Assignment-2.3

2303A51304

Batch-05

Task1:

Prompt: Generate Python code that reads a text file and counts word

Code:

```python
import collections
import re
import os

def count_word_frequency(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            # Read the file and convert to lowercase for uniform counting
            text = file.read().lower()
            # Use regex to find words (ignoring punctuation)
            words = re.findall(r'\b\w+\b', text)

            # Count frequencies
            word_counts = collections.Counter(words)

            # Print the top 10 most common words
            print(f"{'Word':<15} | {'Frequency':<10}")
            print("-" * 28)
            for word, count in word_counts.most_common(10):
                print(f"{word:<15} | {count:<10}")
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
# --- Execution Block ---
if __name__ == "__main__":
    # 1. Define the filename
    filename = 'test_log.txt'
    # 2. Create a dummy file so the script has something to read
    with open(filename, 'w') as f:
        f.write("System check. System check. Error found in system. Error: Timeout. Check connection.")
    # 3. Run the function
    print(f"Analyzing {filename}...\n")
    count_word_frequency(filename)
```

Output:

```
Problems  Output  Debug Console  Terminal  Ports

PS C:\Users\pooji> & C:/Users/pooji/AppData/Local/Python/bin/python.exe c:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/As
PS C:\Users\pooji> & C:/Users/pooji/AppData/Local/Python/bin/python.exe c:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/As
Analyzing test_log.txt...

Word            | Frequency
----------------------------
system          | 3
check           | 3
error           | 2
found           | 1
in              | 1
timeout         | 1
connection      | 1
timeout         | 1
connection      | 1
```

Explanation:

The code reads a text file, converts all text to lowercase, and extracts only words using a regular expression so punctuation is ignored.

It uses collections. Counter to count how many times each word appears in the file.

The program then displays the top 10 most frequent words in a neatly formatted table.

Error handling (try–except) is included to manage cases like missing files or unexpected issues.

In the main block, a sample file is created, analyzed by the function, and optionally removed after execution.

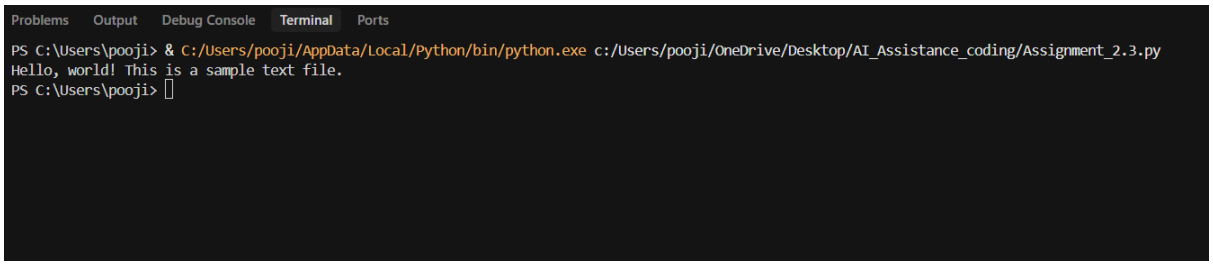
Task2:

Prompt: Write a python code that creates a text file, writes sample text and reads and displays the content

Code:

```python
# Create and write to the file
filename = "sample_text.txt"
with open(filename, 'w', encoding='utf-8') as file:
    file.write("Hello, world! This is a sample text file.")
# Read the file and print the contents
with open(filename, 'r', encoding='utf-8') as file:
    print(file.read())
```

Output:



Explanation:

The program creates a text file named sample.txt using write mode.

It writes sample text into the file using the write() function.

The file is then opened in read mode to access its contents.

The read() function is used to read the entire file content.

Finally, the content of the file is displayed on the screen using print().

Task3:

Prompt: Write a python code read a CSV file and calculate mean, min, and max.

Code:

```
import pandas as pd

# Create a sample DataFrame directly in the code
# You can replace this with your own data
data = {
    'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'B'],
    'Value': [10, 15, 12, 20, 18, 11, 25, 16],
    'Another_Value': [1, 2, 3, 4, 5, 6, 7, 8]
}
df = pd.DataFrame(data)

print("DataFrame created successfully. Here are the first 5 rows:")
display(df.head())
```

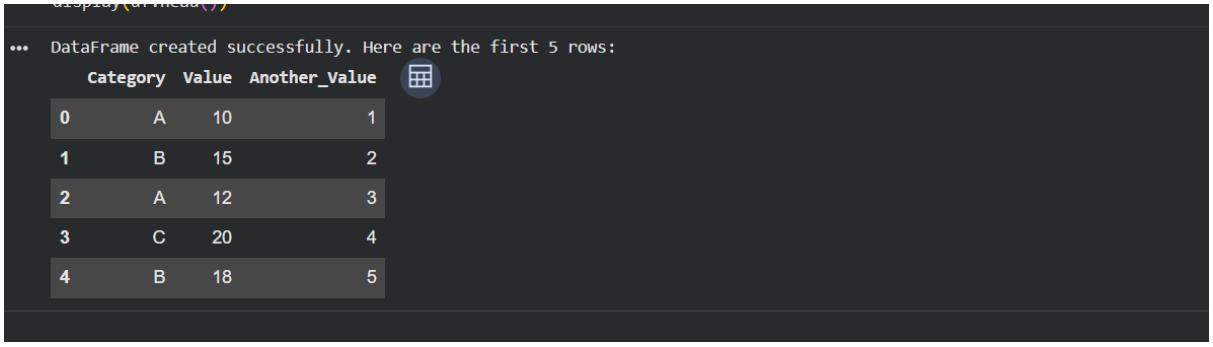
```
DataFrame created successfully. Here are the first 5 rows:
```

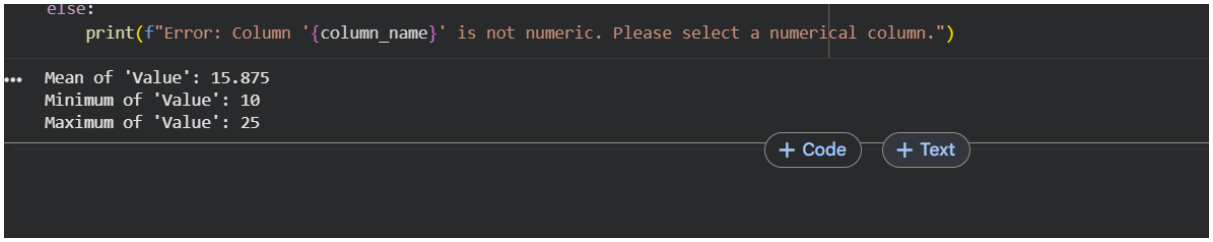
	Category	Value	Another_Value
0	A	10	1
1	B	15	2
2	A	12	3
3	C	20	4
4	B	18	5

```
# Replace 'Value' with the name of the column you want to analyze
column_name = 'Value' # Change this to your actual numerical column name

if column_name in df.columns and pd.api.types.is_numeric_dtype(df[column_name]):
    mean_value = df[column_name].mean()
    min_value = df[column_name].min()
    max_value = df[column_name].max()

    print(f"Mean of '{column_name}': {mean_value}")
    print(f"Minimum of '{column_name}': {min_value}")
    print(f"Maximum of '{column_name}': {max_value}")
elif column_name not in df.columns:
    print(f"Error: Column '{column_name}' not found in the DataFrame. Available columns are: {df.columns.tolist()}")
else:
    print(f"Error: Column '{column_name}' is not numeric. Please select a numerical column.")
```

```
Mean of 'Value': 15.875
Minimum of 'Value': 10
Maximum of 'Value': 25
```

Output:





Explanation:

The code creates a Pandas data frame with sample categorical and numerical data.

It displays the first five rows to preview the dataset.

The program checks whether the selected column (Value) exists and is numeric.

If valid, it calculates and prints the mean, minimum, and maximum values of that column.

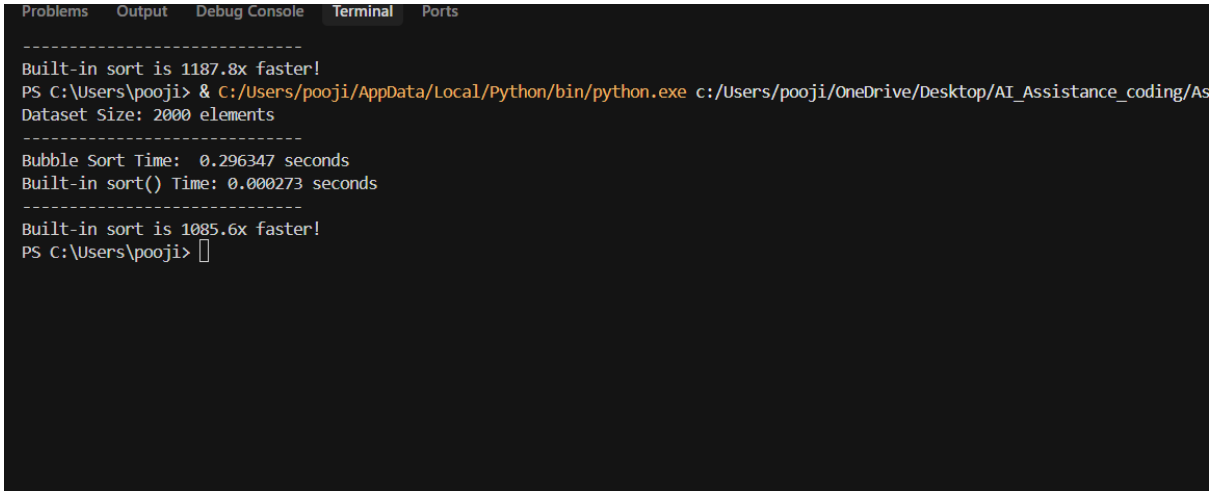
Task4:

Prompt: write a python code for generate: Bubble sort ,Python's built-in sort()and Compare both implementations.

Code:

```python
import time
import random
# 1. Bubble Sort Implementation
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):
            if arr[j] > arr[j + 1]:
                # Swap if the element found is greater than the next element
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# Generate a list of 2000 random integers
data_size = 2000
test_data = [random.randint(1, 10000) for _ in range(data_size)]

# 2. Compare Bubble Sort
data_copy1 = test_data.copy()
start_time = time.time()
bubble_sort(data_copy1)
bubble_duration = time.time() - start_time

# 3. Compare Python's Built-in sort()
data_copy2 = test_data.copy()
start_time = time.time()
data_copy2.sort()
builtin_duration = time.time() - start_time

# Display Results
print(f"Dataset Size: {data_size} elements")
print("-" * 30)
print(f"Bubble Sort Time:  {bubble_duration:.6f} seconds")
print(f"Built-in sort() Time: {builtin_duration:.6f} seconds")
print("-" * 30)
print(f"Built-in sort is {bubble_duration / builtin_duration:.1f}x faster!")
```

Output:



Explanation: The program implements Bubble Sort and measures the time taken to sort 2000 random numbers.

Bubble Sort repeatedly compares adjacent elements and swaps them, making it slow for large datasets.

The same data is sorted using Python's built-in sort() method, which uses an efficient algorithm.

Execution time for both methods is calculated using the time module. The results show that Python's built-in sort is many times faster than Bubble Sort