

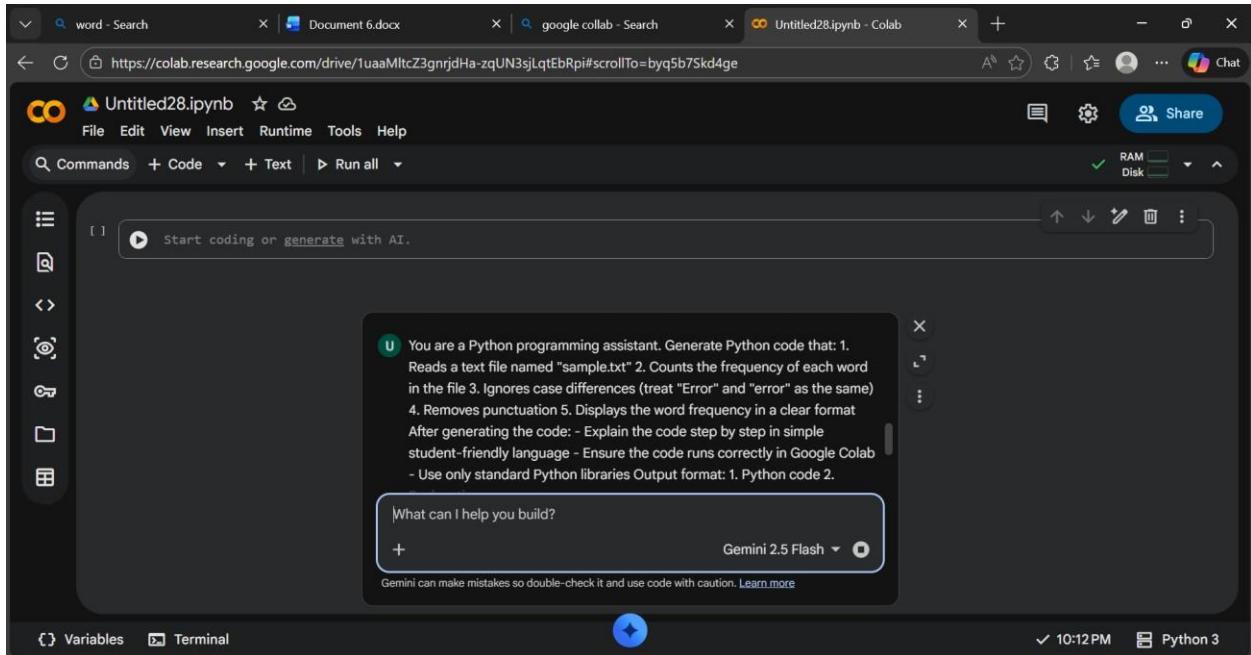
Assignment 2.3 AI ASSISTED CODING

HTNO:2303A51288

BTNO:05 TASK

1:

Prompt:



Code:

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell at the top contains Python code to create a sample text file:

```
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!")
```

The output of this cell is:

```
... 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 code cell below contains the function definition for counting word frequency:

```
[4]  Os  
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 function definition for counting 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.  
    """  
    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 web browser window showing Google Colab. The title bar says "Untitled28.ipynb". The main content area displays a list of words and their frequencies from a file named "sample.txt". The list includes:

```
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 Colab interface shows various tools and status indicators at the top and bottom.

Explanation:

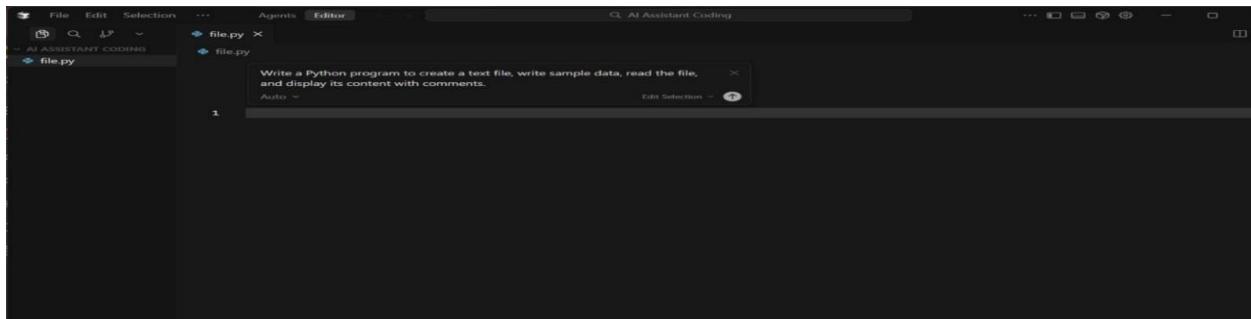
The program reads text from a file and converts all words to lowercase to avoid case differences.

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 interface with a code cell containing Python code for bubble sort. The code defines a function `bubble_sort` that takes an array `arr` and sorts it in place. It includes a comment explaining the algorithm and prints the original and sorted lists. To the right, a Gemini sidebar provides information about the implementation, including a comparison with Python's built-in `sort()` method and links to explain time complexity and test with larger numbers.

```

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:

The terminal window shows a session where sample text is written to a file named `file.py` and then read back. The text includes greetings, a blank line, and a command to generate a command.

```

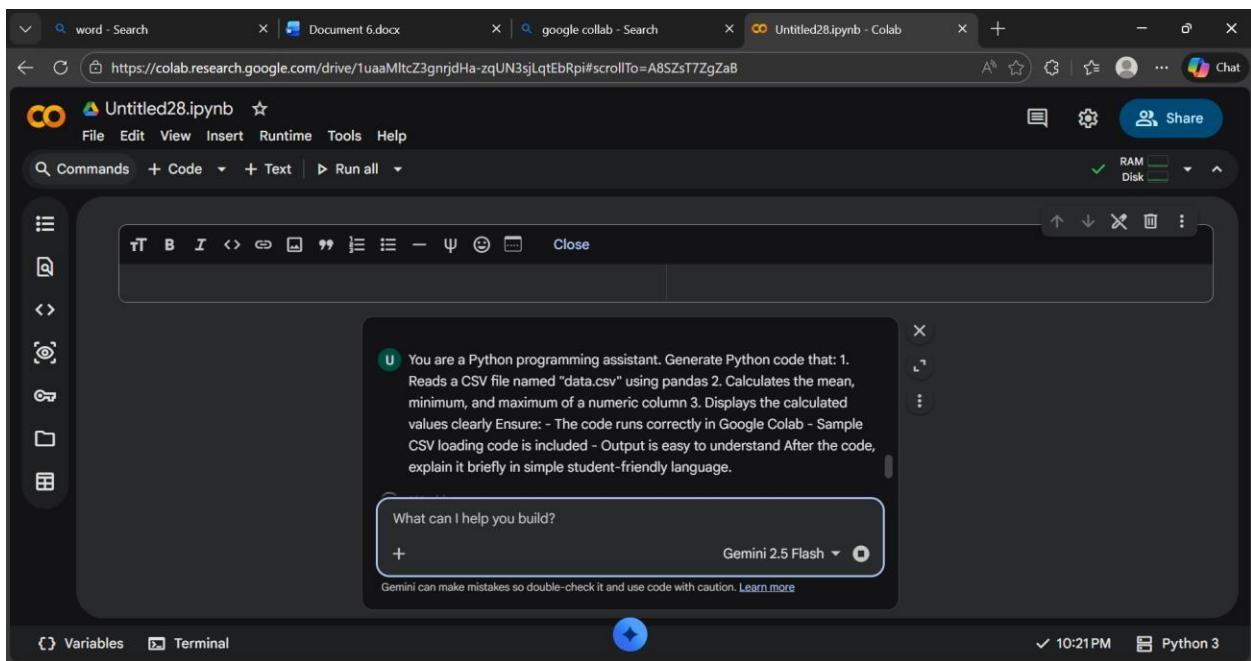
This is another file with new content.
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/file.py"
Hello, world!
Hello, world!
This is some more data.
This is another file with new content.
PS C:\AI Assistant Coding> []

```

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's head and a success message.

```
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())
```

... Sample 'data.csv' created successfully!

Category	Value
A	10
B	15
A	12
C	18
B	13

The screenshot shows a Google Colab notebook titled "Untitled28.ipynb". The code cell [6] contains Python code to read a CSV file, display its first 5 rows, and perform calculations on the 'Value' column. The output shows the DataFrame's head and the calculated mean value.

```
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:

```
# 1. Checks if the column exists
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:

```
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

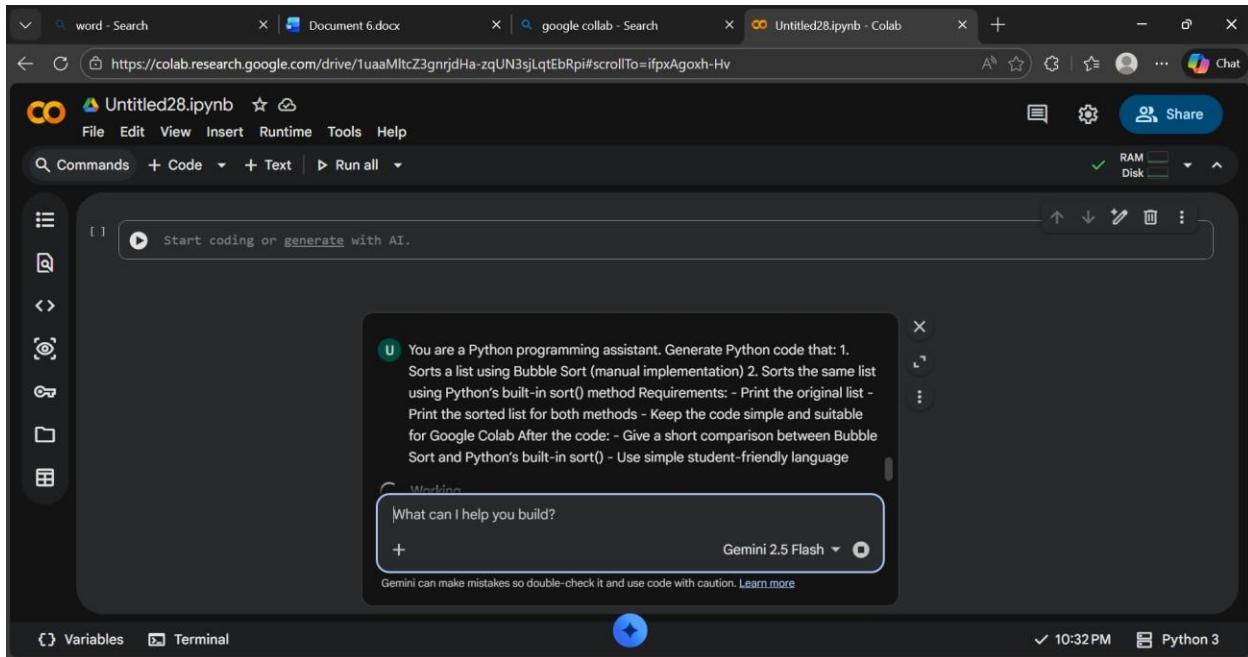
Statistics for column 'Value':
  Mean: 14.80
  Minimum: 10.00
  Maximum: 20.00
```

A preview of the DataFrame is shown, displaying the first five rows with columns "Category" and "Value".

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

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 showing the generated Python code for bubble sort. The code is as follows:

```
# 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)
```

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 browser window with four tabs open:

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

The 'Untitled28.ipynb' tab is active, displaying a dark-themed Jupyter notebook interface. The code cell contains the following Python code and its output:

```
... 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 notebook sidebar includes icons for Variables, Terminal, and Help. The status bar at the bottom right shows the time as 10:39 PM and the Python version as Python 3.