AI ASSISTED CODING

ASSIGNMENT – 2.2

COURSE : B.TECH\CSE

SPECIALISATION : AIML

NAME : MANDALA.TEJASRI

ENROLLMENT NUMBER : 2403A51171

COURSE : AI ASSISTED CODING

DATE : 16 AUGUST 2025

BATCH NO : 01

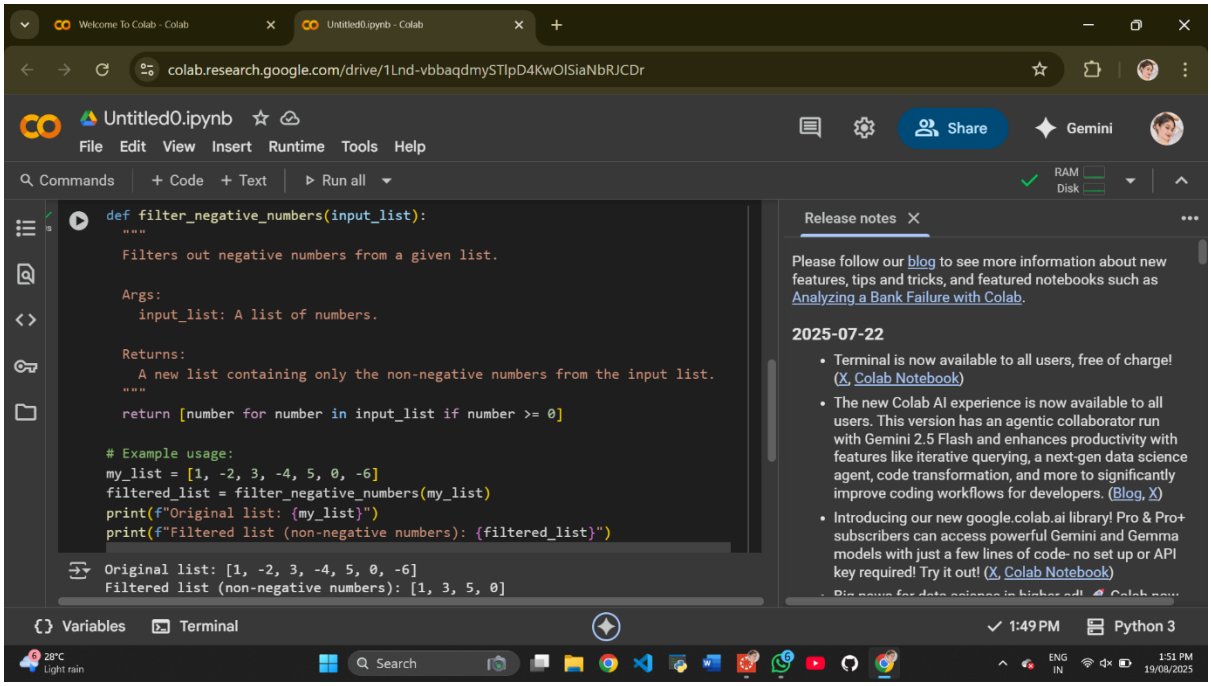
TASK DESCRIPTION#1

>>use gemini to generate all negative numbers from a given list

PROMPT :

>>Develop a python program to give all negative numbers of a given list

EXPECTED OUTPUT AND SCREENSHOT :



EXPLANATION :

Certainly! To filter out negative numbers from a list in Python, you can write a function that iterates through the list and keeps only the non-negative elements. Here's a breakdown of how you could do that:

1. **Define a function:** Start by defining a Python function that takes one argument, which will be the list you want to filter.
2. **Create an empty list:** Inside the function, initialize an empty list where you will store the non-negative numbers.
3. **Iterate through the input list:** Loop through each element in the list provided as input to the function.
4. **Check if the number is non-negative:** For each element, check if it is greater than or equal to zero.
5. **Append non-negative numbers:** If an element is non-negative, append it to the empty list you created in step 2.
6. **Return the new list:** After iterating through all elements in the input list, return the new list containing only the non-negative numbers.

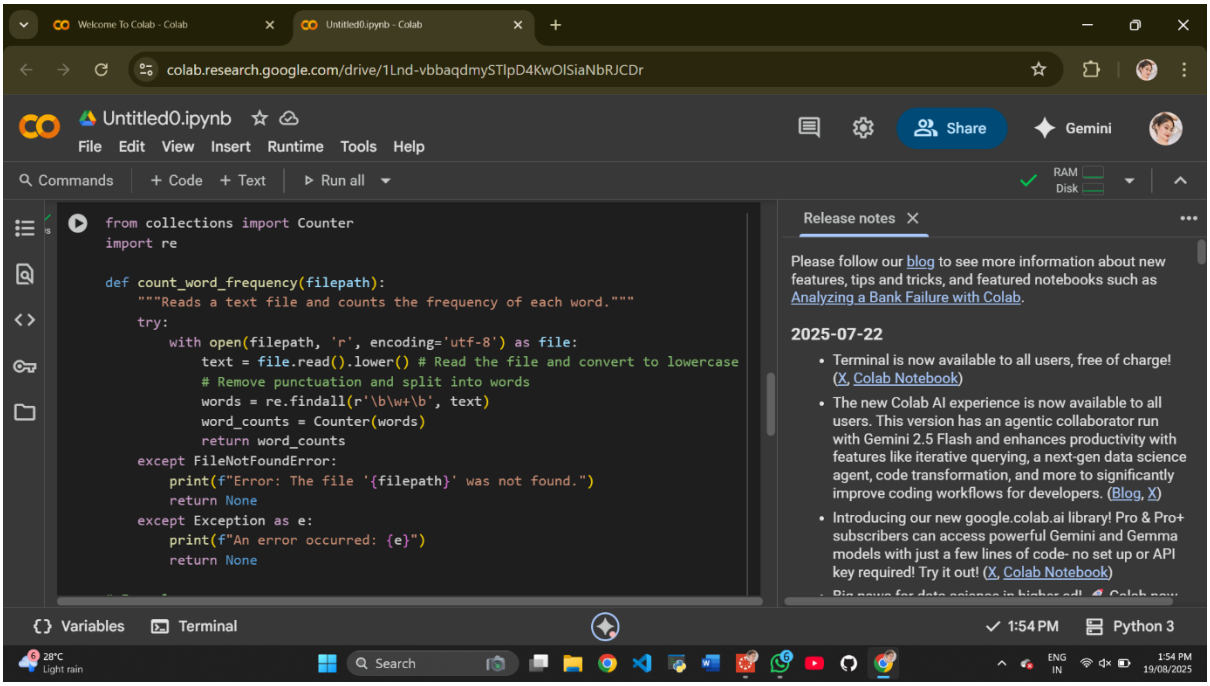
TASK DESCRIPTION#2

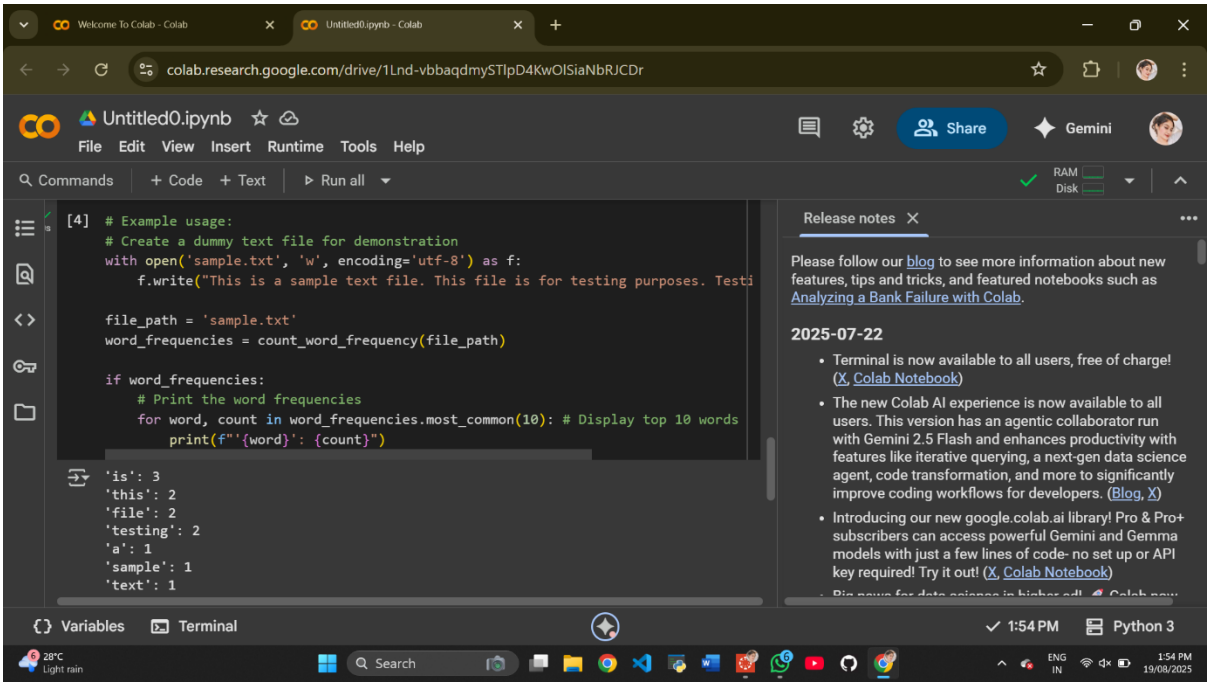
>>use gemini to generate a text file and counts the frequency of each word.

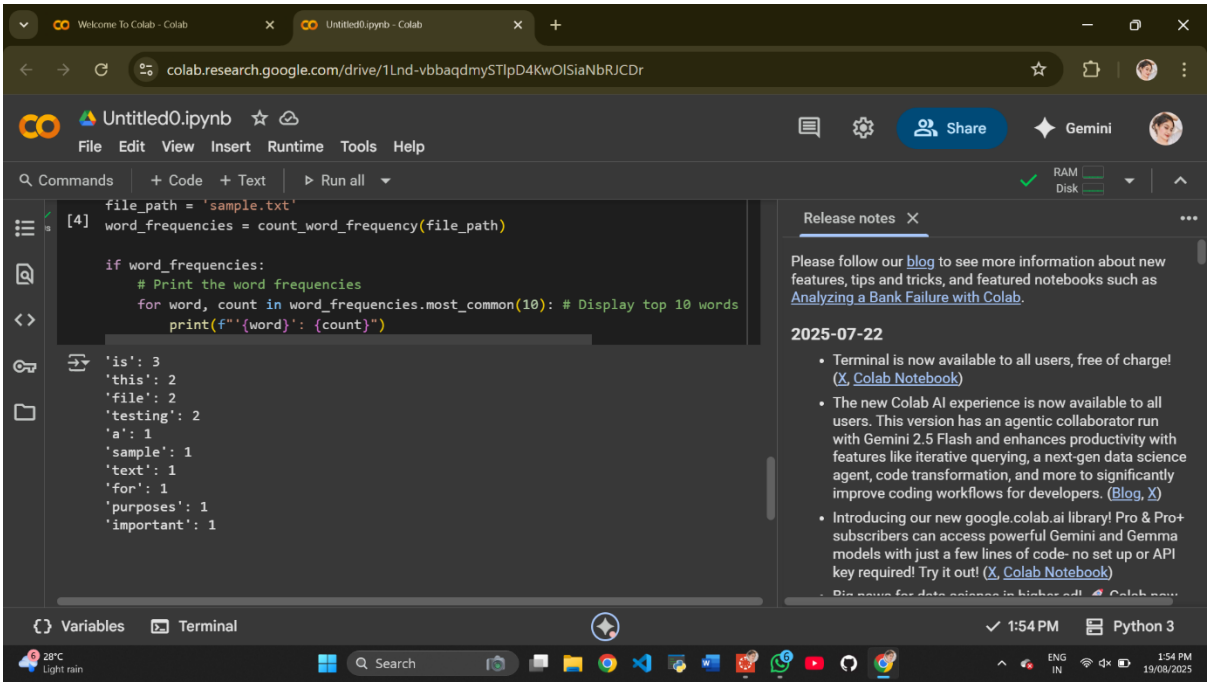
PROMPT :

>>Develop a python program to generate a text file and counts the frequency

EXPECTED OUTPUT AND SCREENSHOT







EXPLANATION :

- **import re**: Imports the regular expression module, which is used for pattern matching in text.
- **from collections import Counter**: Imports the Counter class from the collections module. Counter is a specialized dictionary subclass for counting objects.
- **def count word frequency(file path):**: Defines the function count word frequency that takes one argument, file path.
- **Word counts = Counter()**: Initializes an empty Counter object to store word frequencies.
- **with open(file _path, 'r', encoding='utf-8') as f:**: Opens the file specified by file path in read mode ('r') with UTF-8 encoding. The with statement ensures the file is automatically closed even if errors occur.
- **for line in f:**: Iterates through each line in the file.
- **words = re. find all(r'\b\w+\b', line .lower()**: This line performs two actions:
    - line.lower(): Converts the current line to lowercase to ensure that words like "The" and "the" are counted as the same word.
    - re. find all (r'\b\w+\b', ...): Uses a regular expression to find all words in the lowercase line. \b matches word boundaries, and \w+ matches one or more alphanumeric characters (letters, numbers, and underscore). This effectively extracts words while ignoring punctuation.

- **Word counts. update(words)**: Updates the word counts Counter with the words found in the current line. update() is a method of the Counter class that adds the counts of the elements in the iterable to the existing counts in the Counter.
- **return word counts**: Returns the word counts dictionary, which contains the frequency of each word.
- **File path = 'your_text_file.txt'**: Sets the file path variable to a placeholder filename. **You need to replace 'your_text_file.txt' with the actual path to your text file.**
- **try...except File Not Found Error...except Exception as e:**: This is a try-except block to handle potential errors:
    - It attempts to call the count word frequency function and print the top 10 most common words using word frequencies. Most common(10).
    - If a File not found error occurs (meaning the specified file doesn't exist), it prints an error message.
    - If any other exception occurs, it prints a generic error message along with the exception details.

  In summary, this code provides a robust way to read a text file, clean the text by converting to lowercase and removing punctuation, and then count the frequency of each word using the efficient Counter class.

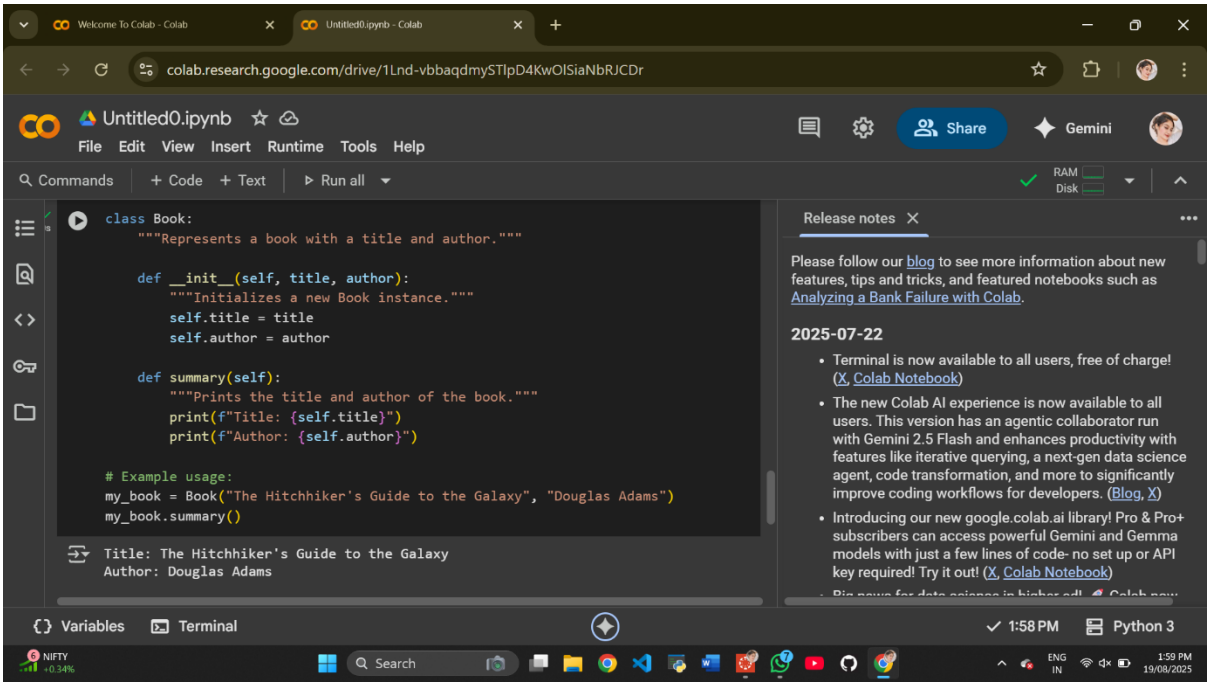
TASK DESCRIPTION#3

>>use gemini to generate a python class called book with attributes title,author,and a method summary

PROMPT:

>>develop a python program to generate a python class called book with attributes title,author,and a method summary

EXPECTED OUTPUT AND SCREENSHOT:



EXPLANATION :

- **class Book:** This line starts the definition of a new class named Book. Classes are blueprints for creating objects (instances).
- **"""Represents a book with title and author."""**: This is a docstring, which explains what the class does.
- **def __in it__(self, title, author):** This is the constructor method.
    - __in it__ is a special method that is automatically called when you create a new object of the class.
    - self refers to the instance of the class being created.
    - title and author are parameters that you pass when creating a Book object.
    - Inside the constructor, self. title = title and self.author = author assign the values passed as arguments to the title and author attributes of the object.
- **def summary(self):** This defines a method named summary for the Book class.
    - self again refers to the instance of the class.
    - This method prints the title and author of the book using f-strings for formatted output.

- **# Example usage:** This is a commented-out example of how to create a Book object and call its summary method. You can uncomment these lines and run the cell to see the class in action.

  In essence, this class allows you to create Book objects, each having a title and an author, and a summary method to display these details.

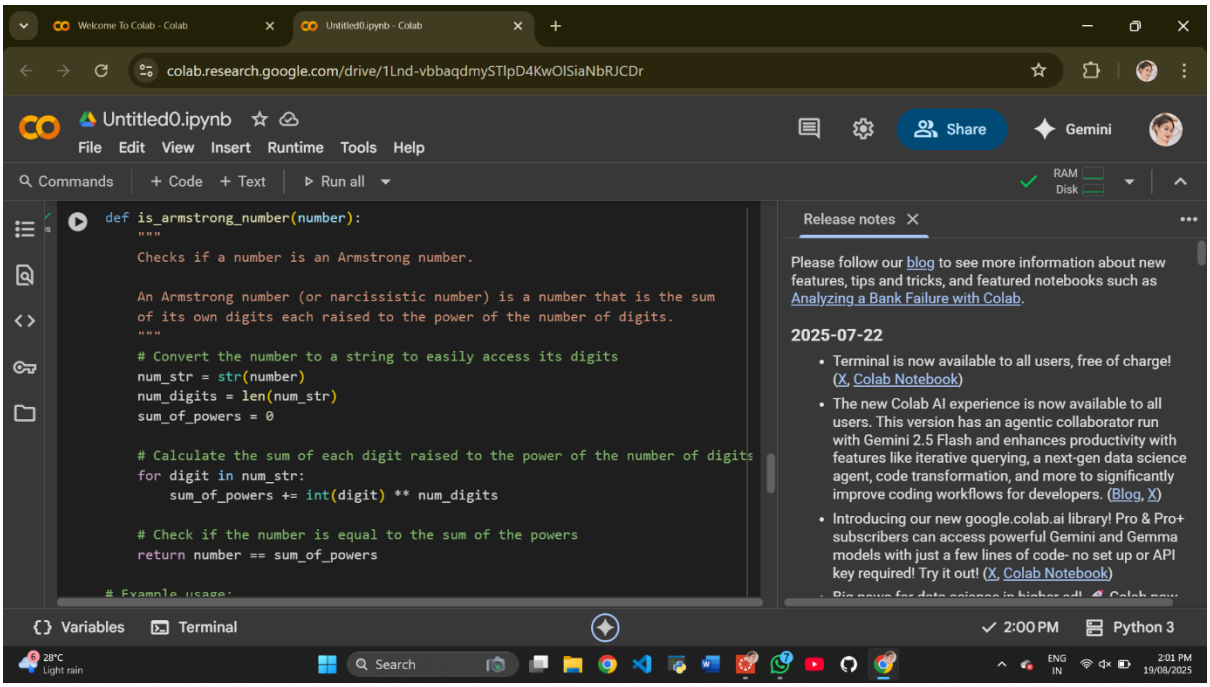
TASK DESCRIPTION#4

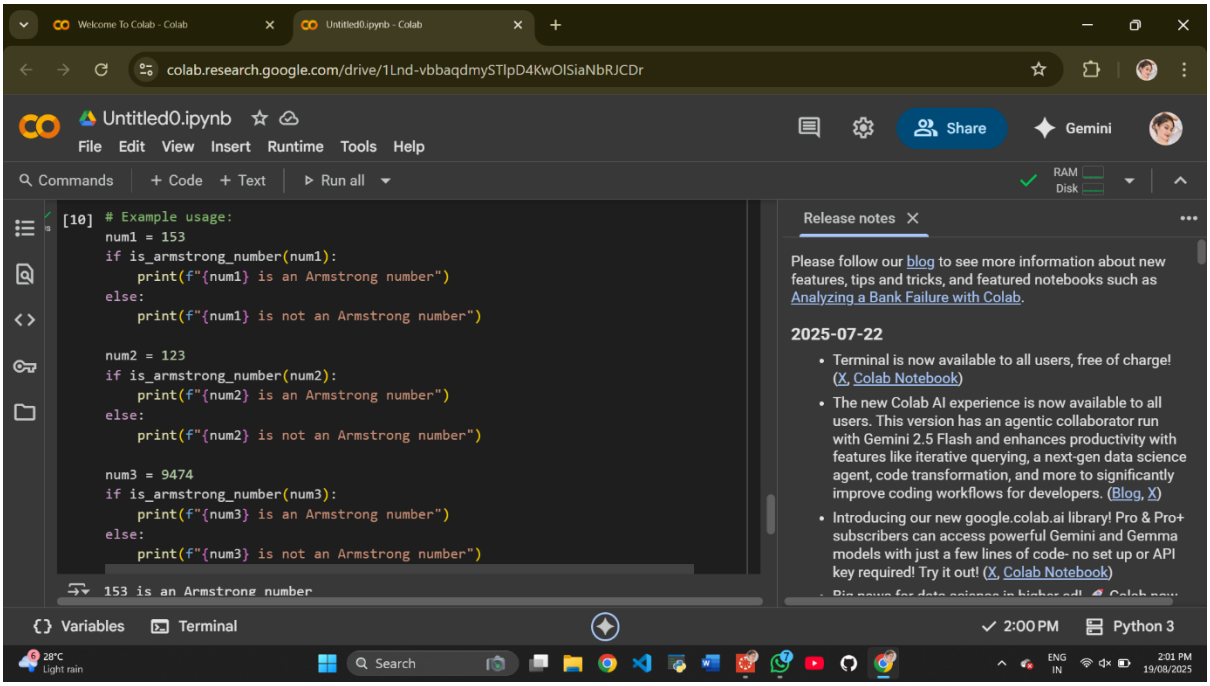
>>use gemini to generate a program to check whether a number is Armstrong or not

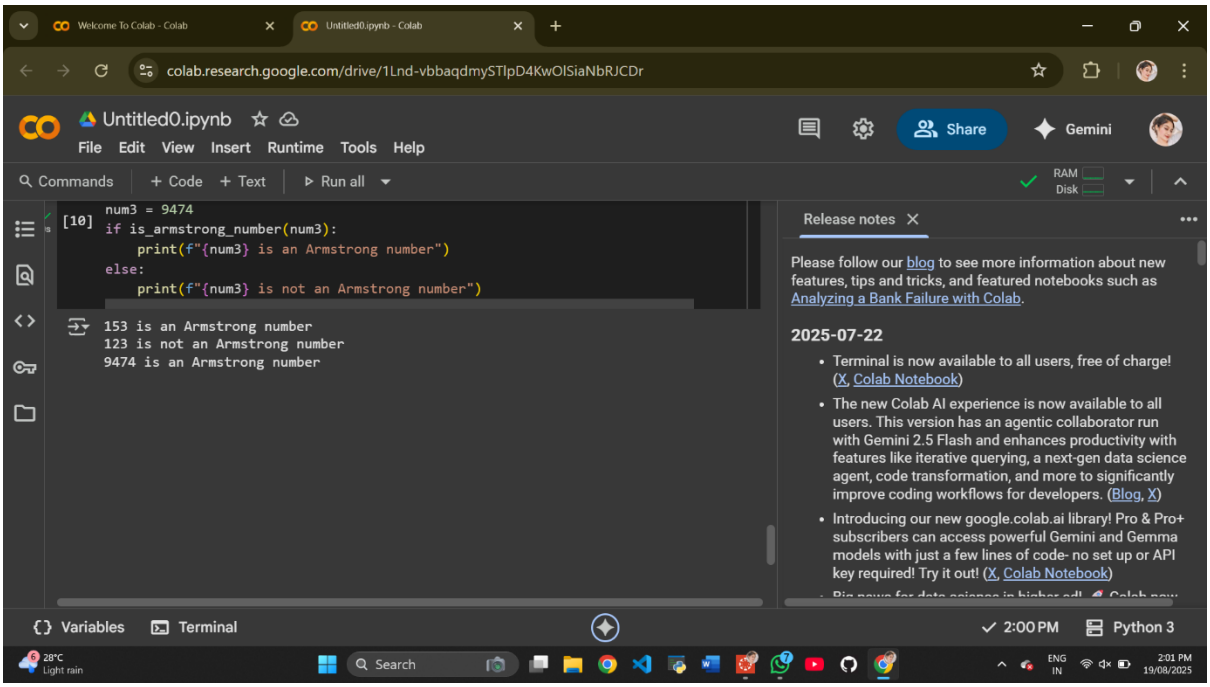
PROMPT :

>>Develop a python program to check whether a number is Armstrong or not

EXPECTED OUTCOME AND SCREENSHOT :







EXPLANATION :

- **def is Armstrong number(number):**: This line defines the function is Armstrong number that takes one argument, number.
- **"""Checks if a number is an Armstrong number..."""**: This is a docstring explaining what the function does, its arguments, and what it returns.

- **Num str = str(number)**: Converts the input number to a string. This makes it easy to iterate through the digits and count how many there are.
- **Num digits = len(num str)**: Calculates the number of digits in the number by getting the length of the string representation.
- **Sum of powers = 0**: Initializes a variable sum_of_powers to 0. This variable will store the sum of each digit raised to the power of the number of digits.
- **for digit in num str:**: This loop iterates through each character (digit) in the num str string.
- **Sum of powers += int(digit) ** num digits**: Inside the loop:
    - int(digit): Converts the current digit (which is a character) back to an integer.
    - ** num digits: Raises the integer value of the digit to the power of num digits.
    - Sum of powers += ...: Adds the result to the sum of powers variable.
- **return sum of powers == number**: After the loop finishes, this line checks if the calculated sum of powers is equal to the original input number. If they are equal, the function returns True (it's an Armstrong number); otherwise, it returns False.
- **# Example usage:**: These commented-out lines show how to use the function with example numbers (153 and 123) and print the results. You can uncomment and run them to test the function.

  In essence, the code converts the number to a string, calculates the sum of each digit raised to the power of the total number of digits, and then compares this sum to the original number to determine if it's an Armstrong number.

  What would you like to do next? We can proceed with step 3 of the plan: modifying the program to improve performance or structure.

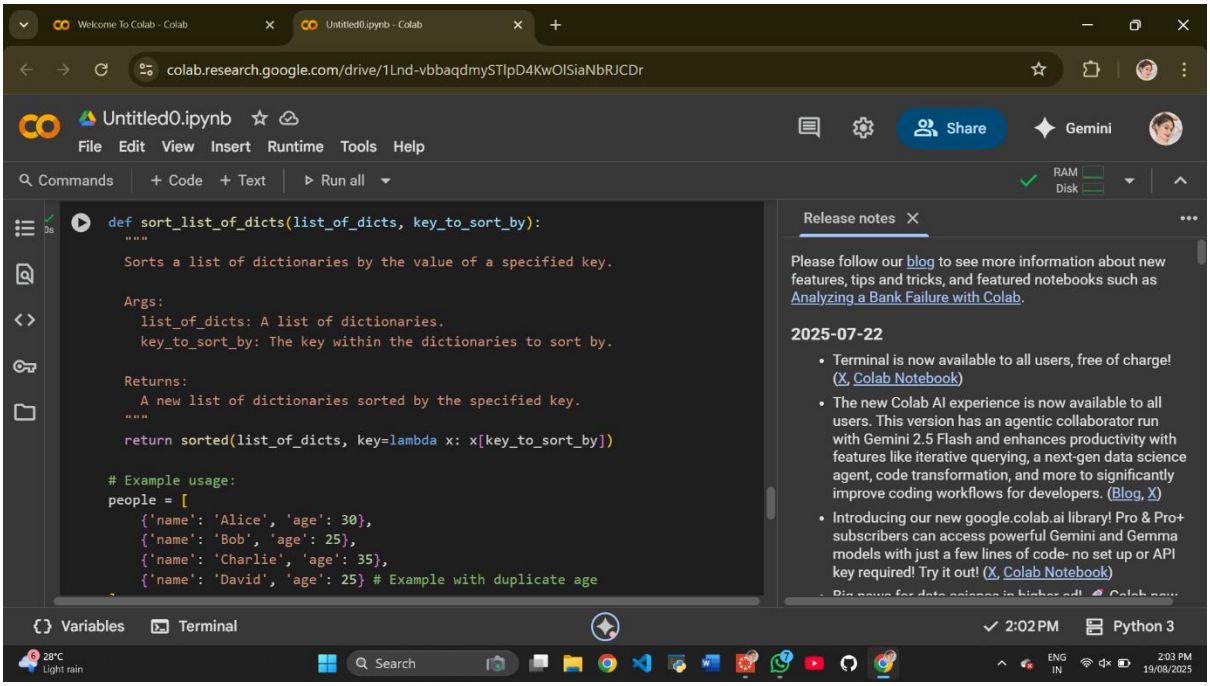
TASK DESCRIPTION#5

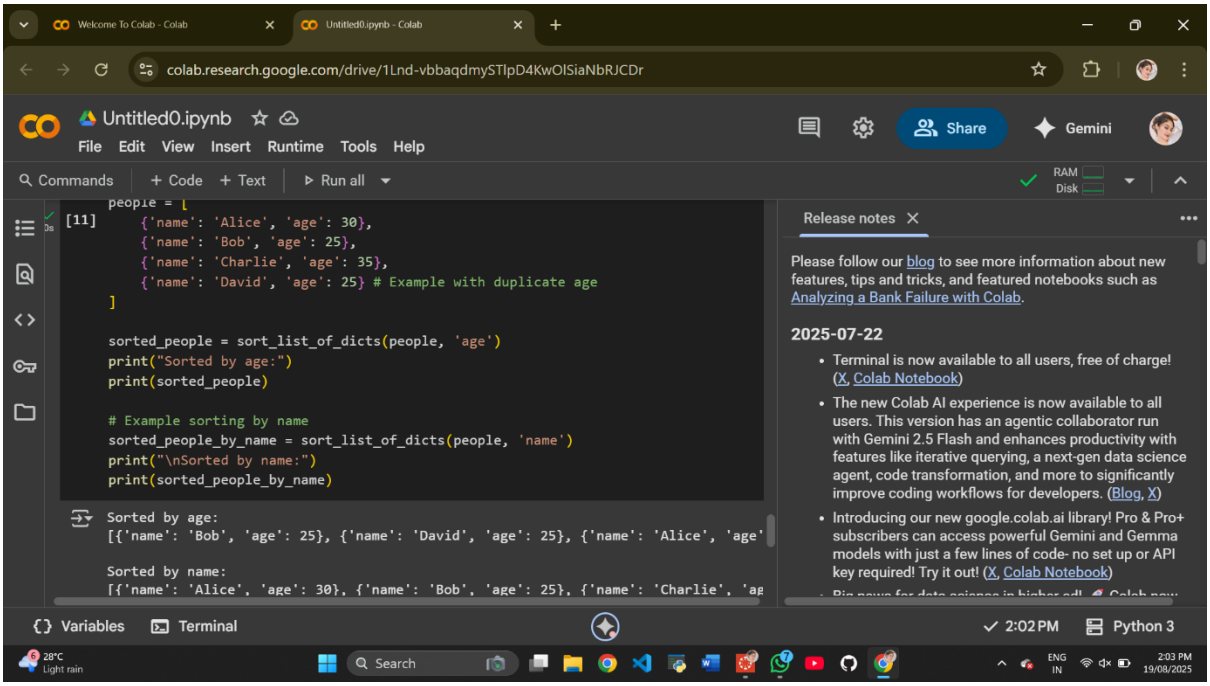
>>use gemini to generate a code for sorting a list of dictionary by a specific key

PROMPT:

>>Develop a python program to generate a code for sorting a list of dictionary by a specific key

EXPECTED OUTPUT AND SCREENSHOT:





EXPLANATION :

This code demonstrates how to sort a list of dictionaries in Python based on the value of a specific key within each dictionary.

Here's an explanation:

- **data = [...]**: This line initializes a list named data. This list contains several dictionaries, where each dictionary represents a person with 'name', 'age', and 'city' information.

- **Sorted data = sorted(data, key=lambda x: x['age'])**: This is the core line for sorting.
    - **sorted()**: This is a built-in Python function that returns a new sorted list from the items in an iterable. It doesn't modify the original list.
    - **data**: This is the list of dictionaries that we want to sort.
    - **key=lambda x: x['age']**: This is the crucial part that tells sorted() *how* to sort the list.
        - **key=**: The key argument specifies a function to be called on each list element prior to making comparisons. The result of this function is used for sorting.
        - **lambda x: x['age']**: This is a small, anonymous function (a lambda function).
            - lambda x:: Defines a lambda function that takes one argument, x. In this context, x will represent each dictionary in the data list as sorted() iterates through it.
            - x['age']: This part of the lambda function accesses the value associated with the key 'age' within the dictionary x.
        - So, the key argument tells sorted() to use the value of the 'age' key in each dictionary as the basis for sorting. The dictionaries will be sorted in ascending order based on their 'age' values.
- **print("Sorted by age:")** and **print(sorted data)**: These lines print a descriptive label and then the sorted data list, which is now sorted by age.
- **Sorted data by name = sorted(data, key=lambda x: x['name'])**: This is another example showing how to sort by a different key, in this case, the 'name' key. The logic is the same as sorting by 'age', but the lambda function now accesses x['name'].
- **print("\nSorted by name:")** and **print(sorted data by name)**: These lines print a label and the list sorted by name. The \n in the first print statement adds a newline for better formatting.

  In summary, the sorted() function combined with a lambda function and the key argument provides a concise and powerful way to sort lists of dictionaries based on the values of specific keys.

  0 / 2000

  Gemini can make mistakes so double-check it and use code with caution. Learn more