

ASSIGNMENT 2.3

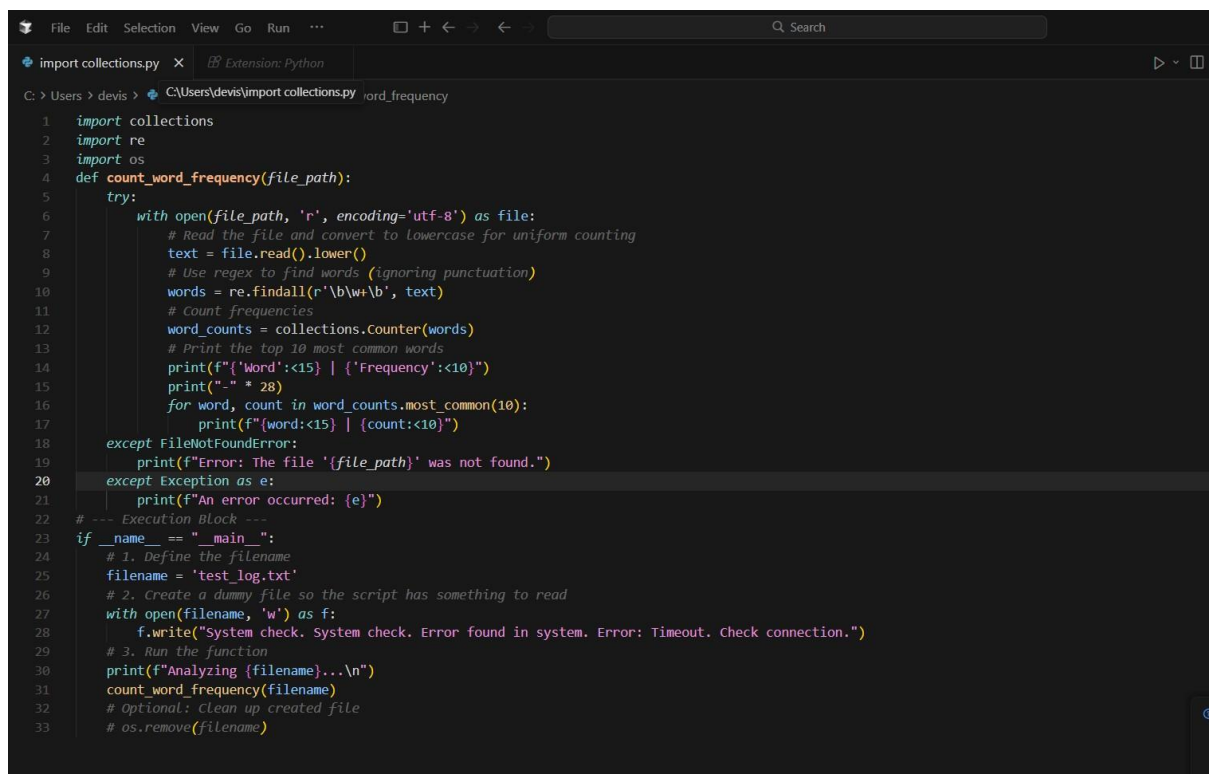
2303A51314

Batch – 05

Task – 01

Prompt : Write a Python script to analyze a log file and count keyword frequency.

Code :



```
1 import collections
2 import re
3 import os
4 def count_word_frequency(file_path):
5     try:
6         with open(file_path, 'r', encoding='utf-8') as file:
7             # Read the file and convert to lowercase for uniform counting
8             text = file.read().lower()
9             # Use regex to find words (ignoring punctuation)
10            words = re.findall(r'\b\w+\b', text)
11            # Count frequencies
12            word_counts = collections.Counter(words)
13            # Print the top 10 most common words
14            print(f'Word:<15 | {'Frequency':<10}')
15            print("-" * 28)
16            for word, count in word_counts.most_common(10):
17                print(f'Word:<15 | {count:<10}')
18        except FileNotFoundError:
19            print(f'Error: The file '{file_path}' was not found.")
20        except Exception as e:
21            print(f'An error occurred: {e}')
22
23 # --- Execution Block ---
24 if __name__ == "__main__":
25     # 1. Define the filename
26     filename = 'test_log.txt'
27     # 2. Create a dummy file so the script has something to read
28     with open(filename, 'w') as f:
29         f.write("System check. System check. Error found in system. Error: Timeout. Check connection.")
30     # 3. Run the function
31     print(f'Analyzing {filename}...\n')
32     count_word_frequency(filename)
33     # Optional: Clean up created file
34     # os.remove(filename)
```

Output :

```
PS C:\Users\devis> & C:/Users/devis/AppData/Local/Python/bin/python.exe "c:/Users/devis/import_collections.py"
Analyzing test_log.txt...

Word          | Frequency
-----
system        | 3
check         | 3
error         | 2
found         | 1
in            | 1
timeout       | 1
connection    | 1
PS C:\Users\devis>
```

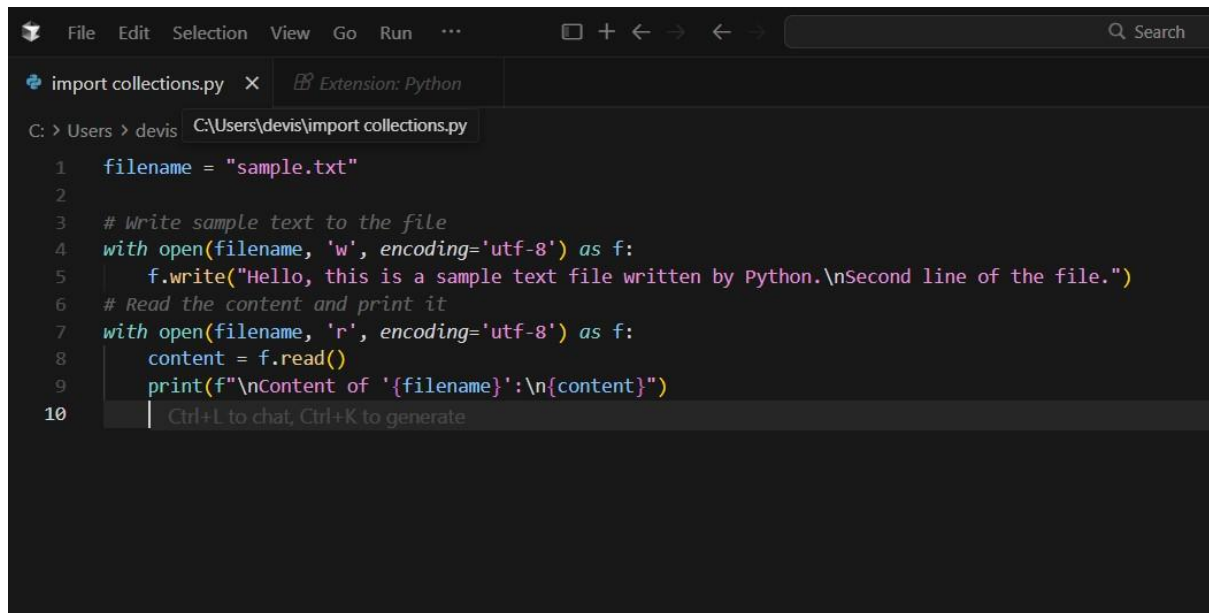
Explanation :

- Imports libraries for word counting (Counter), pattern matching (re), and file handling (os).
- Reads a text file and converts all text to lowercase.
- Uses regex to extract words while ignoring punctuation.
- Counts how many times each word appears.
- Displays the top 10 most frequent words in table format.
- Uses try-except to handle missing files or runtime errors.
- Creates a sample file for testing the function.
- Calls the function to analyze the file content.
- Optional line can delete the test file after execution.

TASK – 02

Prompt : Write a Python program that creates a text file, writes sample text into it, then reads and prints the file content.

Code :



```
import collections.py  Extension: Python
C: > Users > devis C:\Users\devis\import collections.py

1 filename = "sample.txt"
2
3 # Write sample text to the file
4 with open(filename, 'w', encoding='utf-8') as f:
5     f.write("Hello, this is a sample text file written by Python.\nSecond line of the file.")
6 # Read the content and print it
7 with open(filename, 'r', encoding='utf-8') as f:
8     content = f.read()
9     print(f"\nContent of '{filename}':\n{content}")
10 | Ctrl+L to chat, Ctrl+K to generate
```

Output :



```
Problems Output Debug Console Terminal Ports
PS C:\Users\devis> & C:/Users/devis/AppData/Local/Python/bin/python.exe "c:/Users/devis/import collections.py"

Content of 'sample.txt':
Hello, this is a sample text file written by Python.
Second line of the file.
PS C:\Users\devis>
```

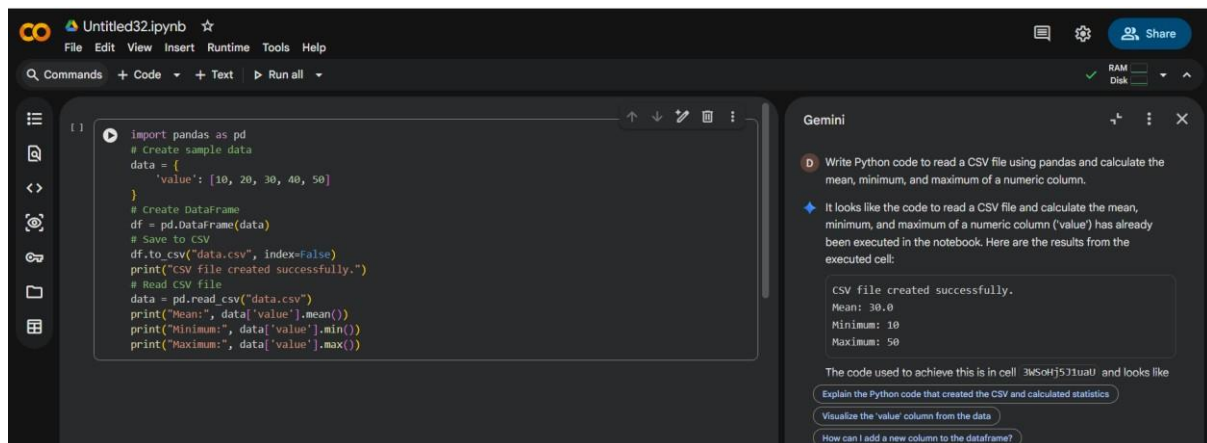
Explanation :

- Stores the file name in a variable.
- Opens the file in write mode and writes sample text into it.
- Automatically closes the file after writing.
- Reopens the file in read mode.
- Reads the entire content and prints it to the screen.

Task – 03

Prompt : Write Python code to read a CSV file using pandas and calculate the mean, minimum, and maximum of a numeric column.

Code :



```
import pandas as pd
# Create sample data
data = {
    'value': [10, 20, 30, 40, 50]
}
# Create DataFrame
df = pd.DataFrame(data)
# Save to CSV
df.to_csv("data.csv", index=False)
print("CSV file created successfully.")
# Read CSV file
data = pd.read_csv("data.csv")
print("Mean:", data['value'].mean())
print("Minimum:", data['value'].min())
print("Maximum:", data['value'].max())
```

Gemini

Write Python code to read a CSV file using pandas and calculate the mean, minimum, and maximum of a numeric column.

It looks like the code to read a CSV file and calculate the mean, minimum, and maximum of a numeric column ('value') has already been executed in the notebook. Here are the results from the executed cell:

```
CSV file created successfully.
Mean: 30.0
Minimum: 10
Maximum: 50
```

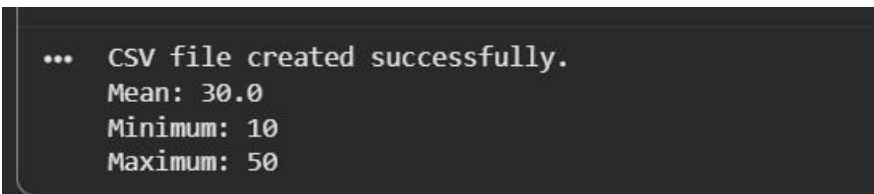
The code used to achieve this is in cell 3K5oHj5J1uauU and looks like

[Explain the Python code that created the CSV and calculated statistics](#)

[Visualize the 'value' column from the data](#)

[How can I add a new column to the dataframe?](#)

Output :



```
... CSV file created successfully.
Mean: 30.0
Minimum: 10
Maximum: 50
```

Explanation :

- Imports the pandas library for data handling.
- Creates sample numeric data in a dictionary.
- Converts the data into a DataFrame.
- Saves the DataFrame as a CSV file.
- Reads the CSV file back into the program.
- Calculates and prints the mean, minimum, and maximum values.

TASK – 04

Prompt : Write Python code for Bubble Sort and Python's built-in sort() method, then give a short comparison of both in terms of efficiency and simplicity.

Code :

```
import collections.py • Python Extension: Python
C:\Users\devis> python import collections.py ...

1 def bubble_sort(arr):
2     n = len(arr)
3     # Traverse through all array elements
4     for i in range(n):
5         # Last i elements are already in place, so we don't check them
6         for j in range(0, n - i - 1):
7             # Traverse the array from 0 to n-i-1
8             # Swap if the element found is greater than the next element
9             if arr[j] > arr[j + 1]:
10                arr[j], arr[j + 1] = arr[j + 1], arr[j]
11    return arr
12 # --- Execution ---
13 if __name__ == "__main__":
14     # Test Data
15     data_1 = [64, 34, 25, 12, 22, 11, 90]
16     data_2 = [64, 34, 25, 12, 22, 11, 90]
17     print(f"Original List: {data_1}\n")
18     # 1. Bubble Sort (Manual)
19     sorted_bubble = bubble_sort(data_1)
20     print(f"Bubble Sort Result: {sorted_bubble}")
21     # 2. Python Built-in Sort
22     # .sort() modifies the list in-place
23     data_2.sort()
24     print(f"Built-in Sort Result: {data_2}")
```

Output :

```
Problems  Output  Debug Console  Terminal  Ports
Original List: [64, 34, 25, 12, 22, 11, 90]

Bubble Sort Result:  [11, 12, 22, 25, 34, 64, 90]
Built-in Sort Result: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\devis>
```

Explanation :

- Bubble Sort is a manual sorting algorithm that repeatedly swaps adjacent elements, it is easy to understand but slow for large lists.
- Built-in sort() is optimized and much faster because it uses Python's efficient internal algorithm.
- Bubble Sort has higher time complexity ($O(n^2)$), while built-in sort is more efficient ($\approx O(n \log n)$).
- Built-in sort() is preferred in real applications due to better performance and simplicity.