AI-ASSISTED Coding Lab
Assignment 2
Enrollment No: 2503A51L144
Batch: 20
Student Name: Meer Burhan Ali Hashmi

Task 1

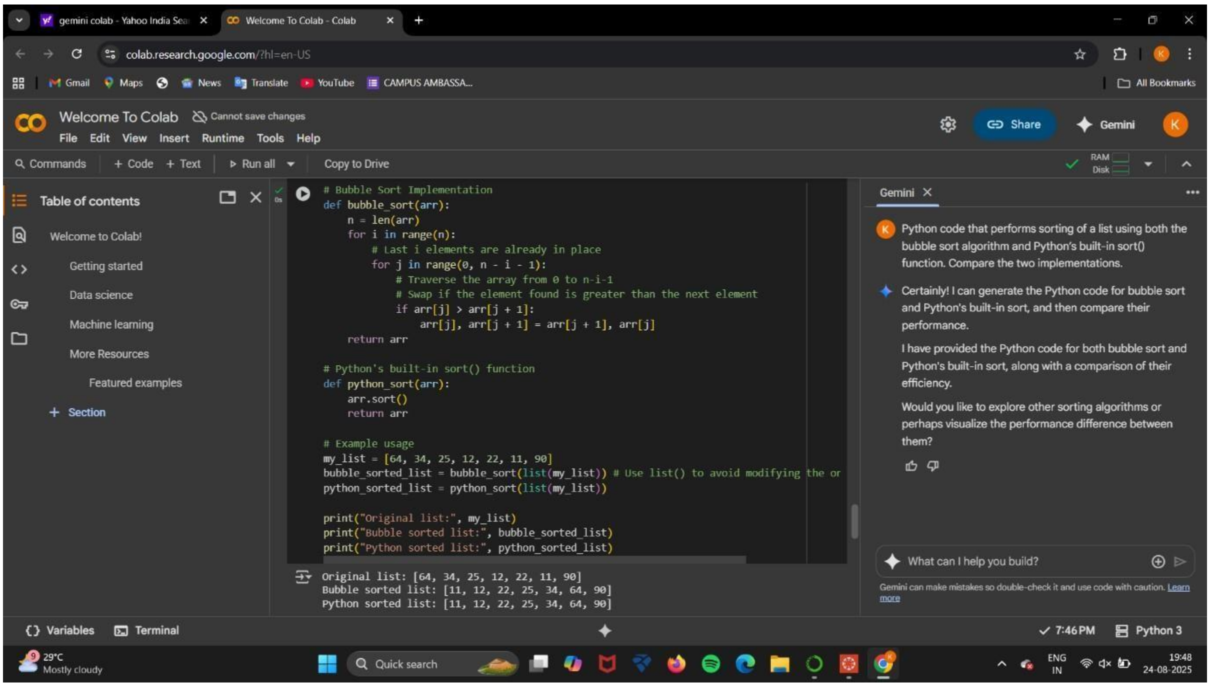
Task Overview:
Launch Google Colab and utilize the Google Gemini model to create a Python program that sorts a list using two different approaches:

1. **Bubble Sort algorithm (manual implementation).**
2. **Python's built-in sort() method.**

After implementing both, analyze and compare their performance and approach.

Prompt Used:
"Write a Python program to sort a list using both the bubble sort method and Python's built-in sort() function, then compare the results and efficiency."

CODE:-



```
# 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):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# Python's built-in sort() function
def python_sort(arr):
    arr.sort()
    return arr

# Example usage
my_list = [64, 34, 25, 12, 22, 11, 90]
bubble_sorted_list = bubble_sort(list(my_list)) # Use list() to avoid modifying the or
python_sorted_list = python_sort(list(my_list))

print("Original list:", my_list)
print("Bubble sorted list:", bubble_sorted_list)
print("Python sorted list:", python_sorted_list)
```

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Bubble sorted list: [11, 12, 22, 25, 34, 64, 90]
Python sorted list: [11, 12, 22, 25, 34, 64, 90]
```

Python code that performs sorting of a list using both the bubble sort algorithm and Python's built-in sort() function. Compare the two implementations.

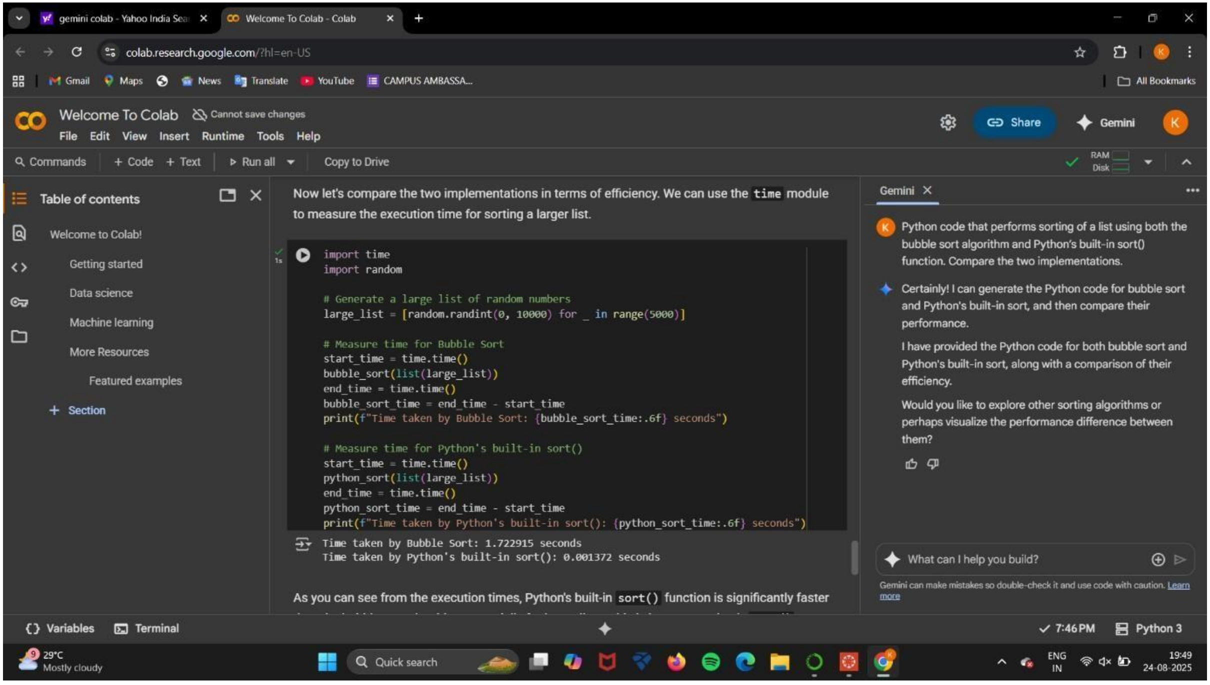
Certainly! I can generate the Python code for bubble sort and Python's built-in sort, and then compare their performance.

I have provided the Python code for both bubble sort and Python's built-in sort, along with a comparison of their efficiency.

Would you like to explore other sorting algorithms or perhaps visualize the performance difference between them?

What can I help you build?

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



Now let's compare the two implementations in terms of efficiency. We can use the `time` module to measure the execution time for sorting a larger list.

```
import time
import random

# Generate a large list of random numbers
large_list = [random.randint(0, 10000) for _ in range(5000)]

# Measure time for Bubble Sort
start_time = time.time()
bubble_sort(list(large_list))
end_time = time.time()
bubble_sort_time = end_time - start_time
print(f"Time taken by Bubble Sort: {bubble_sort_time:.6f} seconds")

# Measure time for Python's built-in sort()
start_time = time.time()
python_sort(list(large_list))
end_time = time.time()
python_sort_time = end_time - start_time
print(f"Time taken by Python's built-in sort(): {python_sort_time:.6f} seconds")
```

```
Time taken by Bubble Sort: 1.722915 seconds
Time taken by Python's built-in sort(): 0.001372 seconds
```

As you can see from the execution times, Python's built-in `sort()` function is significantly faster

Python code that performs sorting of a list using both the bubble sort algorithm and Python's built-in sort() function. Compare the two implementations.

Certainly! I can generate the Python code for bubble sort and Python's built-in sort, and then compare their performance.

I have provided the Python code for both bubble sort and Python's built-in sort, along with a comparison of their efficiency.

Would you like to explore other sorting algorithms or perhaps visualize the performance difference between them?

What can I help you build?

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

Task 2

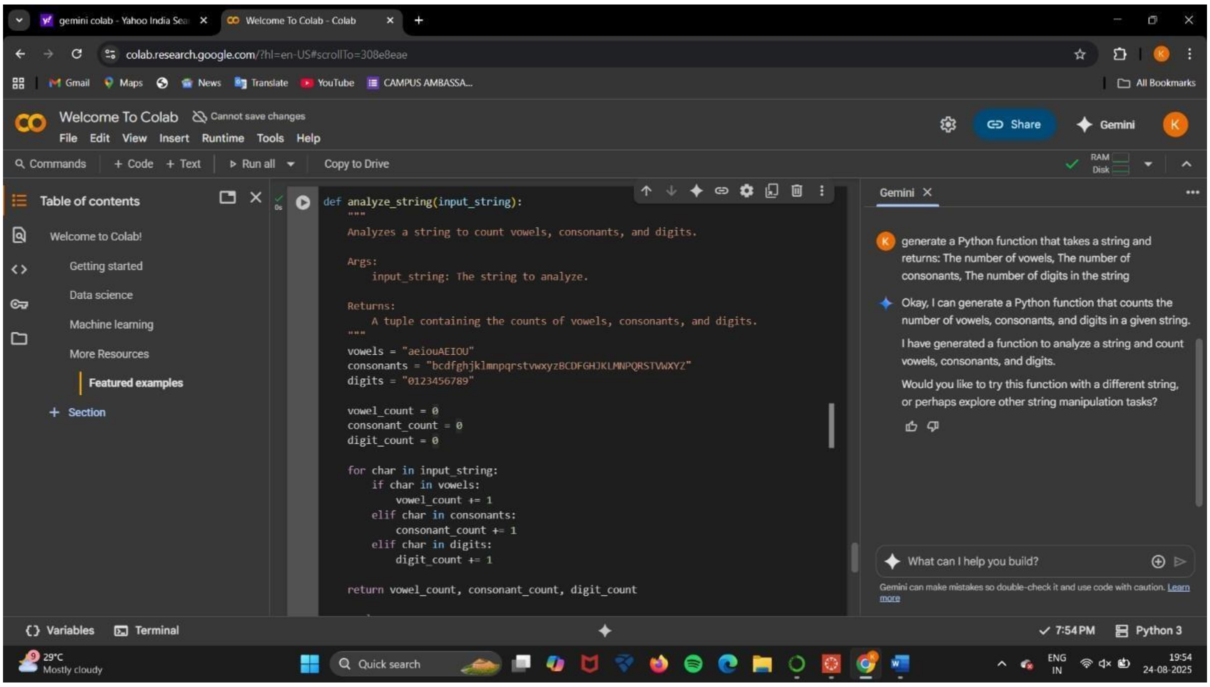
Overview:

Within Google Colab, make use of Google Gemini to create a Python function that accepts a text string as input and returns:

- **The total count of vowels.**
- **The total count of consonants.**
- **The total count of numeric characters.**

Prompt Provided to Gemini:
"Create a Python function that receives a string as input and outputs three values — the count of vowels, the count of consonants, and the count of digits present in the given string."

CODE



```python
def analyze_string(input_string):
    """
    Analyzes a string to count vowels, consonants, and digits.

    Args:
        input_string: The string to analyze.

    Returns:
        A tuple containing the counts of vowels, consonants, and digits.
    """
    vowels = "aeiouAEIOU"
    consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    digits = "0123456789"

    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        if char in vowels:
            vowel_count += 1
        elif char in consonants:
            consonant_count += 1
        elif char in digits:
            digit_count += 1

    return vowel_count, consonant_count, digit_count
```

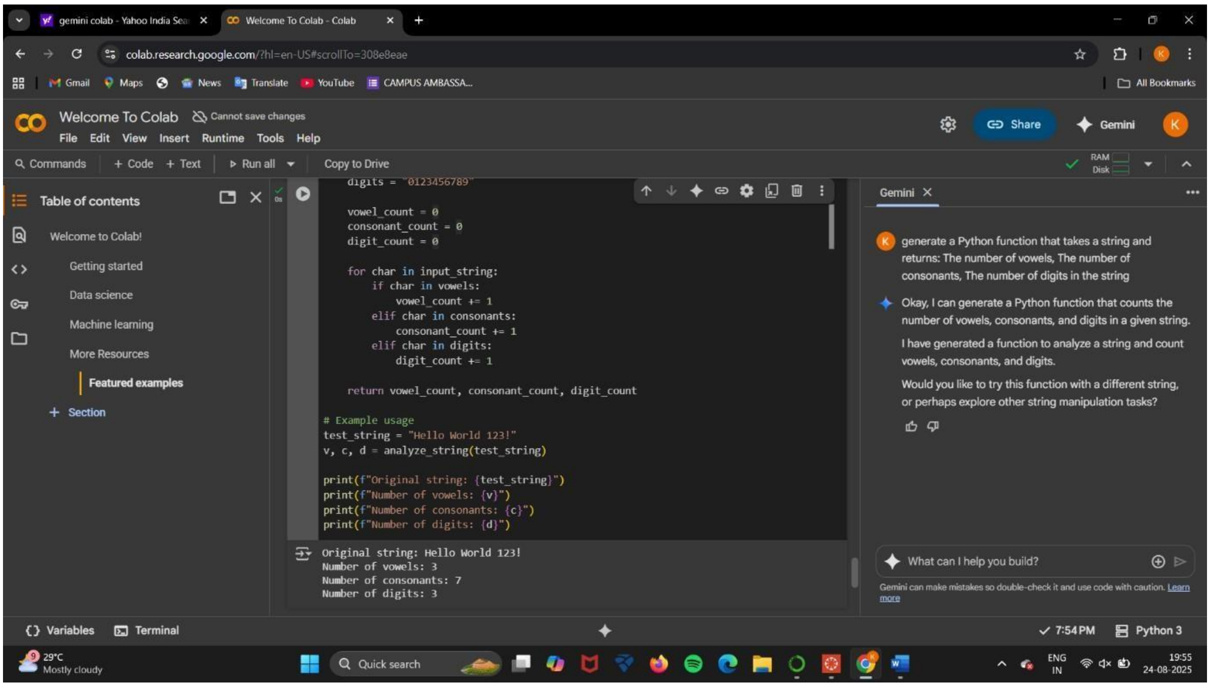
generate a Python function that takes a string and returns: The number of vowels, The number of consonants, The number of digits in the string

Okay, I can generate a Python function that counts the number of vowels, consonants, and digits in a given string.

I have generated a function to analyze a string and count vowels, consonants, and digits.

Would you like to try this function with a different string, or perhaps explore other string manipulation tasks?

What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more


Welcome To Colab
Cannot save changes
Table of contents
Share
Gemini
7:54PM
Python 3

```python
    digits = "0123456789"

    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        if char in vowels:
            vowel_count += 1
        elif char in consonants:
            consonant_count += 1
        elif char in digits:
            digit_count += 1

    return vowel_count, consonant_count, digit_count

# Example usage
test_string = "Hello World 123!"
v, c, d = analyze_string(test_string)

print(f"Original string: {test_string}")
print(f"Number of vowels: {v}")
print(f"Number of consonants: {c}")
print(f"Number of digits: {d}")
```

```
Original string: Hello World 123!
Number of vowels: 3
Number of consonants: 7
Number of digits: 3
```

generate a Python function that takes a string and returns: The number of vowels, The number of consonants, The number of digits in the string

Okay, I can generate a Python function that counts the number of vowels, consonants, and digits in a given string.

I have generated a function to analyze a string and count vowels, consonants, and digits.

Would you like to try this function with a different string, or perhaps explore other string manipulation tasks?

Task 4

Description:

In Google Colab, prompt Google Gemini to produce a Python script for a basic calculator. The program should define separate functions to handle the four primary operations:

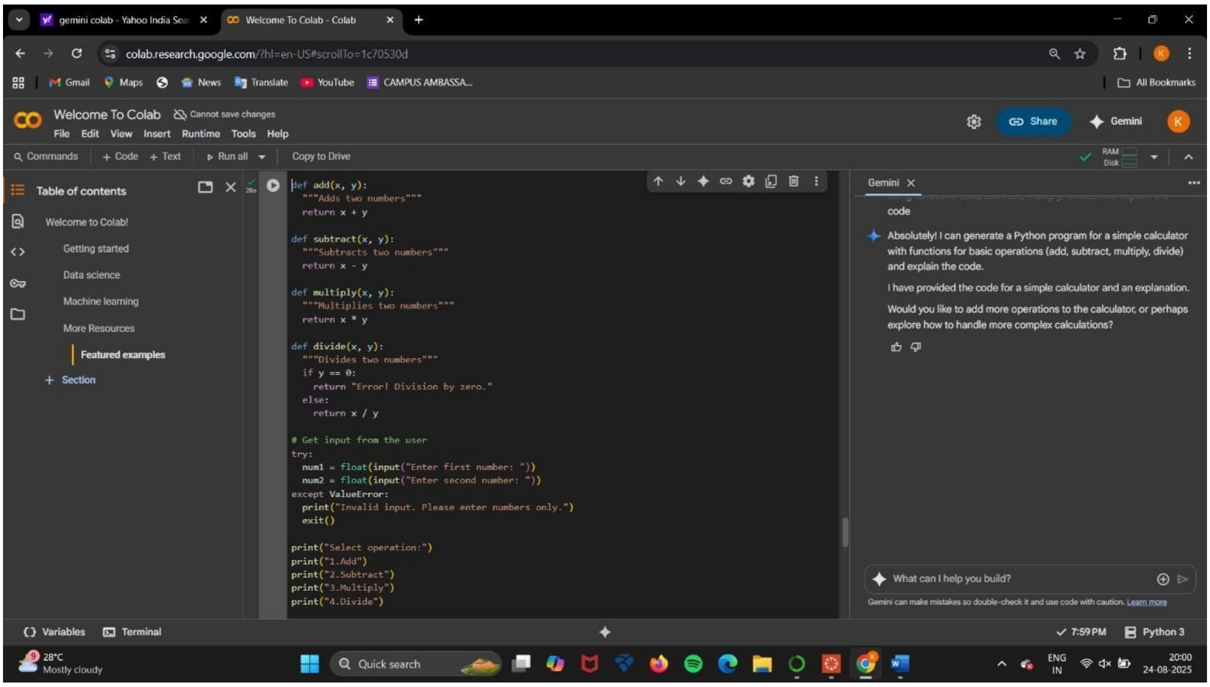
- **Addition**

- **Subtraction**
- **Multiplication**
- **Division**

After generating the program, request Gemini to provide an explanation of how each part of the code works.

Prompt to Gemini:

"Write a Python program that implements a basic calculator with functions for addition, subtraction, multiplication, and division. Then, describe the functionality and workflow of the code."



```python
def add(x, y):
  """Adds two numbers"""
  return x + y

def subtract(x, y):
  """Subtracts two numbers"""
  return x - y

def multiply(x, y):
  """Multiplies two numbers"""
  return x * y

def divide(x, y):
  """Divides two numbers"""
  if y == 0:
    return "Error! Division by zero."
  else:
    return x / y

# Get input from the user
try:
  num1 = float(input("Enter first number: "))
  num2 = float(input("Enter second number: "))
except ValueError:
  print("Invalid input. Please enter numbers only.")
  exit()

print("Select operation:")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
```

Gemini

code

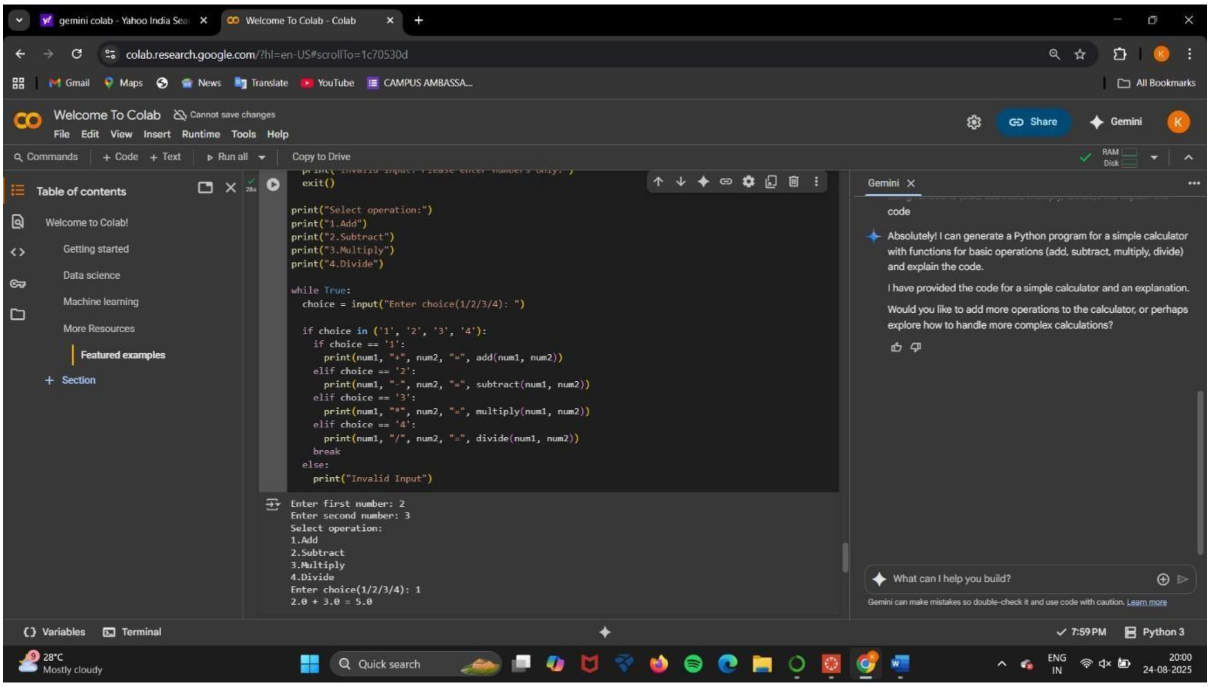
Absolutely! I can generate a Python program for a simple calculator with functions for basic operations (add, subtract, multiply, divide) and explain the code.

I have provided the code for a simple calculator and an explanation.

Would you like to add more operations to the calculator, or perhaps explore how to handle more complex calculations?

What can I help you build?

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



```
  exit()

print("Select operation:")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
  choice = input("Enter choice(1/2/3/4): ")

  if choice in ('1', '2', '3', '4'):
    if choice == '1':
      print(num1, "+", num2, "=", add(num1, num2))
    elif choice == '2':
      print(num1, "-", num2, "=", subtract(num1, num2))
    elif choice == '3':
      print(num1, "*", num2, "=", multiply(num1, num2))
    elif choice == '4':
      print(num1, "/", num2, "=", divide(num1, num2))
    break
  else:
    print("Invalid Input")
```

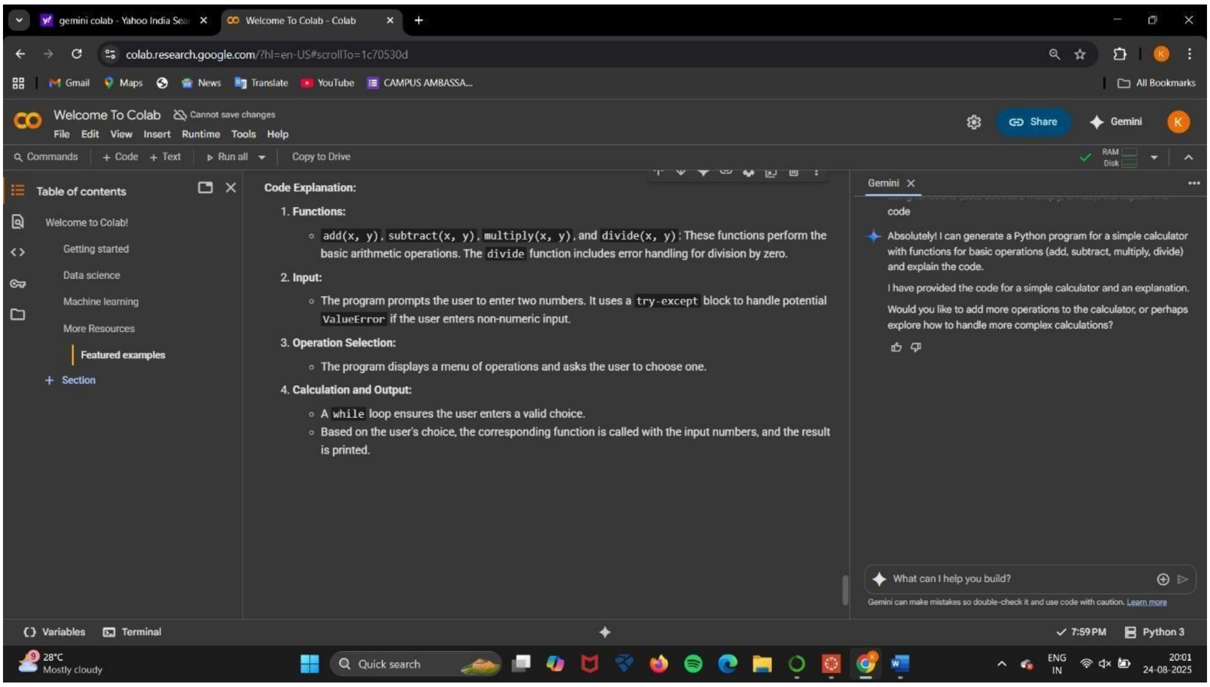
```
Enter first number: 2
Enter second number: 3
Select operation:
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 1
2.0 + 3.0 = 5.0
```

code

Absolutely! I can generate a Python program for a simple calculator with functions for basic operations (add, subtract, multiply, divide) and explain the code.

I have provided the code for a simple calculator and an explanation.

Would you like to add more operations to the calculator, or perhaps explore how to handle more complex calculations?



Code Explanation:

1. **Functions:**
    - `add(x, y)`, `subtract(x, y)`, `multiply(x, y)`, and `divide(x, y)`: These functions perform the basic arithmetic operations. The `divide` function includes error handling for division by zero.
2. **Input:**
    - The program prompts the user to enter two numbers. It uses a `try-except` block to handle potential `ValueError` if the user enters non-numeric input.
3. **Operation Selection:**
    - The program displays a menu of operations and asks the user to choose one.
4. **Calculation and Output:**
    - A `while` loop ensures the user enters a valid choice.
    - Based on the user's choice, the corresponding function is called with the input numbers, and the result is printed.

code

Absolutely! I can generate a Python program for a simple calculator with functions for basic operations (add, subtract, multiply, divide) and explain the code.

I have provided the code for a simple calculator and an explanation.

Would you like to add more operations to the calculator, or perhaps explore how to handle more complex calculations?

Through this assignment, I explored how Google Gemini can produce Python programs when given precise and well-structured prompts. I also noticed how it can be used to compare and evaluate various problem-solving techniques.

- **Task 1: Gemini provided two sorting solutions — one implemented manually using the Bubble Sort algorithm and another using Python's built-in sort() method. This comparison showed the contrast between a detailed, step-by-step algorithm and a pre-optimized built-in function. While manual algorithms are valuable for**

understanding concepts, built-in methods are generally more efficient for realworld applications.

- **Task 2: Gemini created a function capable of analyzing a string and returning the number of vowels, consonants, and digits. This illustrated its proficiency in handling string operations and applying conditional logic to achieve accurate results.**
- **Task 4: Gemini developed a simple calculator program that used separate functions for addition, subtraction, multiplication, and division. Furthermore, when asked for an explanation, it provided a clear breakdown of how the code works, demonstrating that AI can be an effective tool for both generating solutions and explaining programming concepts.**