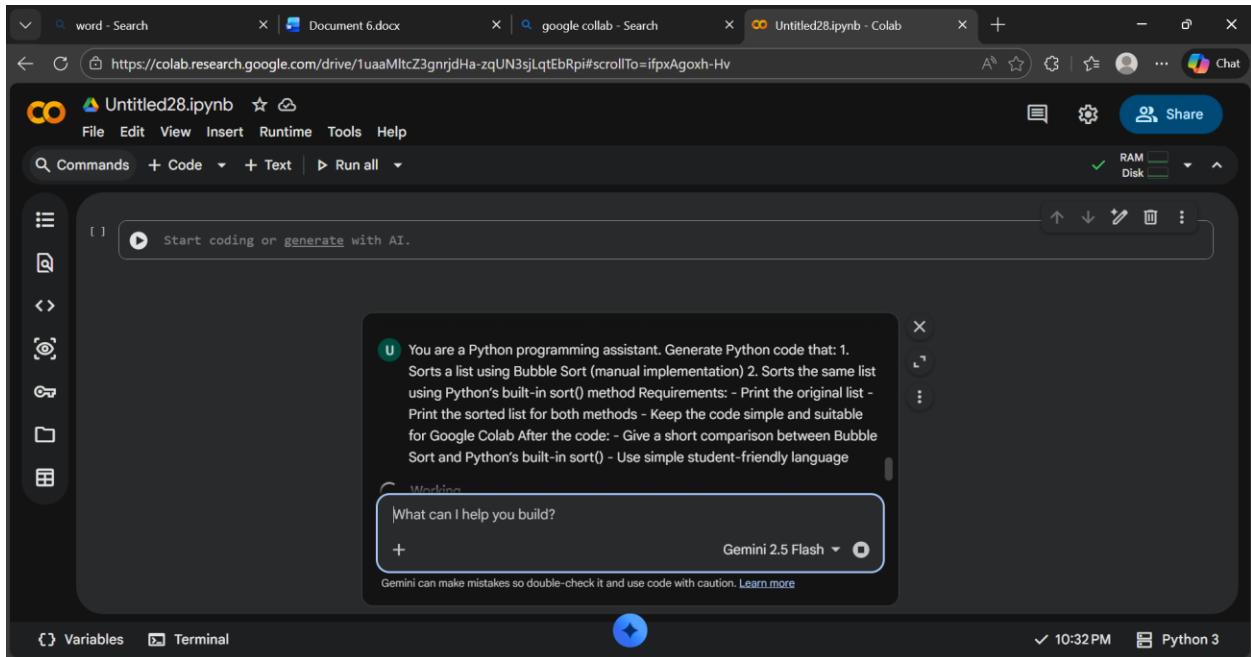
- Statistics for column 'Value':
 - Mean: 14.80
 - Minimum: 10.00
 - Maximum: 20.00

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. The code cell contains 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]
return arr

# 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:

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

```
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 output cell shows the results of the code execution. The status bar at the bottom right indicates the time is 10:39 PM and the Python version is Python 3.

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.