

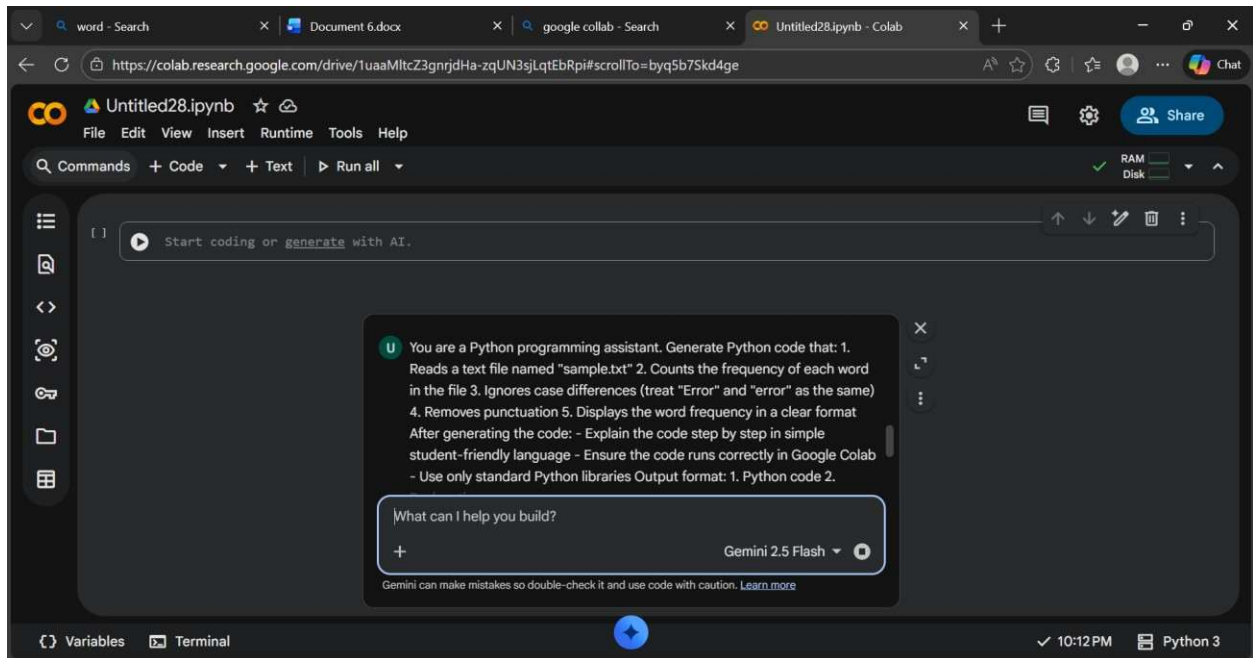
Assignment 2.3 AI ASSISTED CODING

HTNO:2303A510C1

BTNO:06

TASK 1:

Prompt:



Code:

The image displays two sequential screenshots of a Google Colab notebook titled 'Untitled28.ipynb'. The browser's address bar shows the URL: <https://colab.research.google.com/drive/1uaaMltcZ3gnrjdHa-zqUN3sjLqtEbRpi#scrollTo=246734a3>.

Top Screenshot: The notebook is at cell [3]. The code defines a string `sample_text` and writes it to a file named `sample.txt`. The output shows the file was created successfully.

```
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!")
```

Below the code, a text box contains the instruction: "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:".

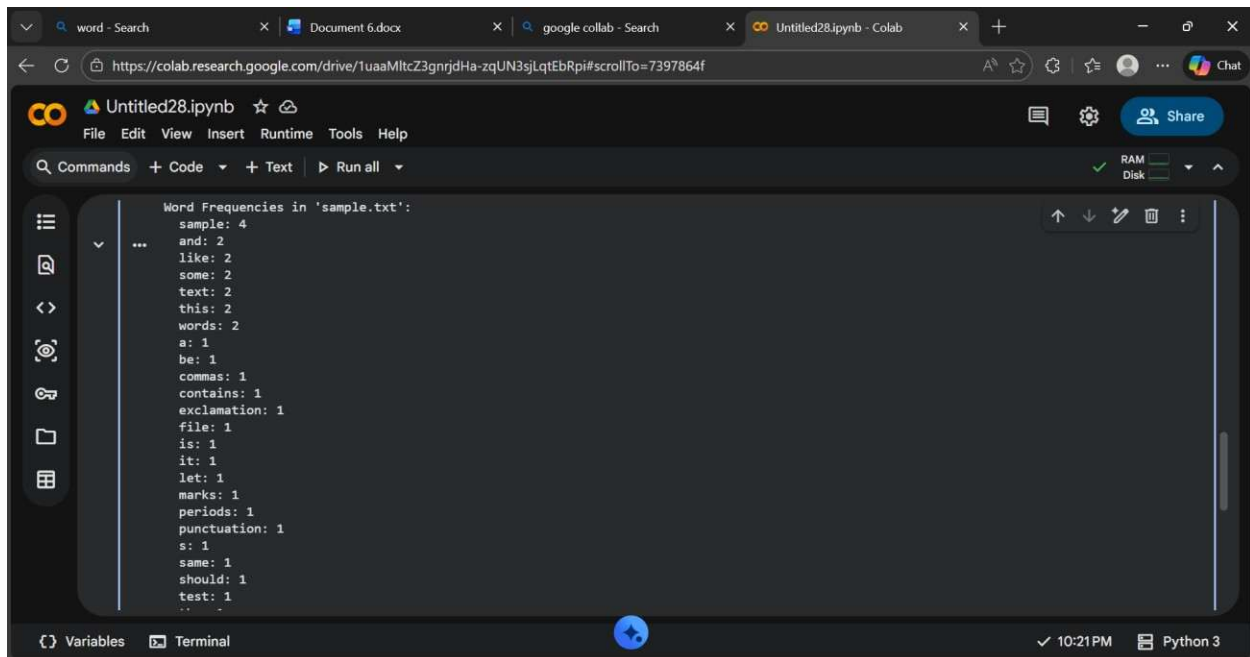
Bottom Screenshot: The notebook is at cell [4]. The code defines a function `count_word_frequency` that reads a file and returns a list of words.

```
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:



The screenshot shows a Google Colab notebook titled 'Untitled28.ipynb'. The code cell contains a Python script that reads a file named 'sample.txt' and prints the frequency of each word. The output is as follows:

```
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
..
```

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:



Code:

The screenshot shows a Google Colab notebook with the title "Implementation of Bubble Sort". The code in the notebook defines a `bubble_sort` function and demonstrates its use on a sample list. The Gemini AI chat interface on the right provides a prompt to generate Python code for Bubble Sort and offers several follow-up suggestions.

```
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)
```

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

Gemini chat interface:

- 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

Output:

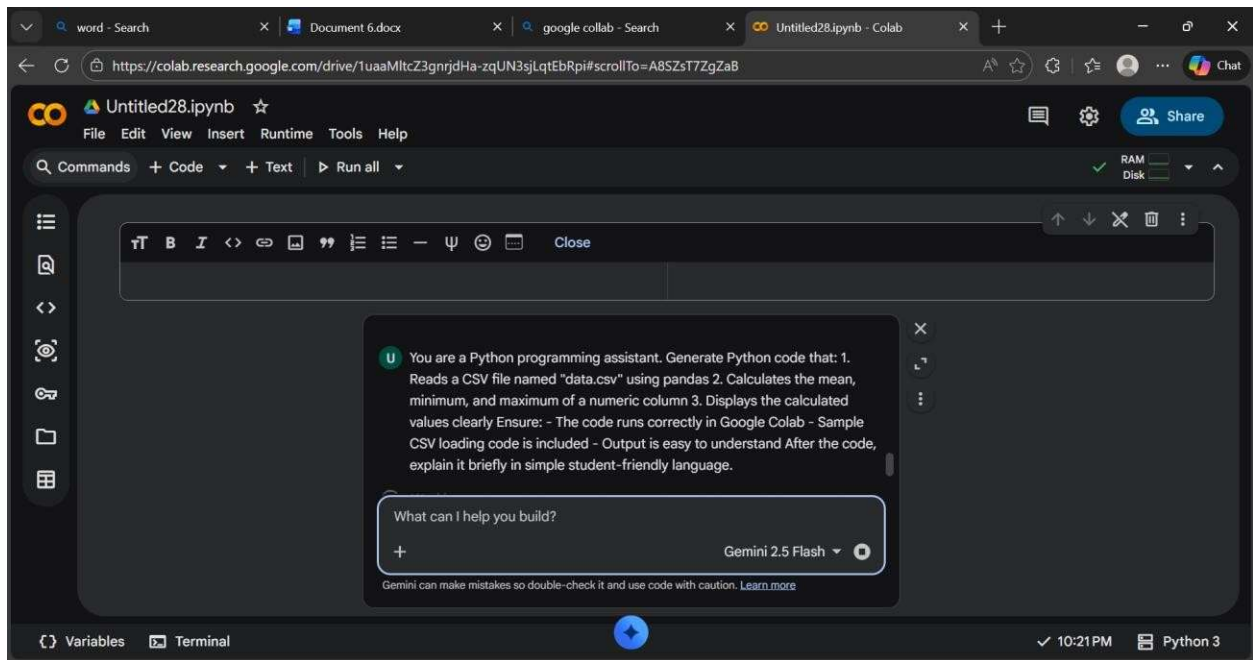
The screenshot shows a Windows terminal window with the following output:

```
PS C:\AI Assistant Coding> C:\Users\edula\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:/AI Assistant Coding/file.py"
This is another file with new content.
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:

word - Search | Document 6.docx | google colab - Search | Untitled28.ipynb - Colab

https://colab.research.google.com/drive/1uaaMltcZ3gnrjdHa-zqUN3sjLqtEbRpi#scrollTo=e2404f88

Untitled28.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

```
[5] 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

Variables Terminal

10:32 PM Python 3

word - Search | Document 6.docx | google colab - Search | Untitled28.ipynb - Colab

https://colab.research.google.com/drive/1uaaMltcZ3gnrjdHa-zqUN3sjLqtEbRpi#scrollTo=e2404f88

Untitled28.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

```
[6] 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()
```

Variables Terminal

10:32 PM Python 3

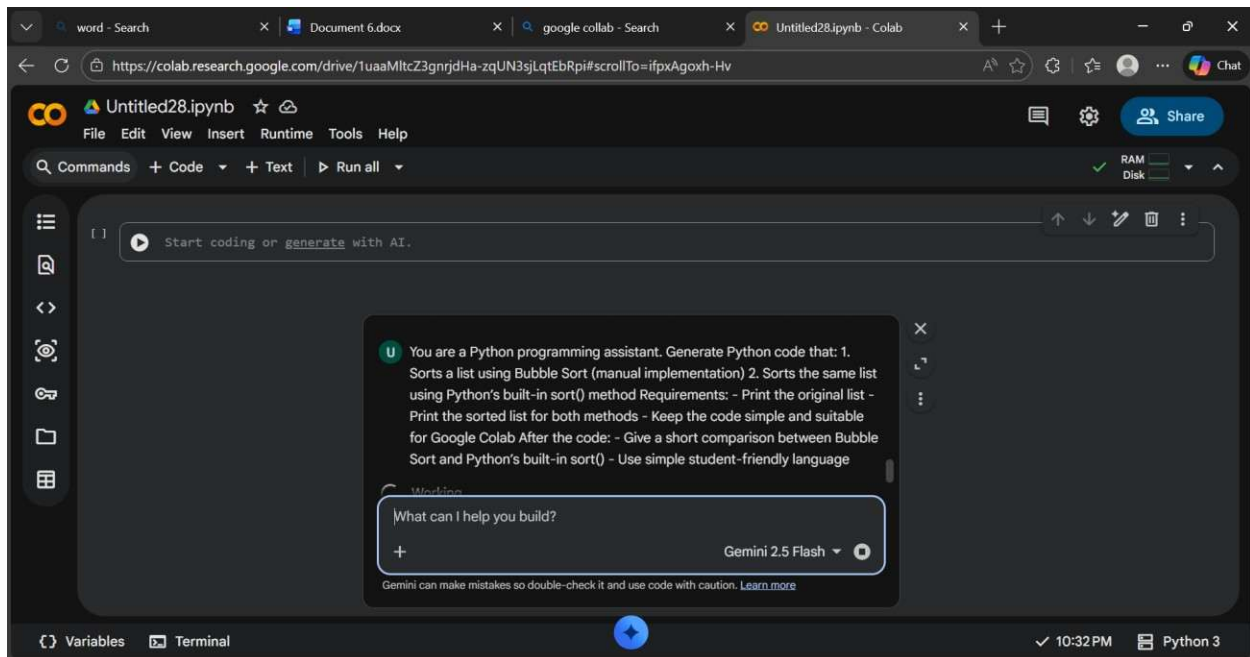
```
def get_stats(df, numeric_column):  
    if not df[numeric_column].empty:  
        # 2. Calculates the mean, minimum, and maximum of a numeric column  
        mean_value = df[numeric_column].mean()  
        min_value = df[numeric_column].min()  
        max_value = df[numeric_column].max()  
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

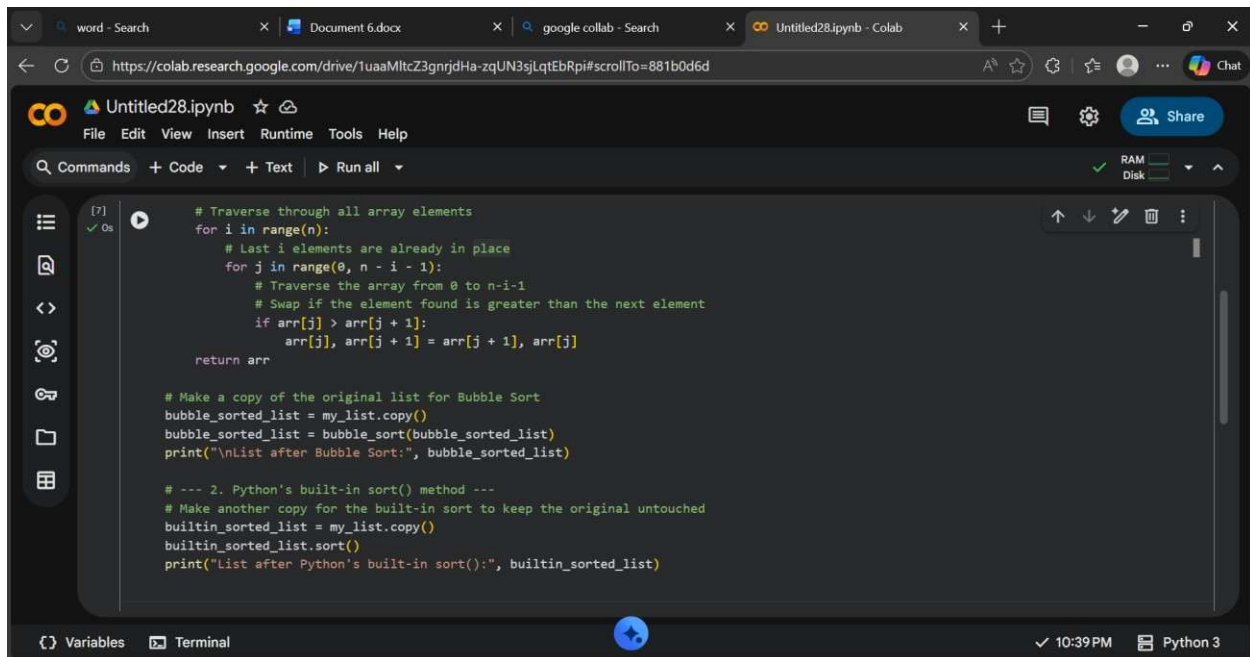
```
print(f"An unexpected error occurred: {e}")  
... CSV file 'data.csv' loaded successfully.  
  
First 5 rows of the DataFrame:  
   Category  Value  
0         A     10  
1         B     15  
2         A     12  
3         C     18  
4         B     13  
  
Statistics for column 'Value':  
Mean: 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:



Output:

Expalantion: 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.

word - Search × Document 6.docx × google collab - Search × Untitled28.ipynb - Colab ×

https://colab.research.google.com/drive/1uaaMltcZ3gnrjdHa-zqUN3sjLqtEbRpi#scrollTo=881b0d6d

Untitled28.ipynb ☆ Share

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

```
... 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]
```

Variables Terminal

✓ 10:39 PM Python 3