**HALL TICKET NUMBER: 2403A51365**

**NAME: RAYAPUDI HARITHA**

**BATCH: 24BTCAICSB14**

**Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques**

Assignment Number:4.1

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

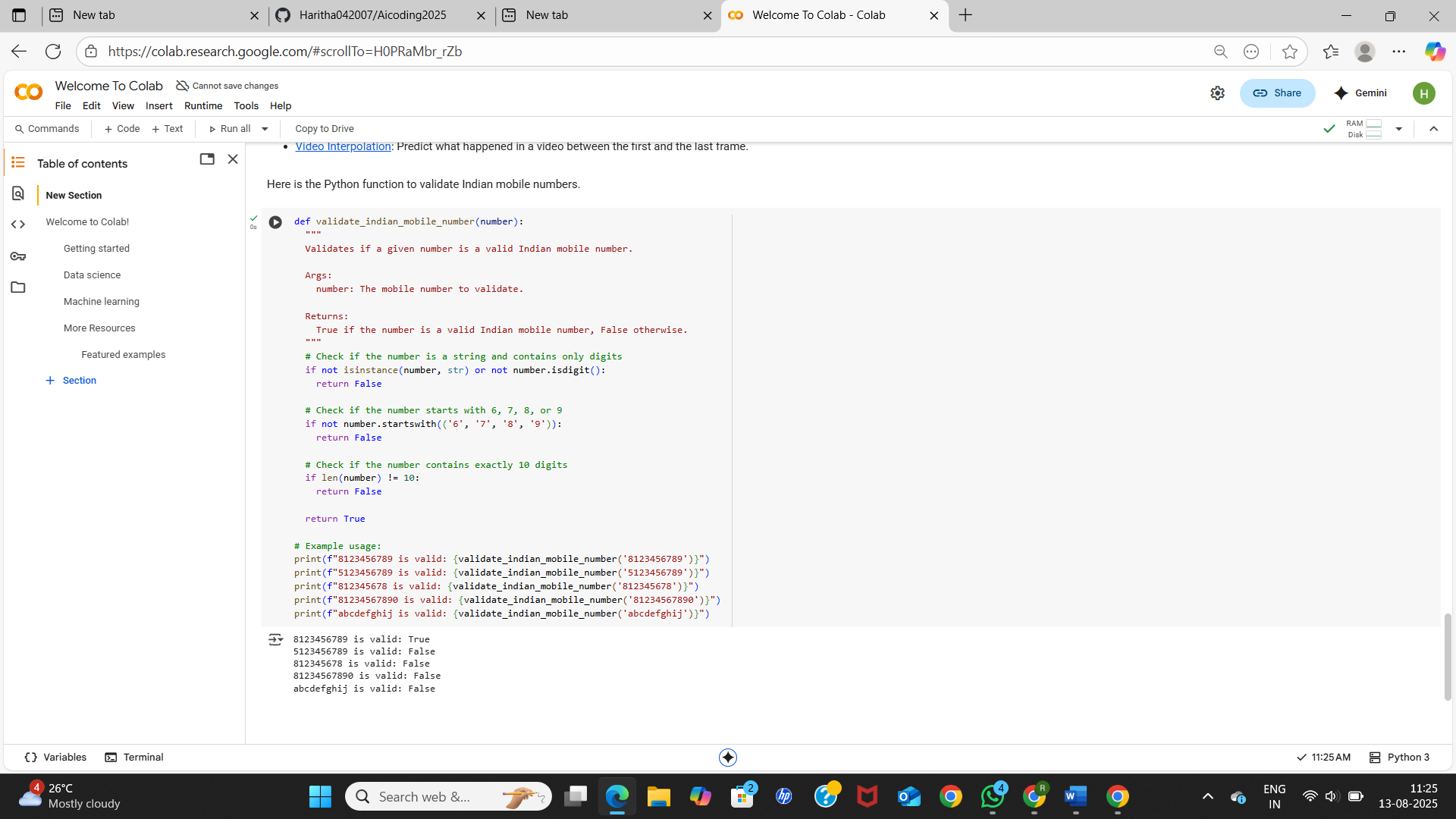
Objective

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

Code and output:-



**code explanation:-**

**Step 1:  
The function starts by checking whether the input is a string. Mobile numbers should be passed as strings, not as integers or other types.**

**Step 2:  
It then checks if the string contains only digits. This means the number should not include any letters, spaces, or special characters.**

**Step 3:  
Next, the function looks at the first digit of the number. In India, valid mobile numbers must begin with either 7, 8, or 9. If the number starts with any other digit, it is considered invalid.**

**Step 4:  
After that, the function checks the total length of the number. A valid Indian mobile number must be exactly 10 digits long. If it has more or fewer digits, it fails the validation.**

**Step 5:  
If the input passes all the above checks—being a string, containing only digits, starting with 7/8/9, and having exactly 10 digits—the function returns True, meaning the number is valid.**

**Step 6:  
If any of the checks fail, the function returns False, indicating that the number is not valid.**

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

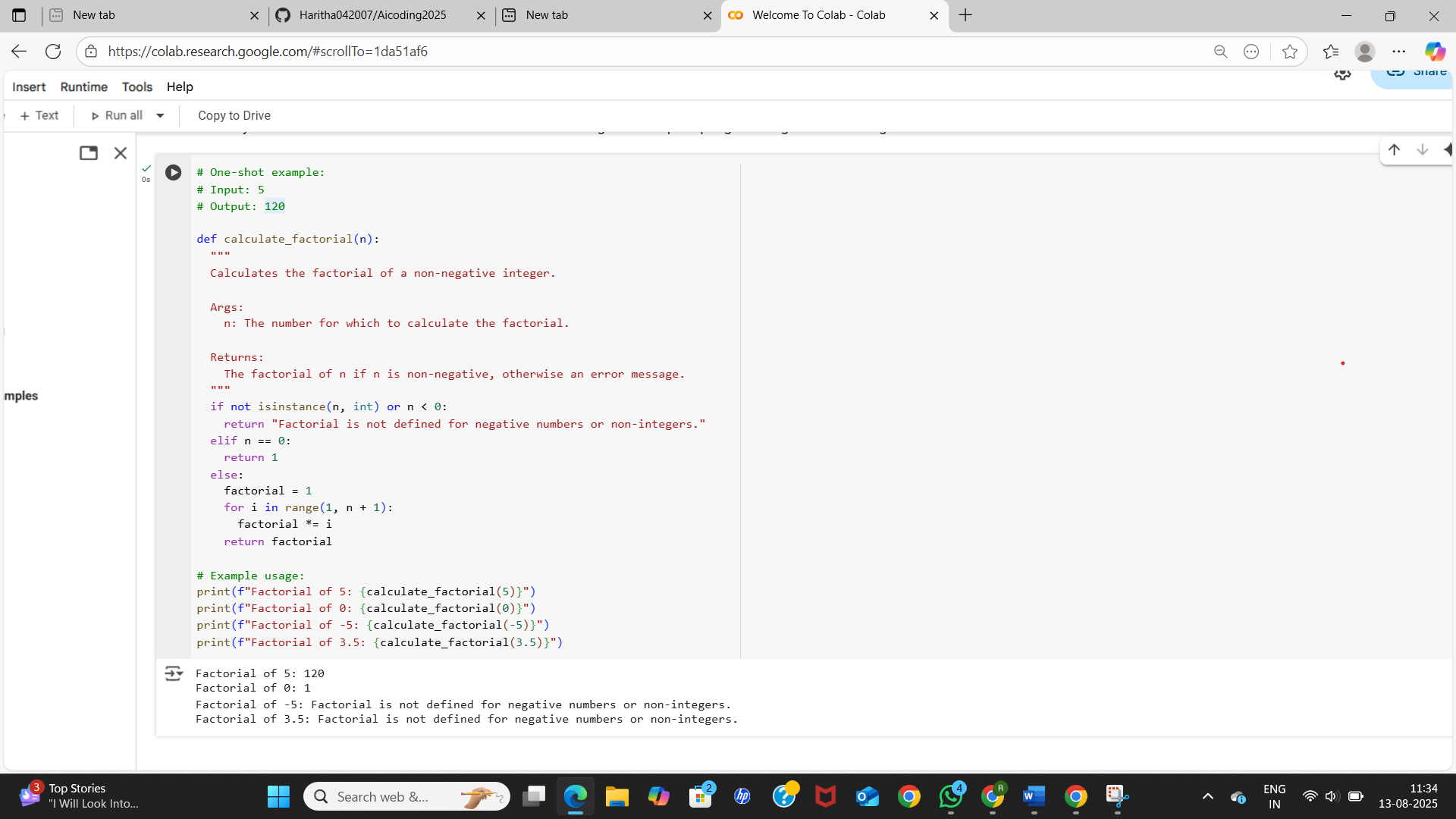
Objective

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.

Code and output:-



**code explanation:-**

**Step 1:  
Two variables, a and b, are defined with values 3 and 5. These will be used to test the function later.**

**Step 2:  
A function named calculate\_factorial(n) is defined. Its purpose is to calculate the factorial of a non-negative integer n.**

**Step 3:  
Inside the function, the first check ensures that n is an integer and that it is not negative. If n is either negative or not an integer (like a float or string), the function returns an error message:  
*"Factorial is not defined for negative numbers or non-integers."***

**Step 4:  
If n passes the check, the function initializes a variable called factorial with the value 1. This will be used to store the result.**

**Step 5:  
A for loop runs from 1 to n, multiplying factorial by each number in that range. This is how the factorial is calculated.**

**Step 6:  
After the loop finishes, the function returns the final value of factorial.**

**Step 7:  
The function is tested with several inputs:**

* **For a = 3, the output is 6 because 3! = 3 × 2 × 1.**
* **For b = 5, the output is 120 because 5! = 5 × 4 × 3 × 2 × 1.**
* **For -3, the function returns the error message because factorials aren't defined for negative numbers.**
* **For 5.5, the function also returns the error message because 5.5 is not an integer.**
* **For 7, the output is 5040 because 7! = 7 × 6 × 5 × 4 × 3 × 2 × 1.**

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

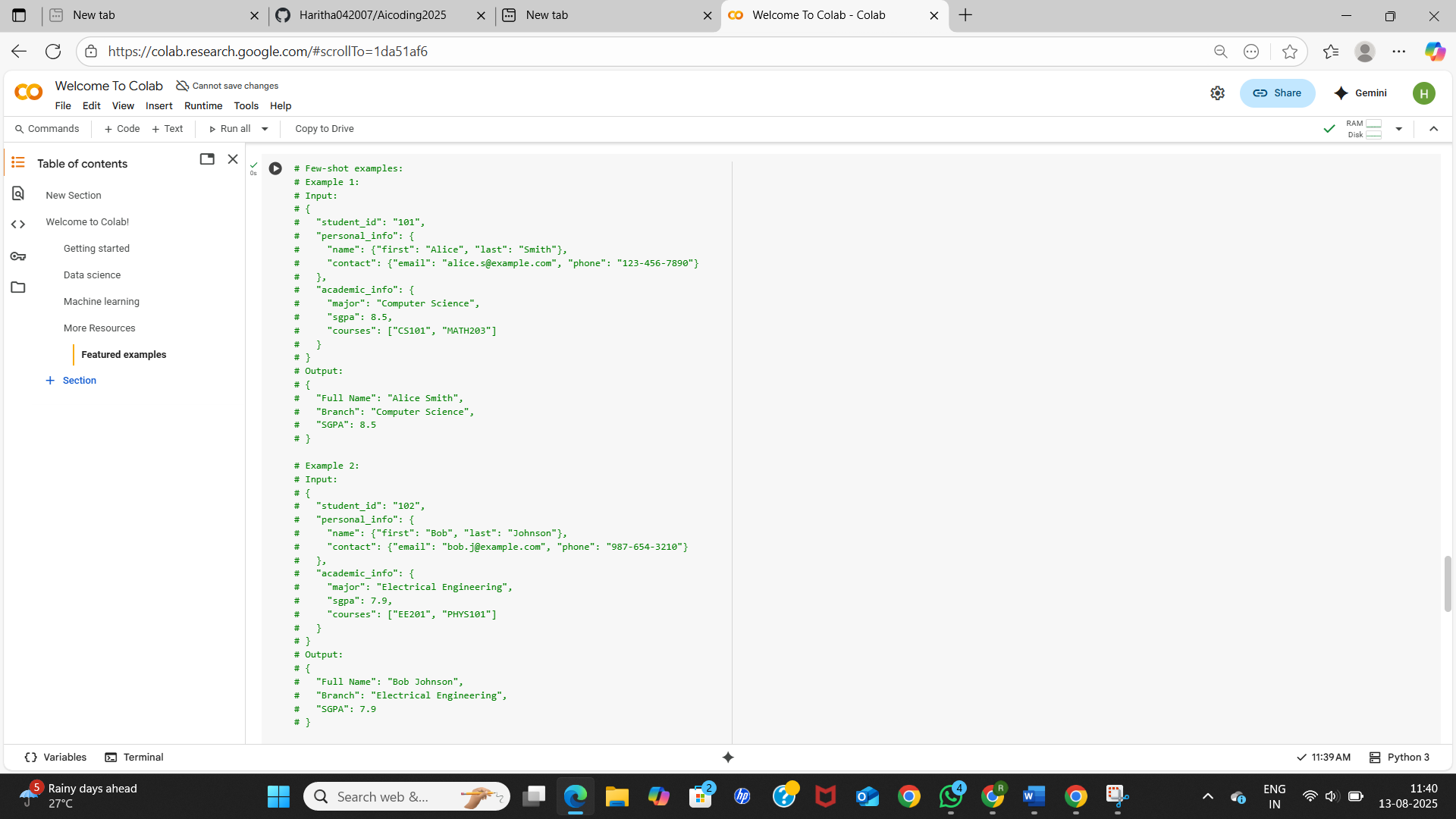
Objective

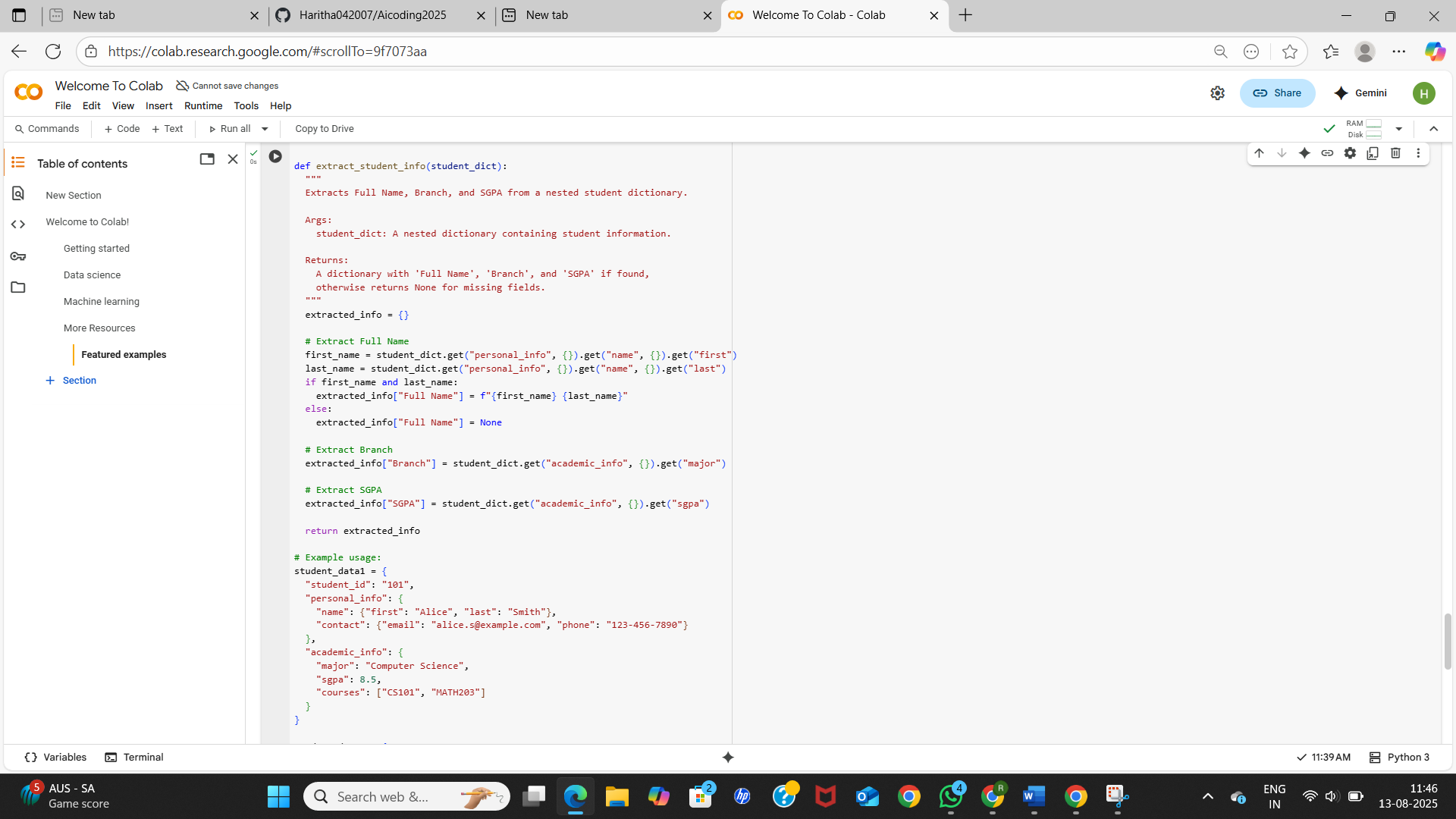
Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.

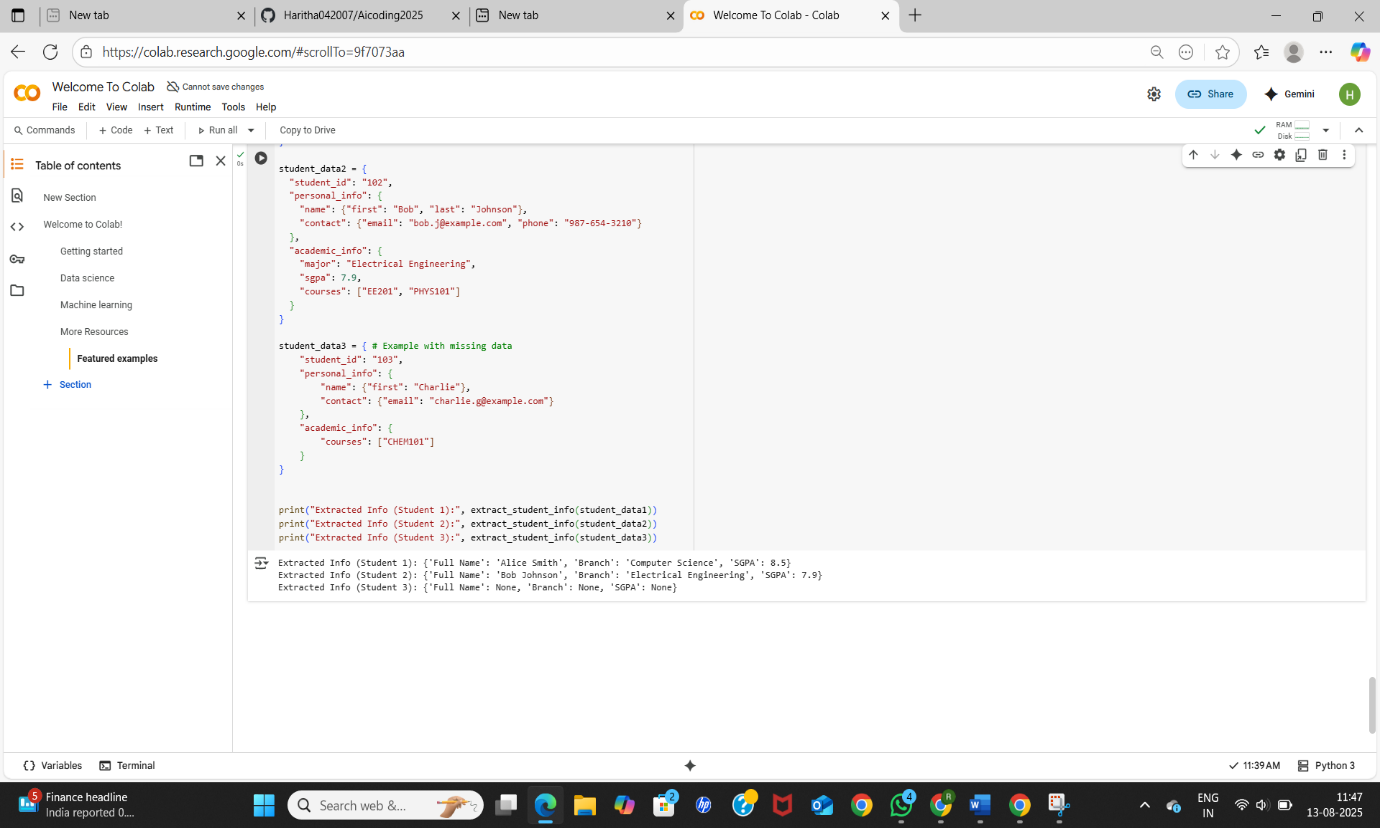
Requirements

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

Code:-







Output:-

A screenshot of a computer

AI-generated content may be incorrect.

Explanation:-

1. The function extract\_student\_info(student\_dict) is defined to take a nested dictionary of student details and extract three specific pieces of information: the student’s full name, branch, and SGPA.
2. A docstring is included to describe the purpose of the function, the argument it expects, and what it returns.
3. An empty dictionary extracted\_info is created to store the results.
4. The code tries to retrieve the student’s first name by looking inside personal\_info → name → first using chained .get() calls to avoid errors if keys are missing.
5. Similarly, the last name is retrieved from personal\_info → name → last.
6. If both first name and last name exist, they are combined into a single string with a space in between and stored under the key "Full Name" in extracted\_info.
7. If either the first name or last name is missing, "Full Name" is set to None.
8. The branch (major) is retrieved from academic\_info → major and stored in extracted\_info["Branch"].
9. The SGPA (gpa) is retrieved from academic\_info → gpa and stored in extracted\_info["SGPA"].
10. The extracted\_info dictionary containing all extracted values is returned to the caller.
11. Three example student dictionaries (student\_data1, student\_data2, and student\_data3) are created — the first two contain complete information, while the third one has missing fields.
12. The function is called with each example dictionary, and the extracted information is printed.
13. The output shows the full name, branch, and SGPA for students with complete data, and None for any fields that were missing in the input dictionary.

**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

zero-shot code:-

A screenshot of a computer

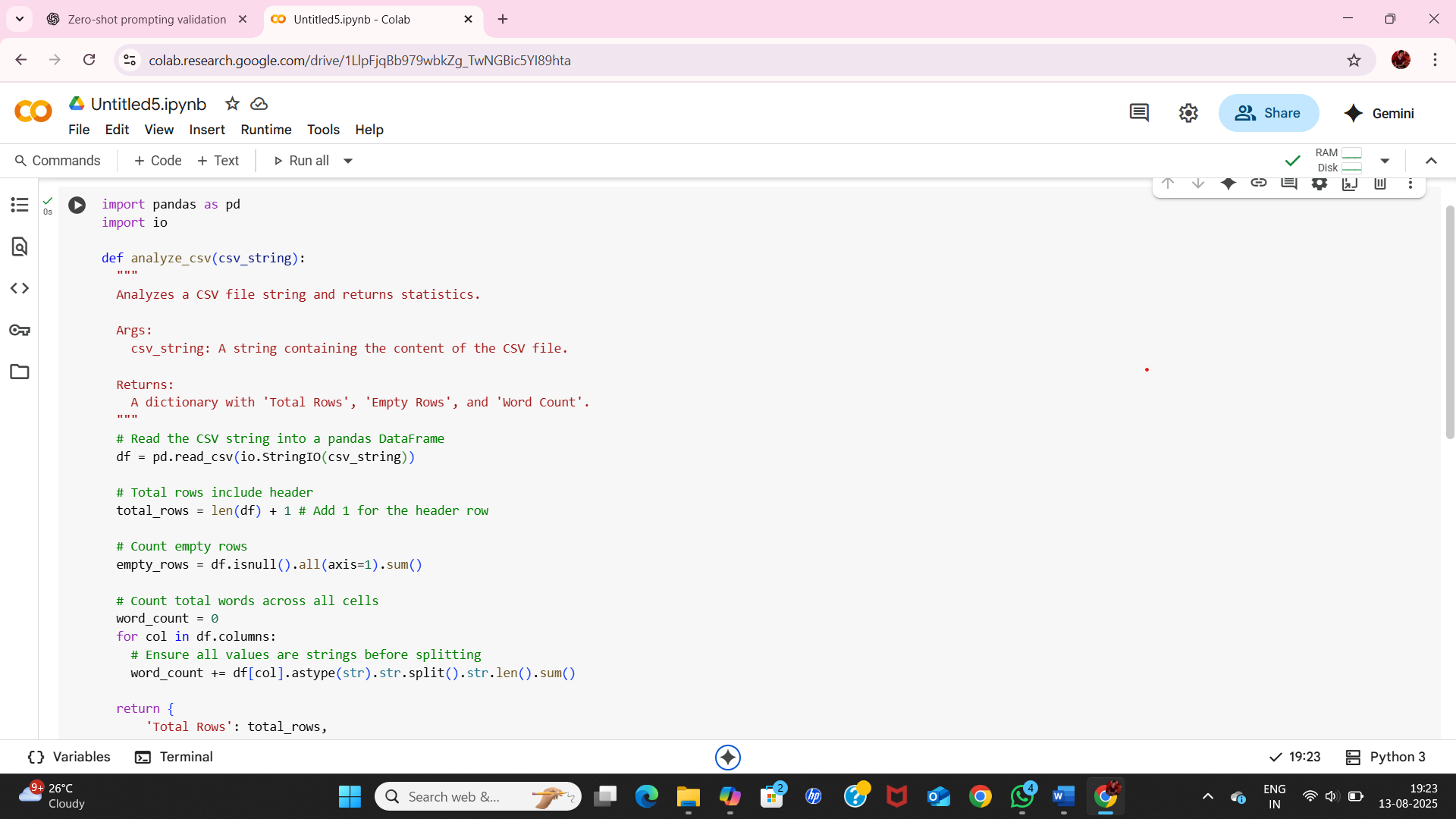
AI-generated content may be incorrect.

OUTPUT:-

A screenshot of a computer

AI-generated content may be incorrect.

One shot code and output:-



A screenshot of a computer

AI-generated content may be incorrect.

Few shot code and output:-

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Explanation:-

1. **Import pandas:**

import pandas as pd

This line imports the pandas library, which is a powerful tool for data manipulation and analysis in Python. It's commonly used for working with tabular data like CSV files. The as pd part is an alias, allowing us to refer to the pandas library as pd throughout the code.

1. **Define the function analyze\_csv\_file:**

def analyze\_csv\_file(file\_path):  
  """  
  Analyzes a CSV file and returns statistics.  
  
  Args:  
    file\_path: The path to the CSV file.  
  
  Returns:  
    A dictionary with 'Total Rows', 'Empty Rows', and 'Word Count'.  
  """  
  # ... function body ...

This defines a function named analyze\_csv\_file that takes one argument: file\_path, which is expected to be the path to the CSV file you want to analyze. The triple quotes ("""...""") enclose a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns).

1. **Error Handling (Try-except block):**

try:  
  # ... code that might raise errors ...  
except FileNotFoundError:  
  return f"Error: File not found at {file\_path}"  
except Exception as e:  
  return f"An error occurred: {e}"

This block is used for error handling. The code inside the try block will be executed. If a FileNotFoundError occurs (meaning the specified file\_path doesn't exist), the code in the first except block will run, returning an error message. If any other type of Exception occurs, the second except block will catch it and return a general error message including the specific error (e).

1. **Read the CSV file:**

df = pd.read\_csv(file\_path)

Inside the try block, this line uses the read\_csv function from pandas to read the data from the file specified by file\_path into a DataFrame named df. A DataFrame is a two-dimensional labeled data structure with columns of potentially different types, similar to a spreadsheet or SQL table.

1. **Calculate Total Rows:**

total\_rows = len(df) + 1 # Add 1 for the header row

len(df) gives the number of data rows in the DataFrame (excluding the header). We add 1 to account for the header row in the total count.

1. **Count Empty Rows:**

empty\_rows = df.isnull().all(axis=1).sum()

* df.isnull() creates a DataFrame of the same size as df but with True where values are missing (null or NaN) and False otherwise.
* .all(axis=1) checks if *all* values in a given *row* (axis=1) are True (i.e., if the entire row is null). This returns a pandas Series of boolean values (True for empty rows, False otherwise).
* .sum() on this boolean Series counts the number of True values, effectively giving us the number of empty rows.
  1. **Count Total Words:**

word\_count = 0  
for col in df.columns:  
  # Ensure all values are strings before splitting and handle potential NaN values  
  word\_count += df[col].astype(str).str.split().str.len().sum()

* This initializes word\_count to 0.
* It then iterates through each column of the DataFrame (for col in df.columns:).
* df[col] selects the current column as a pandas Series.
* .astype(str) converts all values in the column to strings. This is important to handle potential non-string data types and NaN values gracefully.
* .str.split() splits each string value in the column into a list of words based on whitespace.
* .str.len() calculates the length of each list of words (i.e., the word count for each cell).
* .sum() adds up the word counts for all cells in the current column.
* word\_count += ... adds the total word count for the current column to the running word\_count.
  1. **Return the results:**

return {  
    'Total Rows': total\_rows,  
    'Empty Rows': empty\_rows.item(), # Convert numpy int64 to standard Python int  
    'Word Count': word\_count.item() # Convert numpy int64 to standard Python int  
}

The function returns a dictionary containing the calculated 'Total Rows', 'Empty Rows', and 'Word Count'. .item() is used to convert the NumPy integer types returned by .sum() to standard Python integers, which can be helpful in some cases.

1. **Example Usage:** The remaining lines of the code block demonstrate how to use the analyze\_csv\_file function with example CSV data. It creates two temporary CSV files (example1.csv and example2.csv), writes the example data into them, calls the analyze\_csv\_file function with the file paths, and prints the results. This part is for demonstration purposes and would typically be replaced with your actual file path when using the function.

**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

Code and output:-

A screenshot of a computer

AI-generated content may be incorrect.

**Step 1:**  
The code begins by importing two Python modules: string (to handle punctuation) and Counter from collections (to count word frequencies).

**Step 2:**  
A function named analyze\_word\_frequency\_standard(paragraph) is defined. This function processes a paragraph by removing punctuation, converting all letters to lowercase, and identifying the most frequent word.

**Step 3:**  
Inside the function, the paragraph is first converted to lowercase using .lower(). This ensures that words like "This" and "this" are treated as the same.

**Step 4:**  
Next, punctuation is removed from the paragraph using str.translate() along with string.punctuation. This step cleans the text so that words are not affected by commas, periods, or other symbols.

**Step 5:**  
The cleaned paragraph is then split into individual words using .split(), which creates a list of words.

**Step 6:**  
The Counter function is used to count how many times each word appears in the list.

**Step 7:**  
The function then identifies the most frequent word using .most\_common(1)[0], which returns the word that appears the most along with its count.

**Step 8:**  
The result is returned and printed. In the example paragraph "This is a sample paragraph to test the function. This is a test.", the word "this" appears twice, so the output is ('this', 2).

Second Function: With Punctuation

**Step 9:**  
Another function called analyze\_word\_frequency\_with\_punctuation(paragraph) is defined. This one is similar to the first but does **not** remove punctuation.

**Step 10:**  
It still converts the paragraph to lowercase and splits it into words, but punctuation remains attached to the words (e.g., "test." is treated as different from "test").

**Step 11:**  
It counts word frequencies and returns the most frequent word just like the first function.

**Step 12:**  
In the example "This is a sample paragraph without punctuation! This is a test.", the word "this" still appears twice, so the output is again ('this', 2).