**Assignment 4.1**

**Name:** B. Divija

**Batch :13**

**Task #1 – Zero-Shot Prompting with Conditional Validation**

Objective

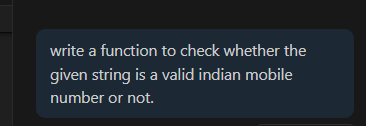
Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.

Requirements

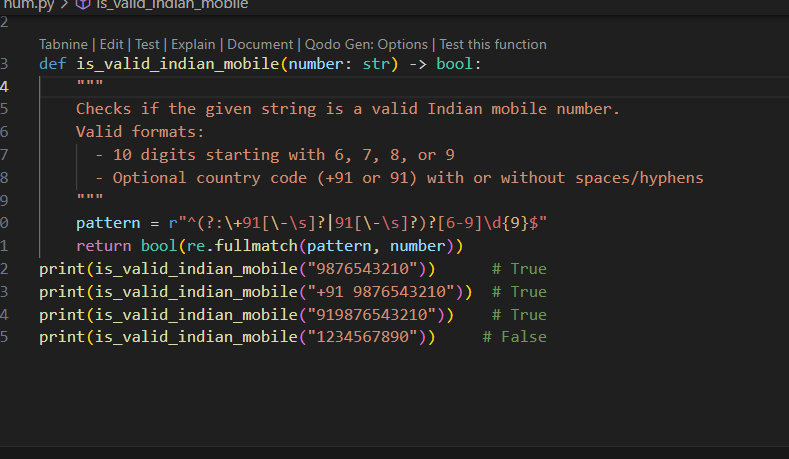
* The function must ensure the mobile number:
  + Starts with 6, 7, 8, or 9
  + Contains exactly 10 digits

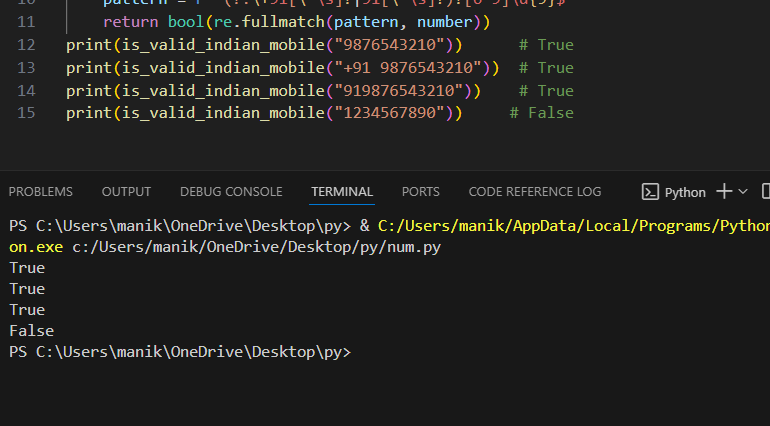
Using Github copilot:

Prompt:



Code:



Output: 

Observation:

* According to the given task Github copilot generated a function to check whether the given number is valid Indian number not without any input-output examples in the prompt.

**Task #2 – One-Shot Prompting with Edge Case Handling**

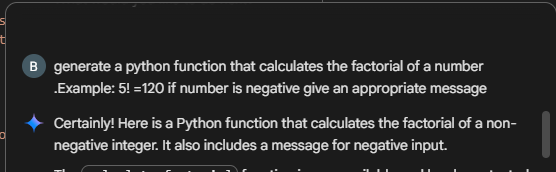
Objective

Use one-shot prompting to generate a Python function that calculates the factorial of a number.

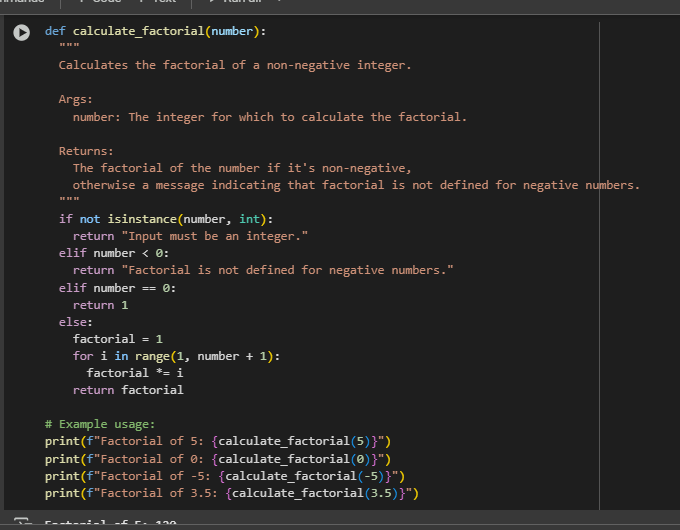
Requirements

* Provide one sample input-output pair in the prompt to guide the AI.
* The function should handle:
  + 0! correctly
  + Negative input by returning an appropriate message

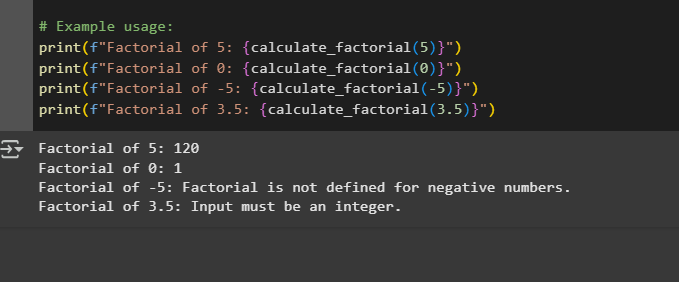
Prompt:



Code:



Output:



Observation:

Gemini generated the function to check the factorial of a number according to the prompt. Also it included a message for float number that is not in prompt.

**Task #3 – Few-Shot Prompting for Nested Dictionary Extraction**

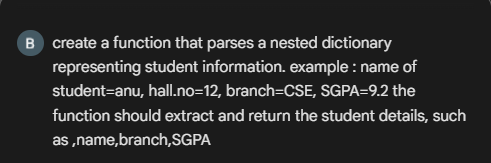
Objective

Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.

Requirements

* The function should extract and return:
  + Full Name
  + Branch
  + SGPA

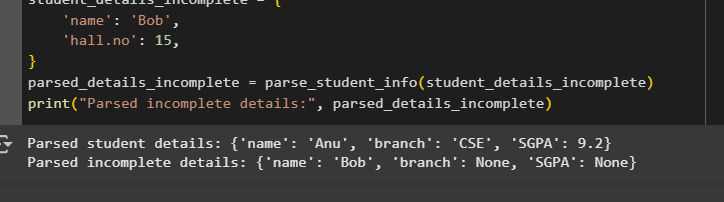
Prompt:



Code:



Output:



Observation: A simple observation:  
The function **parse\_student\_info** safely extracts a student's **name**, **branch**, and **SGPA** from a dictionary, even if some keys are missing—it will just return None for missing values instead of causing an error.

**Task #4 – Comparing Prompting Styles for File Analysis**

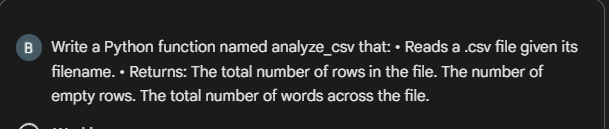
Objective

Experiment with zero-shot, one-shot, and few-shot prompting to generate functions for CSV file analysis.

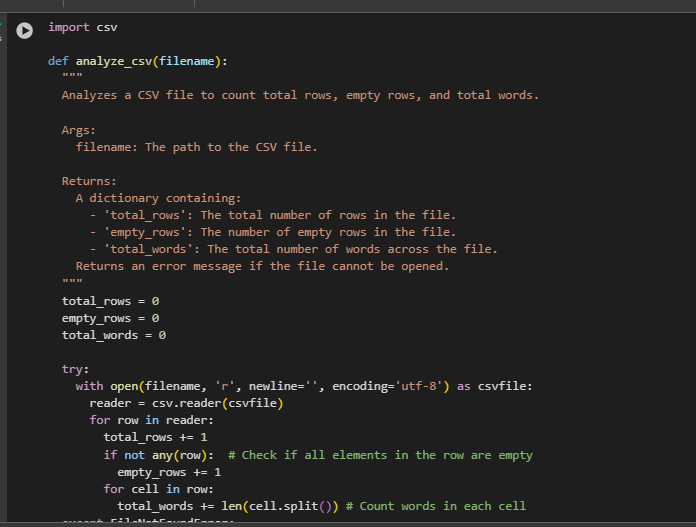
Requirements

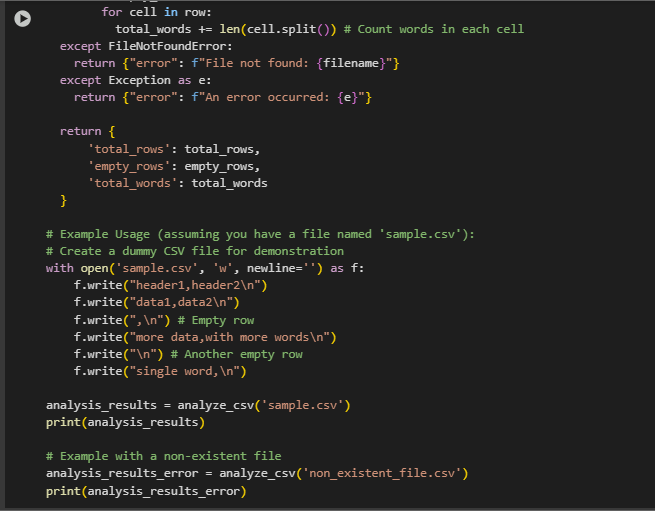
* Each generated function should:
  + Read a .csv file
  + Return the total number of rows
  + Count the number of empty rows
  + Count the number of words across the file

Prompt: 1. **Zero-Shot Prompt**

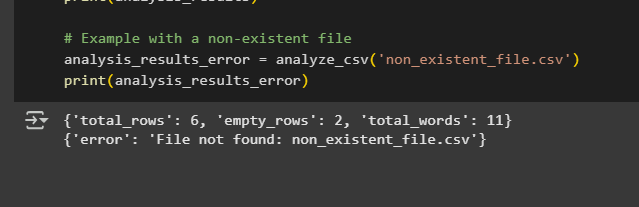


Code:



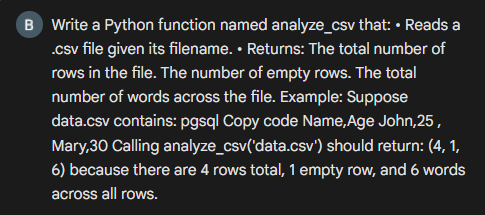


Output:

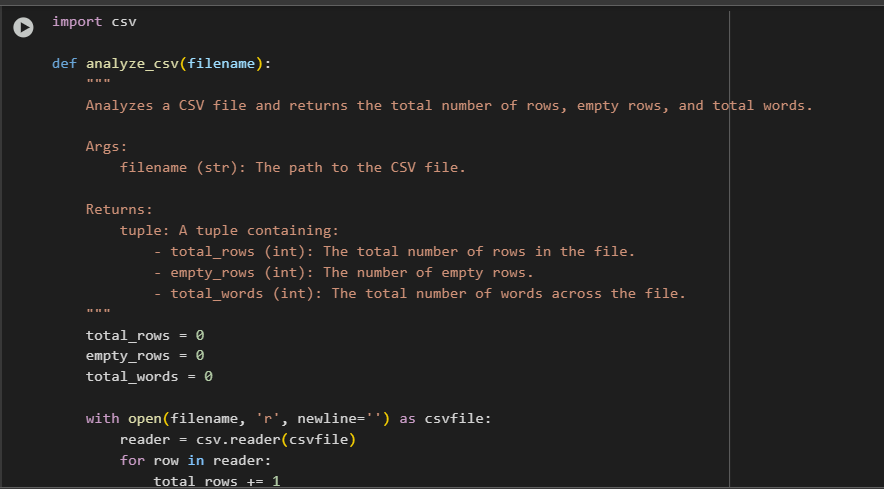


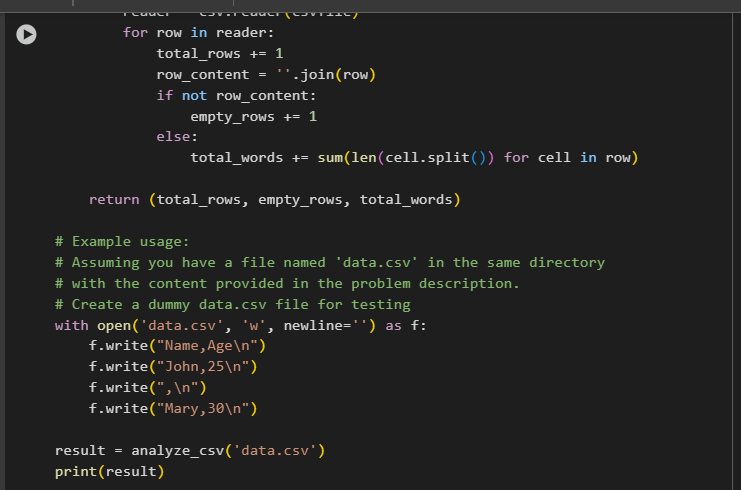
Observation: This code analyzes a CSV file to count total rows, empty rows, and total words, returning the results in a dictionary with error handling for missing files.

Prompt:2. **One-Shot Prompt**



Code:



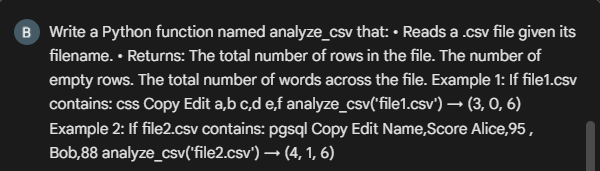


Output: (4, 1, 6)

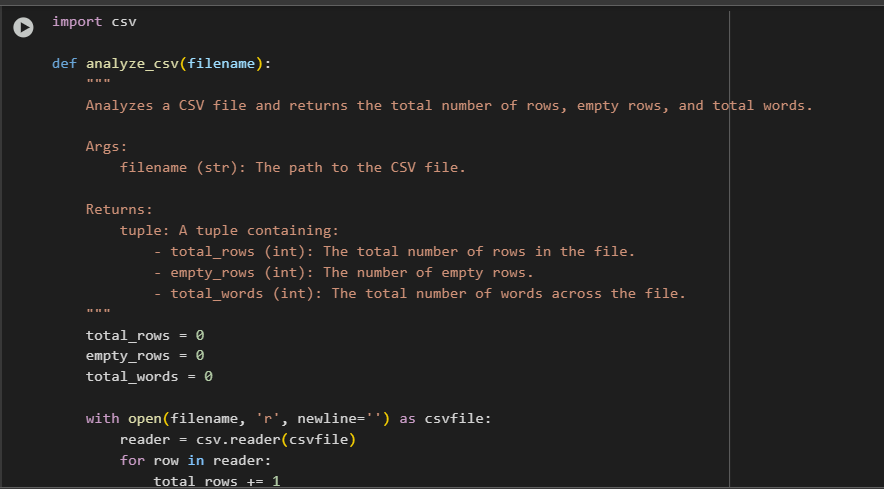
Observation:

* The function **analyze\_csv** correctly counts total rows, empty rows, and total words in a CSV file.
* It uses Python’s built-in **csv** module for reading data.
* Empty rows are detected by checking if the joined row content is empty.
* The example usage creates a sample CSV file and demonstrates the function’s output.

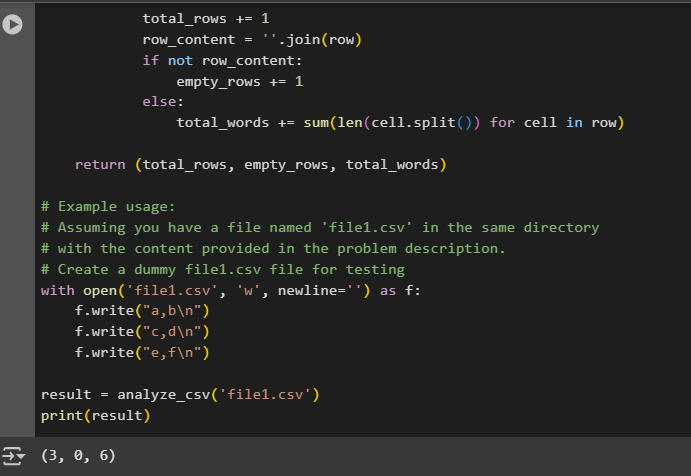
Prompt:3. **Few-Shot Prompt**



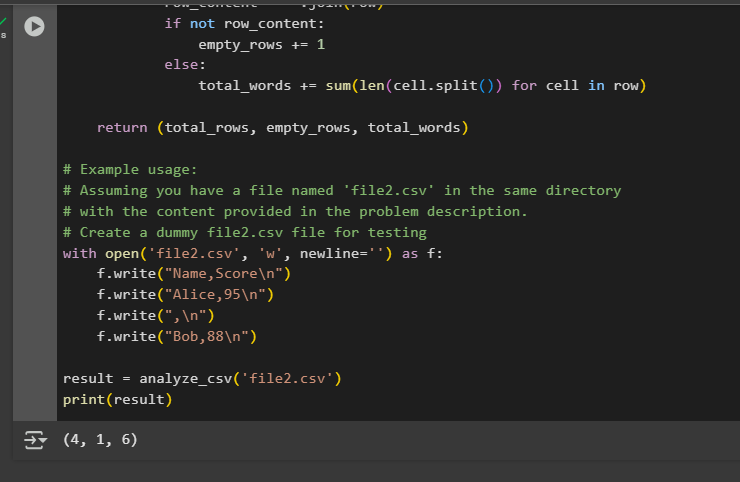
Code:



Example1 output:



Example2 output:



Observation:

* The **few-shot prompt** clearly specifies the requirements and gives 3 examples, which helps the AI generate a correct text-processing function.
* The CSV analysis code is tested with multiple sample files (file1.csv and file2.csv) to verify different scenarios.
* Output tuples (3, 0, 6) and (4, 1, 6) confirm correct row, empty row, and word counts.

**Task #5 – Few-Shot Prompting for Text Processing and Word** **Frequency**

Objective

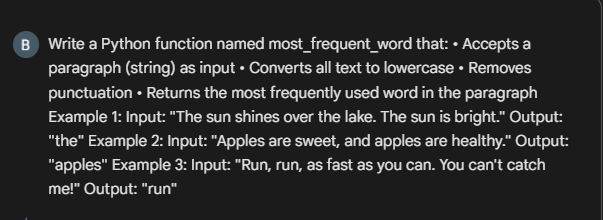
Use few-shot prompting (with at least 3 examples) to generate a Python function that processes text and analyzes word frequency.

Requirements

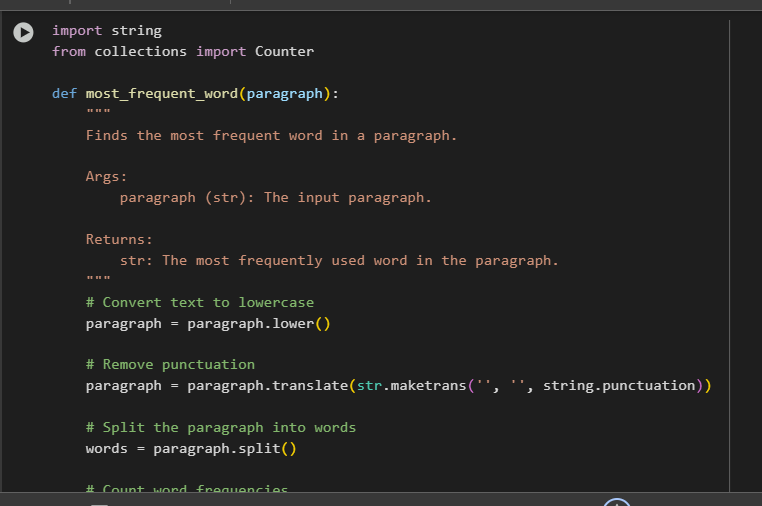
The function must:

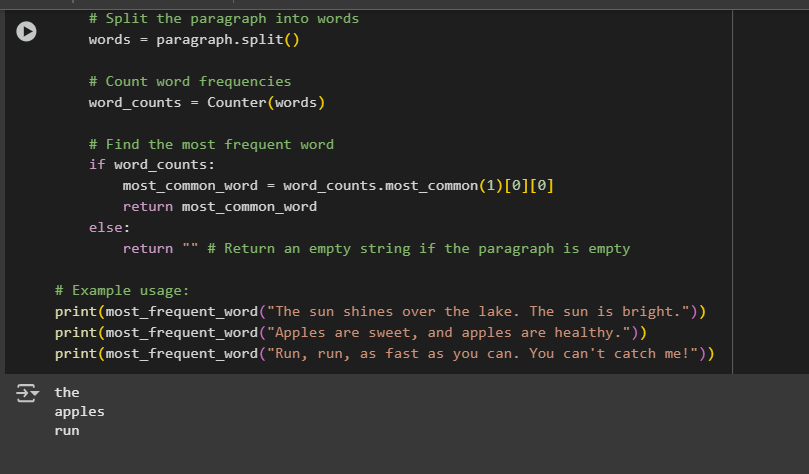
* Accept a paragraph as input
* Convert all text to lowercase
* Remove punctuation
* Return the most frequently used word

Prompt:



Code with output:





Observation:

1. **Clear logic flow** – You’ve broken it down into lowercase conversion, punctuation removal, splitting, counting, and finding the most common word.
2. **Good use of Counter** – It’s the easiest way to find word frequencies.
3. **Handles empty input** – Returns an empty string if no words are found.
4. **Readable variable names** – paragraph, words, word\_counts are self-explanatory.
5. **Docstring included** – Nicely explains what the function does, its arguments, and return value.