AI Assisted Coding

ASSIGNMENT 2.3

Name: PALAKURTHI LALITH PRAKASH

HT No: 2303A52012

Batch: 31

Task 1: Word Frequency from Text File

❖ Scenario: You are analyzing log files for keyword frequency.

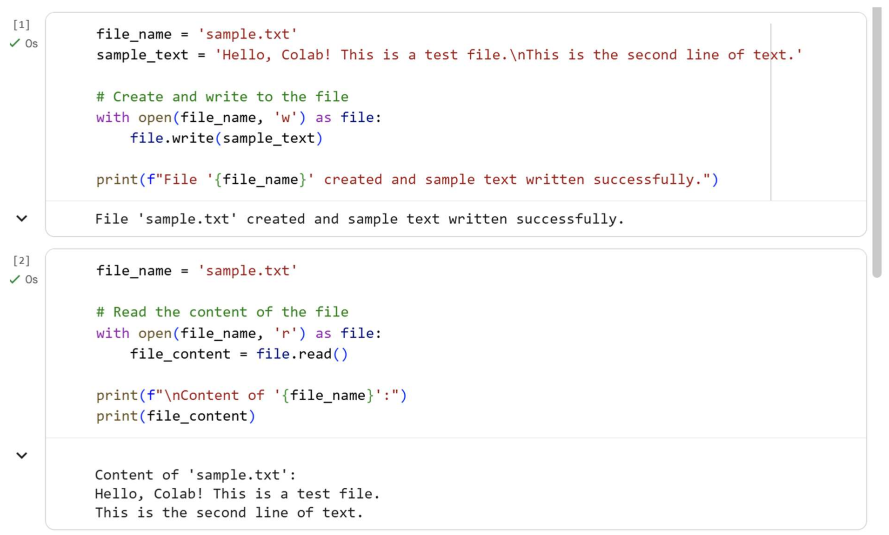
❖ Task: Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.

Prompt:

1. Create a Python program that creates a text file, writes sample text into it, then reads the file and displays its contents.
2. now generate a py code that reads the txt file and prinsta the word frequency

code:

1.

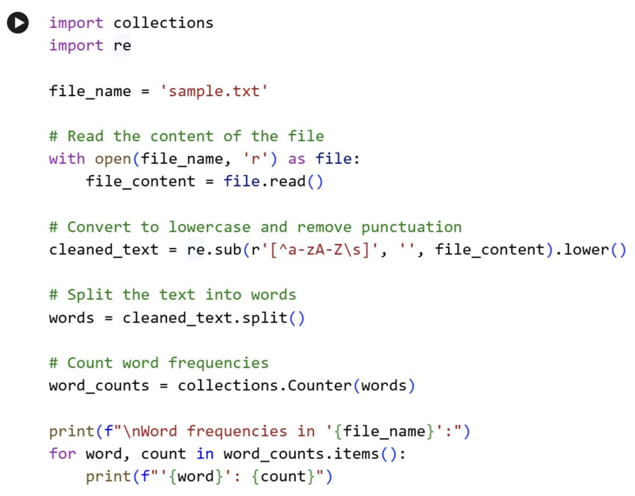


```python
file_name = 'sample.txt'
sample_text = 'Hello, Colab! This is a test file.\nThis is the second line of text.'

# Create and write to the file
with open(file_name, 'w') as file:
    file.write(sample_text)

print(f"File '{file_name}' created and sample text written successfully.")
```

```
File 'sample.txt' created and sample text written successfully.
```

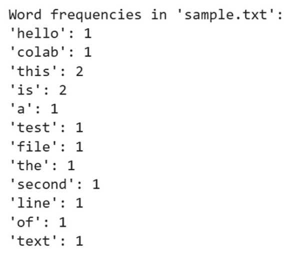
```python
file_name = 'sample.txt'

# Read the content of the file
with open(file_name, 'r') as file:
    file_content = file.read()

print(f"\nContent of '{file_name}':")
print(file_content)
```

```

Content of 'sample.txt':
Hello, Colab! This is a test file.
This is the second line of text.
```



```python
import collections
import re

file_name = 'sample.txt'

# Read the content of the file
with open(file_name, 'r') as file:
    file_content = file.read()

# Convert to lowercase and remove punctuation
cleaned_text = re.sub(r'[^a-zA-Z\s]', '', file_content).lower()

# Split the text into words
words = cleaned_text.split()

# Count word frequencies
word_counts = collections.Counter(words)

print(f"\nWord frequencies in '{file_name}':")
for word, count in word_counts.items():
    print(f"'{word}': {count}")
```

output:



```
Word frequencies in 'sample.txt':
'hello': 1
'colab': 1
'this': 2
'is': 2
'a': 1
'test': 1
'file': 1
'the': 1
'second': 1
'line': 1
'of': 1
'text': 1
```

justification:

the generated code reads the code from the file created earlier generation and thean next using a methods from the collections class it counts all the word frequency.

Task 2: File Operations Using Cursor AI

❖ Scenario:

You are automating basic file operations.

❖ Task:

Use Cursor AI to generate a program that:

➢ Creates a text file

➢ Writes sample text

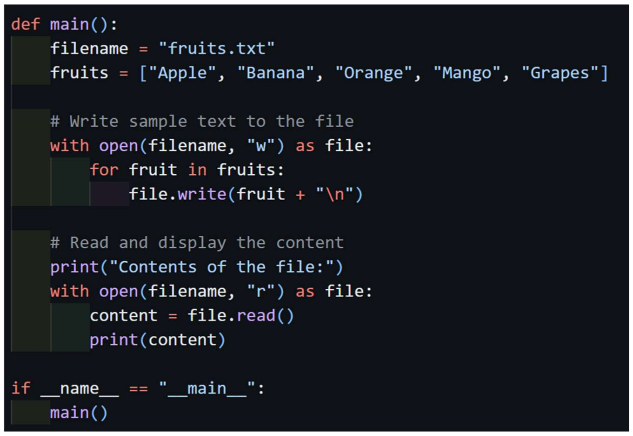
➢ Reads and displays the content

Prompt:

generate a python program, which creates a text file, write sample txt eg.fruits in that, read

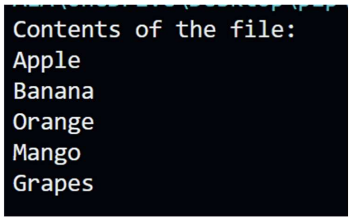
and display that content.

code:



```python
def main():
    filename = "fruits.txt"
    fruits = ["Apple", "Banana", "Orange", "Mango", "Grapes"]

    # Write sample text to the file
    with open(filename, "w") as file:
        for fruit in fruits:
            file.write(fruit + "\n")

    # Read and display the content
    print("Contents of the file:")
    with open(filename, "r") as file:
        content = file.read()
        print(content)

if __name__ == "__main__":
    main()
```

output:



```
Contents of the file:
Apple
Banana
Orange
Mango
Grapes
```

justification:

the program made a file named as fruits.txt and a list with different fruits names and in a loop each fruit name is enter inside the file line byline and then using read() method all the contant is read and printed on to the screen.

Task 3: CSV Data Analysis

❖ Scenario:

You are processing structured data from a CSV file.

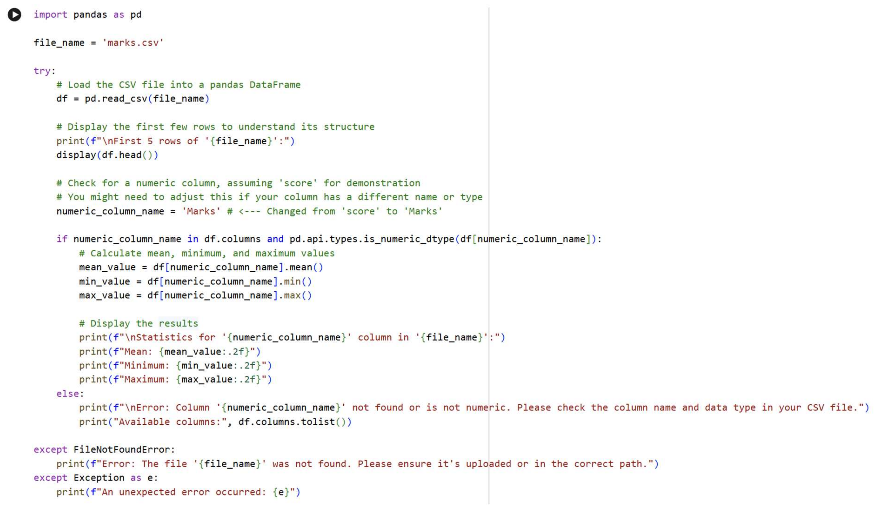
❖ Task:

Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

Prompt:

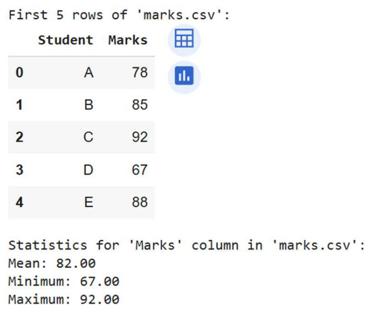
Generate Python code to read marks CSV file and calculate the mean, minimum, and maximum values of a numeric column.

code:



```python
import pandas as pd

file_name = 'marks.csv'

try:
    # Load the CSV file into a pandas DataFrame
    df = pd.read_csv(file_name)

    # Display the first few rows to understand its structure
    print(f"\nFirst 5 rows of '{file_name}':")
    display(df.head())

    # Check for a numeric column, assuming 'score' for demonstration
    # You might need to adjust this if your column has a different name or type
    numeric_column_name = 'Marks' # <--- Changed from 'score' to 'Marks'

    if numeric_column_name in df.columns and pd.api.types.is_numeric_dtype(df[numeric_column_name]):
        # Calculate mean, minimum, and maximum values
        mean_value = df[numeric_column_name].mean()
        min_value = df[numeric_column_name].min()
        max_value = df[numeric_column_name].max()

        # Display the results
        print(f"\nStatistics for '{numeric_column_name}' column in '{file_name}':")
        print(f"Mean: {mean_value:.2f}")
        print(f"Minimum: {min_value:.2f}")
        print(f"Maximum: {max_value:.2f}")
    else:
        print(f"\nError: Column '{numeric_column_name}' not found or is not numeric. Please check the column name and data type in your CSV file.")
        print("Available columns:", df.columns.tolist())

except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found. Please ensure it's uploaded or in the correct path.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

output:



```
First 5 rows of 'marks.csv':
```

	Student	Marks
0	A	78
1	B	85
2	C	92
3	D	67
4	E	88

```
Statistics for 'Marks' column in 'marks.csv':
Mean: 82.00
Minimum: 67.00
Maximum: 92.00
```

justification:

This code reads a CSV file (marks.csv) using pandas and shows the first few rows to understand the data. It then checks whether the Marks column exists and is numeric, and calculates the mean, minimum, and maximum values from it. If the file or column is missing (or invalid), it prints a clear error message instead of crashing.

Task 4: Sorting Lists – Manual vs Built-in

❖ Scenario:

You are reviewing algorithm choices for efficiency.

❖ Task:

Use Gemini to generate:

➢ Bubble sort

➢ Python's built-in sort()

➢ Compare both implementations.

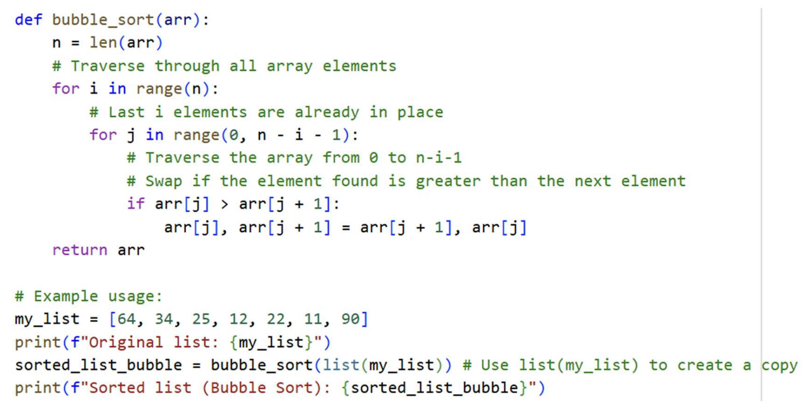
Prompt:

a. generate python code for bubble sort

b. generate python code for built-in sort

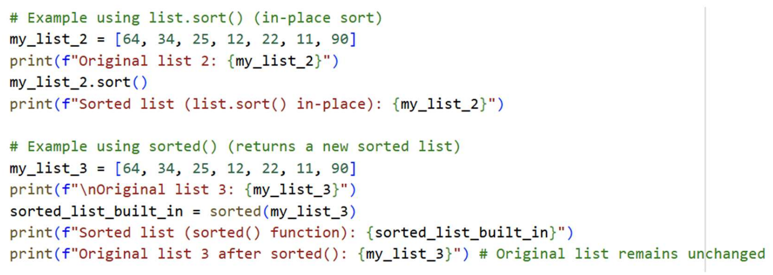
code:

a.



```python
def bubble_sort(arr):
    n = len(arr)
    # 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

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list: {my_list}")
sorted_list_bubble = bubble_sort(list(my_list)) # Use list(my_list) to create a copy
print(f"Sorted list (Bubble Sort): {sorted_list_bubble}")
```

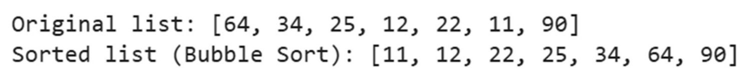
b.



```python
# Example using list.sort() (in-place sort)
my_list_2 = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list 2: {my_list_2}")
my_list_2.sort()
print(f"Sorted list (list.sort() in-place): {my_list_2}")

# Example using sorted() (returns a new sorted list)
my_list_3 = [64, 34, 25, 12, 22, 11, 90]
print(f"\nOriginal list 3: {my_list_3}")
sorted_list_built_in = sorted(my_list_3)
print(f"Sorted list (sorted() function): {sorted_list_built_in}")
print(f"Original list 3 after sorted(): {my_list_3}") # Original list remains unchanged
```

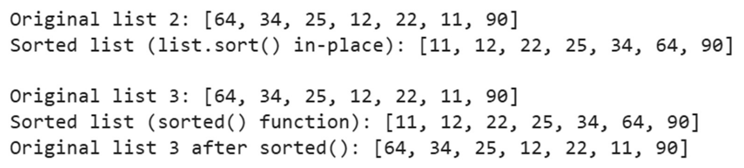
output:

a.



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

b.



```
Original list 2: [64, 34, 25, 12, 22, 11, 90]
Sorted list (list.sort() in-place): [11, 12, 22, 25, 34, 64, 90]

Original list 3: [64, 34, 25, 12, 22, 11, 90]
Sorted list (sorted() function): [11, 12, 22, 25, 34, 64, 90]
Original list 3 after sorted(): [64, 34, 25, 12, 22, 11, 90]
```

justification:

list.sort() sorts the list in place, meaning it changes the original list itself.
sorted() creates a new sorted list and keeps the original list unchanged.
Use sort() when you don’t need the original order, and sorted() when you want to preserve it.