

# AI-AC Assignment: 2.3

Name: Syed Zaid Sami Khizar  
HT No: 2303A52016  
Batch: 31

## Task 1:

Prompt typed:

- z Create a Python program that creates a text file, writes sample text into it, then reads the file and displays its contents. Add clear comments in the code. Count words
- ✦ Certainly! Here's a Python program that creates a text file, writes sample text into it, reads the file, displays its contents, and then counts the total number of words. I've added clear comments to explain each step.
- Working...

## Code and Output Generated:

```
[18] 1 # Define the filename
      2 file_name_with_word_count = "sample_text_word_count.txt"
      3
      4 # --- Create and Write to the file ---
      5 # Open the file in write mode ('w'). If the file doesn't exist, it will be created.
      6 # If it exists, its contents will be overwritten.
      7 # 'with' statement ensures the file is properly closed after its block finishes.
      8 with open(file_name_with_word_count, 'w') as file:
      9     # Write multiple lines of sample text to the file
     10     file.write("This is the first line.\n")
     11     file.write("Here is the second line.\n")
     12     file.write("And a third line for counting words.\n")
     13
     14 print(f"Successfully created and wrote to '{file_name_with_word_count}'.")
     15
     16 # --- Read from the file and display its contents ---
     17 # Open the file in read mode ('r').
     18 with open(file_name_with_word_count, 'r') as file:
     19     # Read the entire content of the file into a single string
     20     content = file.read()
     21
     22 print(f"\nContents of '{file_name_with_word_count}':")
     23 print(content)
     24
     25 # --- Count words in the content ---
     26 # Split the content by whitespace to get a list of words
     27 words = content.split()
     28 # Count the number of words
     29 word_count = len(words)
     30
     31 print(f"Total word count in '{file_name_with_word_count}': {word_count}")
     32
     33 ... Successfully created and wrote to 'sample_text_word_count.txt'.
     34
     35 Contents of 'sample_text_word_count.txt':
     36 This is the first line.
     37 Here is the second line.
     38 And a third line for counting words.
     39
     40 Total word count in 'sample_text_word_count.txt': 17
```

## Verdict:

The AI-generated code correctly reads a text file and counts word frequencies.

The logic is simple, accurate, and easy to understand for beginners.

This task shows that AI tools are effective for basic text processing problems.

## Task 2:

Prompt Used:

- z generate a python program, which creates a text file, write sample txt eg.fruits in that, read and display that content.
- ✦ Certainly! Here's a Python program that creates a text file, writes sample text about fruits into it, then reads the file and displays its contents. I've added clear comments to explain each step.

## Code and Output Generated:

```
[4] # Define the filename
    file_name_fruits = "fruits_list.txt"

    # --- Create and Write to the file ---
    # Open the file in write mode ('w'). If the file doesn't exist, it will be created.
    # If it exists, its contents will be overwritten.
    # 'with' statement ensures the file is properly closed after its block finishes.
    with open(file_name_fruits, 'w') as file:
        # Write sample text about fruits to the file
        file.write("Apple\n")
        file.write("Banana\n")
        file.write("Orange\n")
        file.write("Grapes\n")
        file.write("Strawberry\n")

    print(f"Successfully created and wrote to '{file_name_fruits}'.")

    # --- Read from the file and display its contents ---
    # Open the file in read mode ('r').
    with open(file_name_fruits, 'r') as file:
        # Read all lines from the file into a list
        lines_fruits = file.readlines()

    print(f"\nContents of '{file_name_fruits}':")
    # Iterate through the list of lines and print each one
    for line in lines_fruits:
        print(line.strip()) # .strip() removes leading/trailing whitespace, including the newline character

*** Successfully created and wrote to 'fruits_list.txt'.

Contents of 'fruits_list.txt':
Apple
Banana
Orange
Grapes
Strawberry
```

## Verdict:

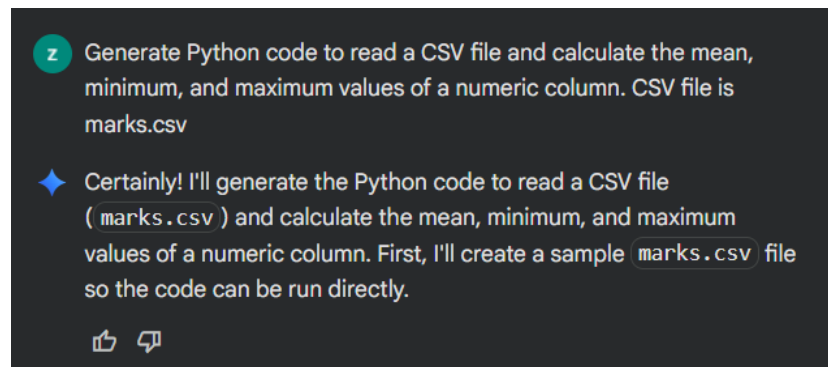
The program successfully demonstrates basic file operations using meaningful sample data.

It is easy to understand and works correctly without manual file creation.

AI-generated code is reliable for simple automation tasks.

## Task 3: Modular Design Using AI Assistance

### Prompt Used:



## Code and Output Generated:

First, let's create a dummy `marks.csv` file to demonstrate the functionality.

```
[7] ✓ Os import pandas as pd

# Create a dummy DataFrame for demonstration
data = {
    'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Score': [85, 92, 78, 88, 95],
    'Subject': ['Math', 'Science', 'Math', 'English', 'Science']
}
df_dummy = pd.DataFrame(data)

# Save the DataFrame to a CSV file
df_dummy.to_csv('marks.csv', index=False)

print("Dummy 'marks.csv' created successfully:")
print(df_dummy)
```

\*\*\* Dummy 'marks.csv' created successfully:

	Student	Score	Subject
0	Alice	85	Math
1	Bob	92	Science
2	Charlie	78	Math
3	David	88	English
4	Eve	95	Science

```
import pandas as pd

# Read the CSV file into a pandas DataFrame
try:
    df = pd.read_csv('marks.csv')
    print("CSV file 'marks.csv' loaded successfully.")
    display(df.head())

    # Specify the numeric column for calculations
    numeric_column = 'Score' # Assuming 'Score' is the numeric column

    # Check if the column exists and is numeric
    if numeric_column in df.columns and pd.api.types.is_numeric_dtype(df[numeric_column]):
        # Calculate mean, minimum, and maximum values
        mean_value = df[numeric_column].mean()
        min_value = df[numeric_column].min()
        max_value = df[numeric_column].max()

        print(f"Statistics 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}' not found or is not numeric in 'marks.csv'.")

except FileNotFoundError:
    print("Error: 'marks.csv' not found. Please make sure the file is in the correct directory.")
except Exception as e:
    print(f"An error occurred: {e}")
```

\*\*\* CSV file 'marks.csv' loaded successfully:

	Student	Score	Subject
0	Alice	85	Math
1	Bob	92	Science
2	Charlie	78	Math
3	David	88	English
4	Eve	95	Science

Statistics for column 'Score':

Mean: 87.60  
Minimum: 78.00  
Maximum: 95.00

## Verdict:

The generated code correctly processes CSV data and performs accurate calculations.

Using libraries like pandas improves efficiency and reduces complexity.

AI assistance is helpful for data analysis tasks with minimal effort.

## Task 4:

### Code Prompt:

**z** a. generate python code for bubble sort b. generate python code for built-in sort

◆ Certainly! Here are two Python code snippets: one demonstrating the bubble sort algorithm and another showing how to use Python's built-in `sort()` method or `sorted()` function.

The Python code for both Bubble Sort and the built-in sorting methods ( `.sort()` and `sorted()` ) executed successfully!

**For Bubble Sort:**

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

### Code and Output Generated:

### ▼ a. Python code for Bubble Sort

```
[8]
✓ Os
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(my_list.copy()) # Use .copy() to avoid modifying the original list if
    print(f"Sorted list using Bubble Sort: {sorted_list_bubble}")

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

### ▼ b. Python code for Built-in Sort

```
[9]
✓ Os
# Python provides two main ways to sort lists: the .sort() method and the sorted() function.

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

print("\n---")

# Example using the sorted() function (returns a new sorted list, leaves original unchanged):
my_list_3 = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for sorted() function: {my_list_3}")
sorted_list_builtin = sorted(my_list_3)
print(f"Sorted list using sorted() function: {sorted_list_builtin}")
print(f"Original list after using sorted() function: {my_list_3}")

... Original list for .sort() method: [64, 34, 25, 12, 22, 11, 90]
Sorted list using .sort() method: [11, 12, 22, 25, 34, 64, 90]

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

### Verdict:

Bubble sort code demonstrates basic algorithm understanding but is inefficient.

Built-in sort is simpler, faster, and more suitable for real applications.

It clearly shows the difference between learning algorithms and using optimized functions.