TASK-1

Task 1: Word Frequency from Text File

❖ Scenario:

You are analyzing log files for keyword frequency.

❖ Task:

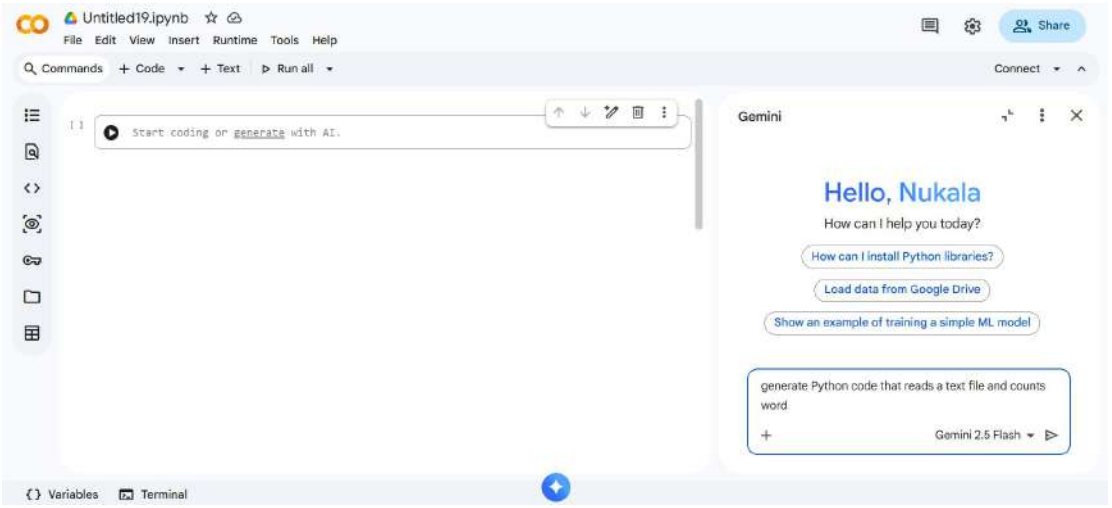
Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code.

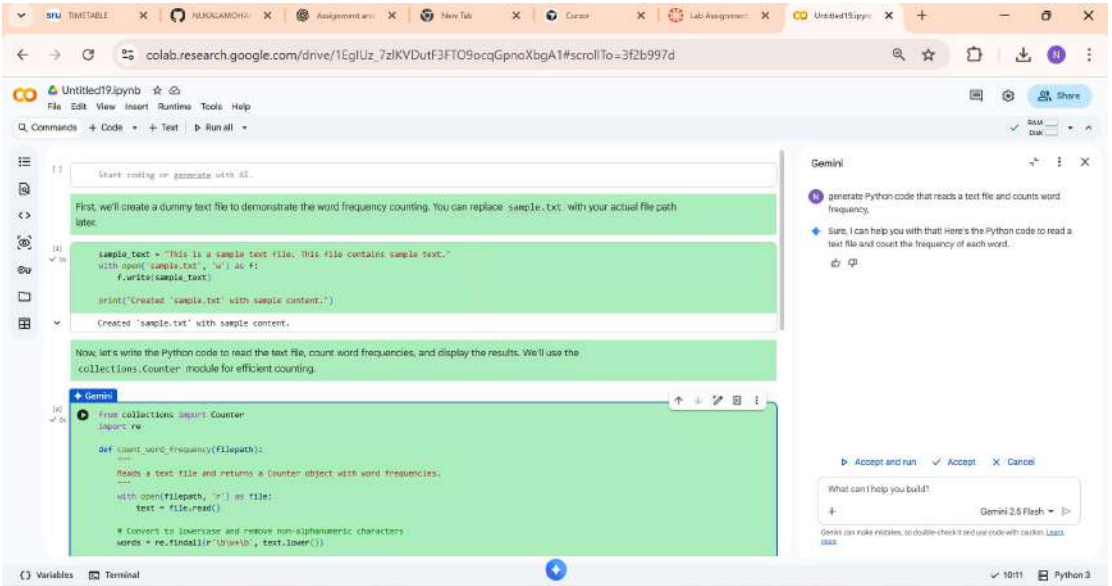
❖ Expected Output:

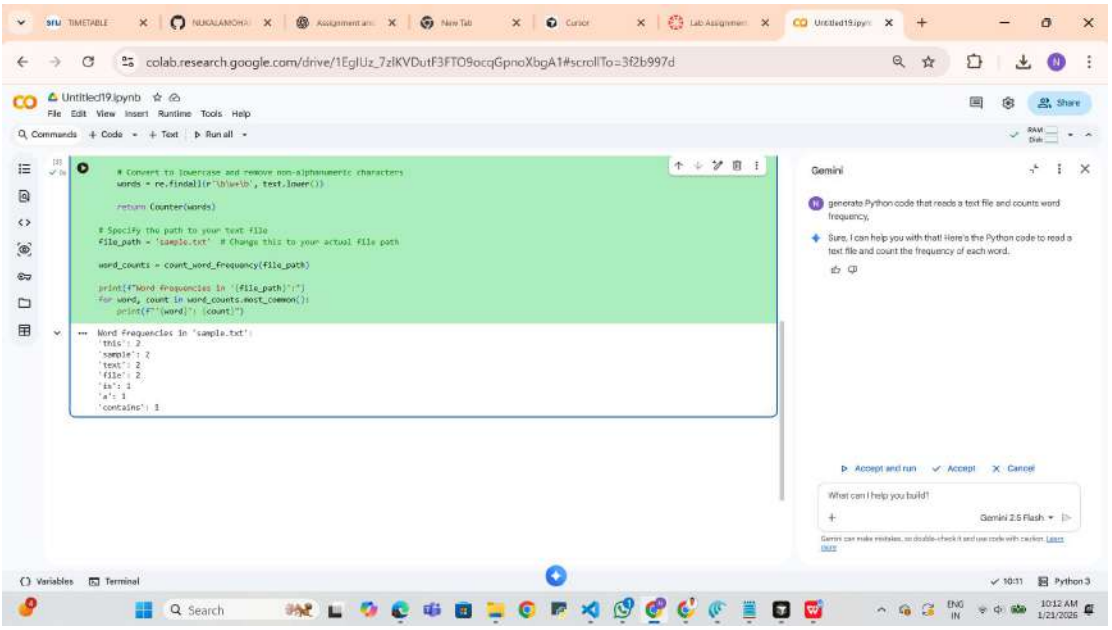
➢ Working code

➢ Explanation

➢ Screenshot







The code defines a function to read a text file, sample.txt, for processing.
It converts the entire text to lowercase and extracts all alphanumeric words using regular expressions.
The collections.Counter module is then used to efficiently count the frequency of each extracted word.
The function is called with the sample.txt file, which was pre-populated with example text.
Finally, the code iterates through the word counts and prints each word along with its frequency.

TASK-2

❖ Scenario:

You are automating basic file operations.

❖ Task:

Use Cursor AI to generate a program that:

➢ Creates a text file

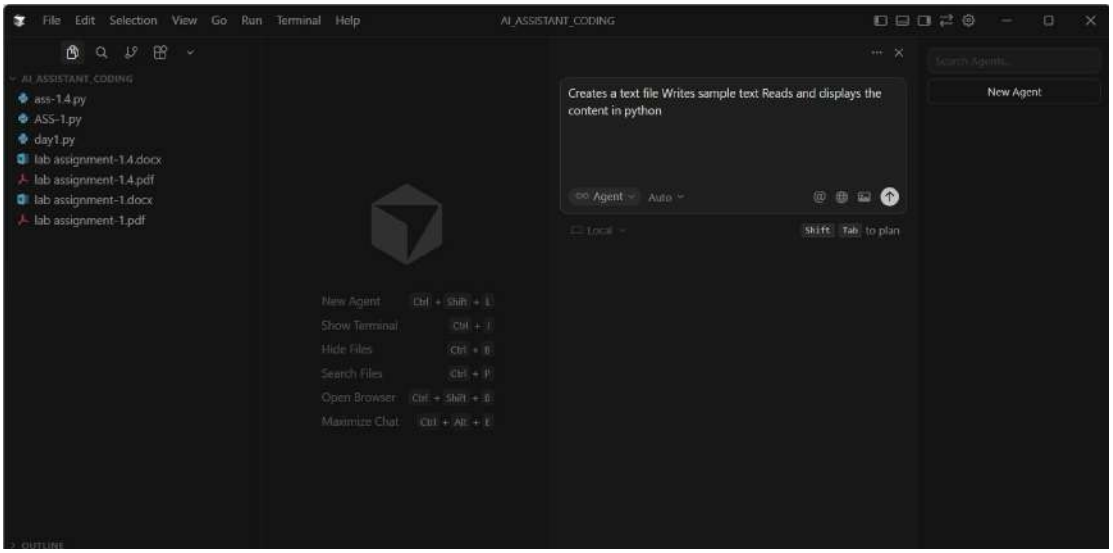
➢ Writes sample text

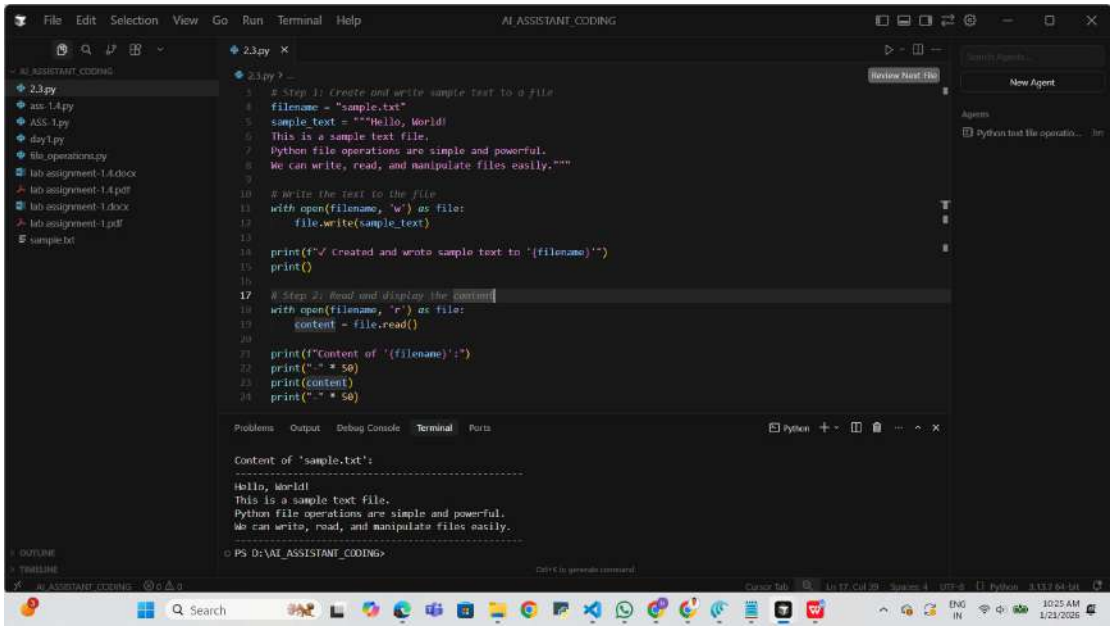
➢ Reads and displays the content

❖ Expected Output:

➢ Functional code

➢ Cursor AI screenshots





Task 3: CSV Data Analysis

❖ Scenario:

You are processing structured data from a CSV file.

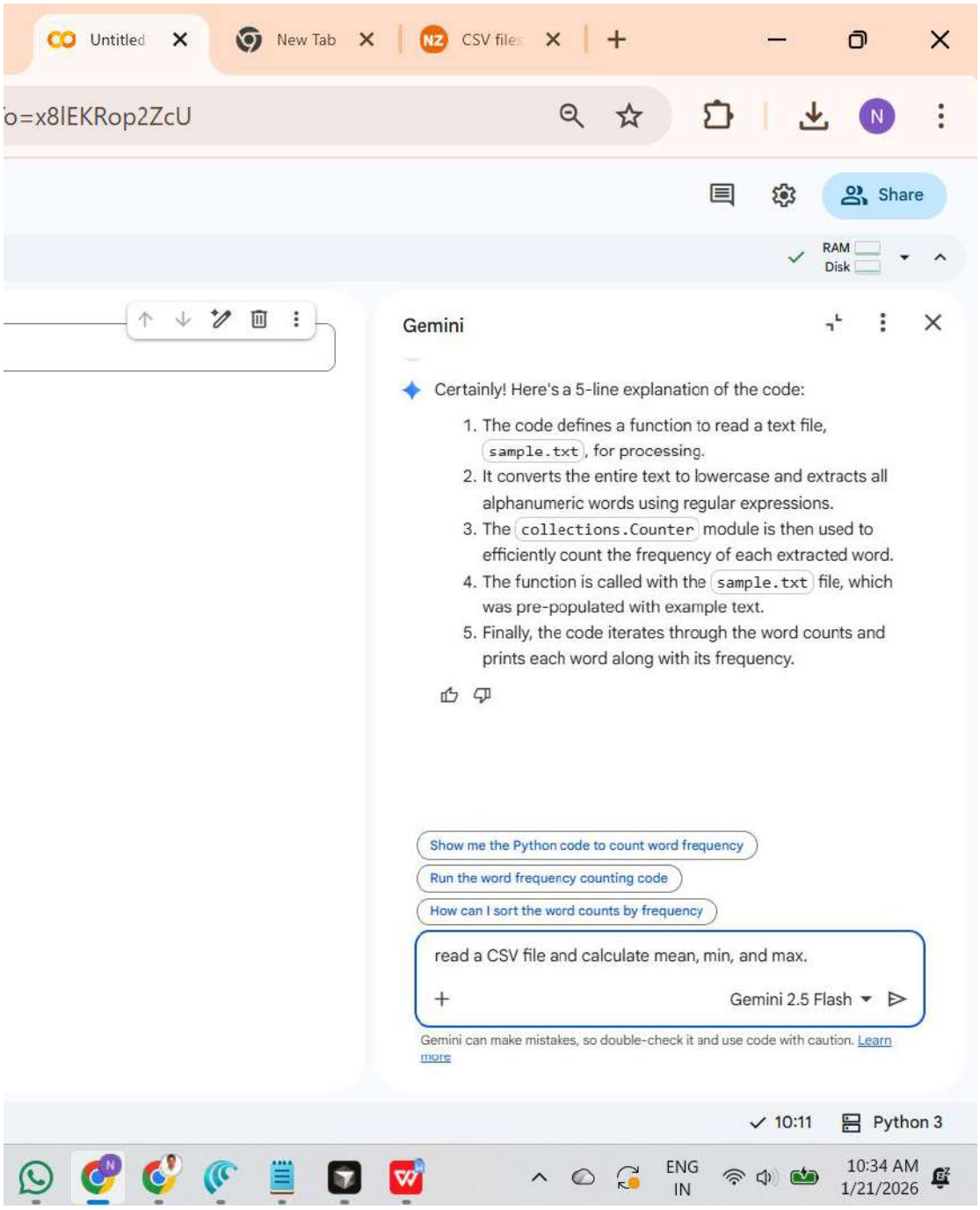
❖ Task:

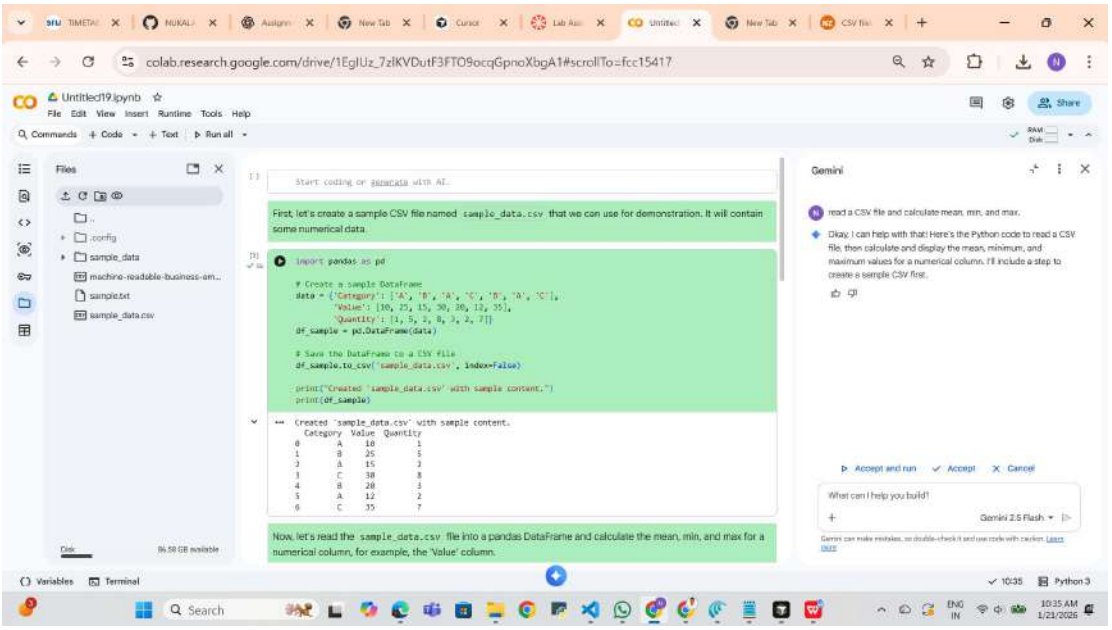
Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

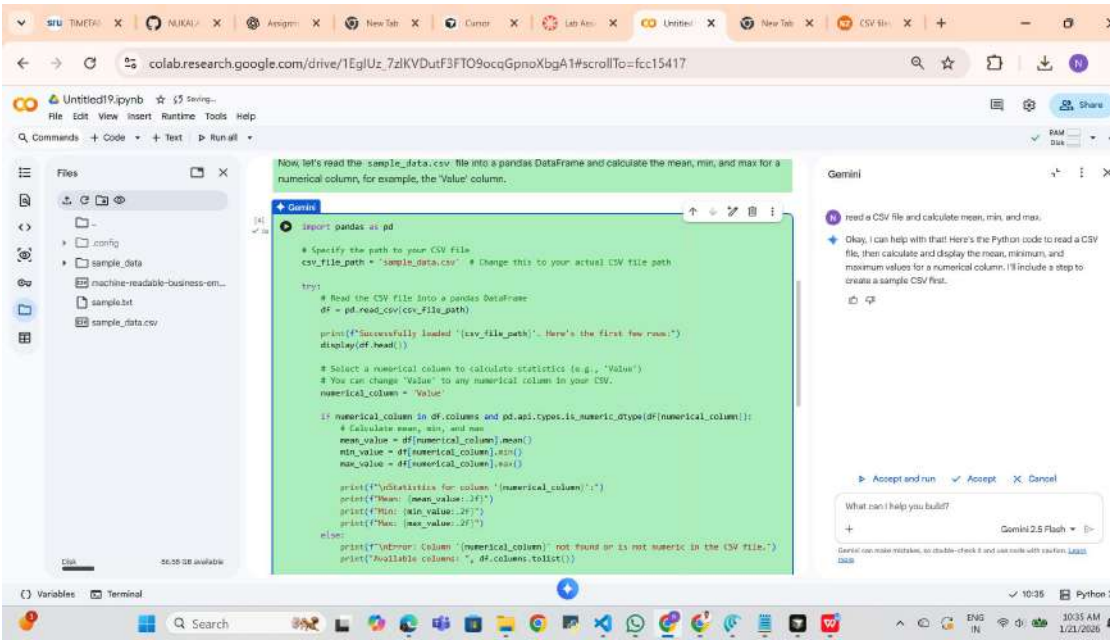
❖ Expected Output:

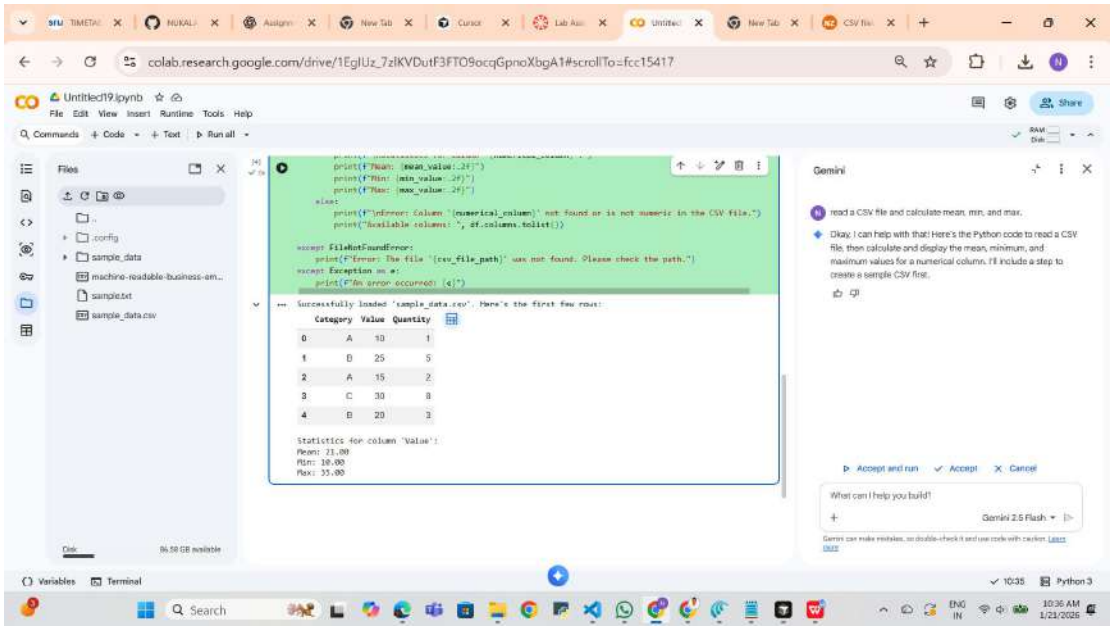
➢ Correct output

➢ Screenshot


Untitled
New Tab
CSV files
o=x8lEKRop2ZcU
Share
RAM
Disk
Gemini
Certainly! Here's a 5-line explanation of the code:
1. The code defines a function to read a text file, sample.txt, for processing.
2. It converts the entire text to lowercase and extracts all alphanumeric words using regular expressions.
3. The collections.Counter module is then used to efficiently count the frequency of each extracted word.
4. The function is called with the sample.txt file, which was pre-populated with example text.
5. Finally, the code iterates through the word counts and prints each word along with its frequency.
Show me the Python code to count word frequency
Run the word frequency counting code
How can I sort the word counts by frequency
read a CSV file and calculate mean, min, and max.
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more
10:11
Python 3
ENG
IN
10:34 AM
1/21/2026


colab.research.google.com/drive/1EglUz_7zlKVDutF3FTO9ocqGpnoXbgA1#scrollTo=fcc15417
Untitled19.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Files
sample_data
machine-readable-business-em...
sample.txt
sample_data.csv
Start coding or generate with AI.
First, let's create a sample CSV file named sample_data.csv that we can use for demonstration. It will contain some numerical data.
import pandas as pd
# Create a sample DataFrame
data = {'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C'],
'Value': [10, 25, 15, 30, 20, 12, 35],
'Quantity': [1, 5, 2, 8, 3, 2, 7]}
df_sample = pd.DataFrame(data)
# Save the DataFrame to a CSV file
df_sample.to_csv('sample_data.csv', index=False)
print("Created 'sample_data.csv' with sample content.")
print(df_sample)
Created 'sample_data.csv' with sample content.
Category Value Quantity
0 A 10 1
1 B 25 5
2 A 15 2
3 C 30 8
4 B 20 3
5 A 12 2
6 C 35 7
Now, let's read the sample_data.csv file into a pandas DataFrame and calculate the mean, min, and max for a numerical column, for example, the 'Value' column.
Gemini
read a CSV file and calculate mean, min, and max.
Okay, I can help with that! Here's the Python code to read a CSV file, then calculate and display the mean, minimum, and maximum values for a numerical column. I'll include a step to create a sample CSV first.
Accept and run
Accept
Cancel
What can I help you build?
Gemini 2.5 Flash
Variables
Terminal
Python 3
Search
10:35 AM
1/21/2026


colab.research.google.com/drive/1EglUz_7zlKVDutF3FTO9ocqGpnoXbgA1#scrollTo=fcc15417
Untitled19.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Files
sample_data
machine-readable-business-em...
sample.txt
sample_data.csv
Now, let's read the sample_data.csv file into a pandas DataFrame and calculate the mean, min, and max for a numerical column, for example, the 'Value' column.
Gemini
import pandas as pd
# Specify the path to your CSV file
csv_file_path = 'sample_data.csv' # Change this to your actual CSV file path
try:
# Read the CSV file into a pandas DataFrame
df = pd.read_csv(csv_file_path)
print(f"Successfully loaded '{csv_file_path}'. Here's the first few rows:")
display(df.head())
# Select a numerical column to calculate statistics (e.g., 'Value')
# You can change 'Value' to any numerical column in your CSV.
numerical_column = 'Value'
if numerical_column in df.columns and pd.api.types.is_numeric_dtype(df[numerical_column]):
# Calculate mean, min, and max
mean_value = df[numerical_column].mean()
min_value = df[numerical_column].min()
max_value = df[numerical_column].max()
print(f"\nStatistics for column '{numerical_column}':")
print(f"Mean: {mean_value:.2f}")
print(f"Min: {min_value:.2f}")
print(f"Max: {max_value:.2f}")
else:
print(f"\nError: Column '{numerical_column}' not found or is not numeric in the CSV file.")
print("Available columns: ", df.columns.tolist())
Gemini
read a CSV file and calculate mean, min, and max.
Okay, I can help with that! Here's the Python code to read a CSV file, then calculate and display the mean, minimum, and maximum values for a numerical column. I'll include a step to create a sample CSV first.
Accept and run
Accept
Cancel
What can I help you build?
Gemini 2.5 Flash
Variables
Terminal
Search
10:35 AM
1/21/2026



Task 4: Sorting Lists – Manual vs Built-in

❖ Scenario:

You are reviewing algorithm choices for efficiency.

❖ Task:

Use Gemini to generate:

➢ Bubble sort

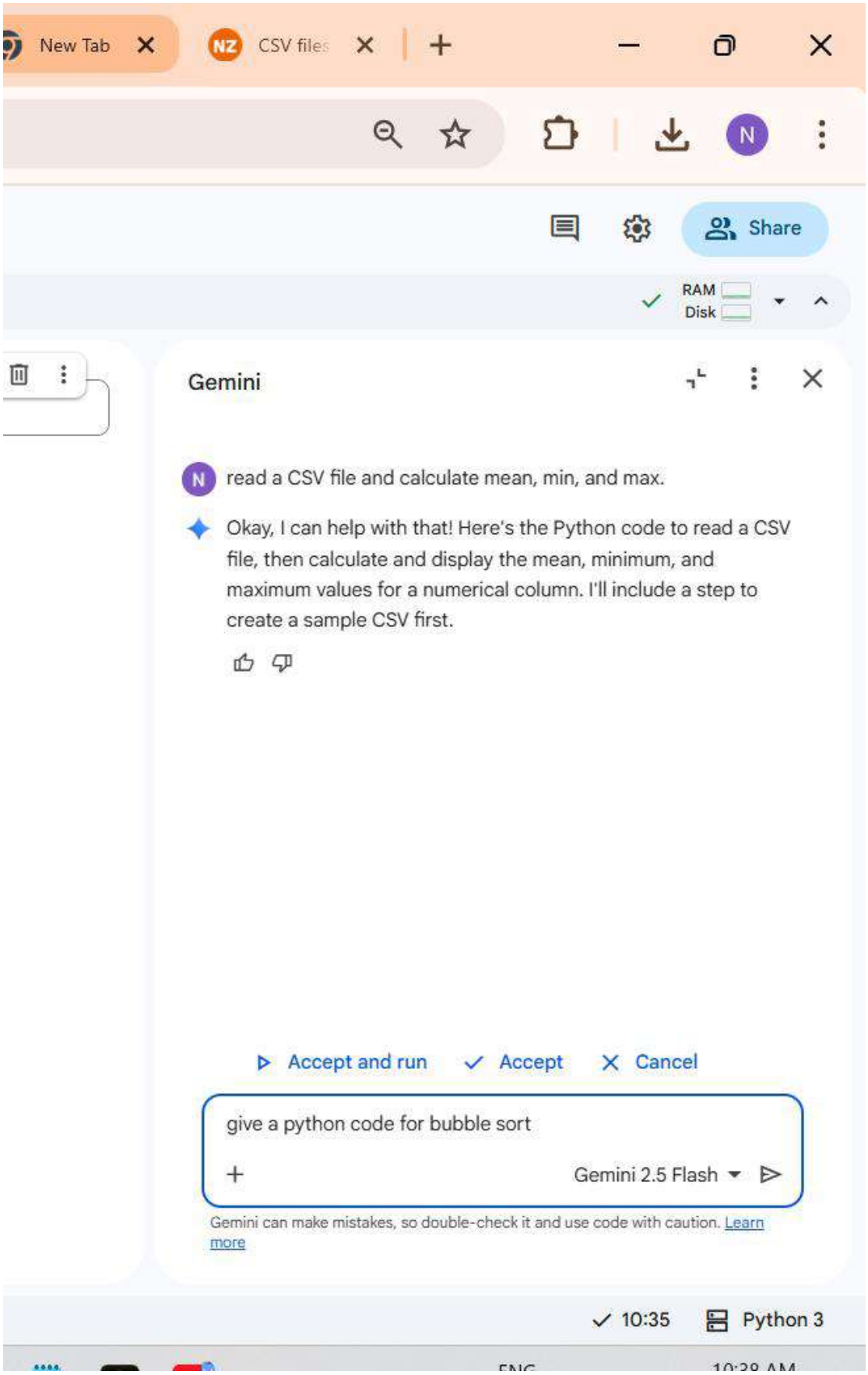
➢ Python's built-in sort()

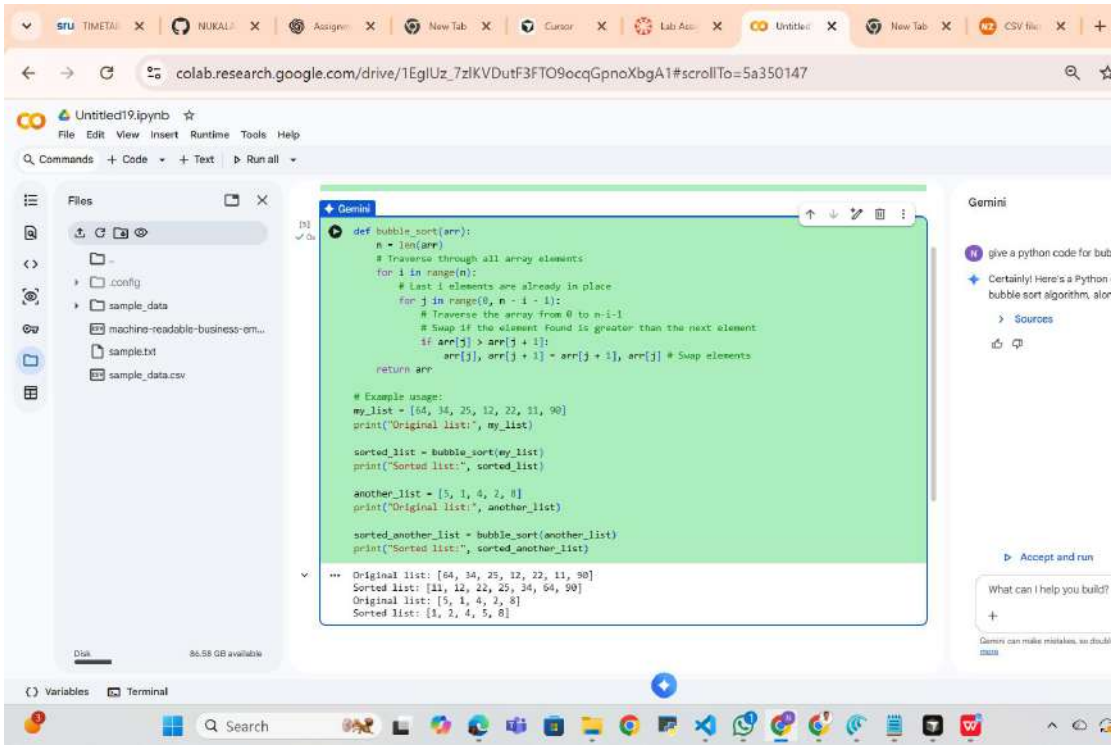
➢ Compare both implementations.

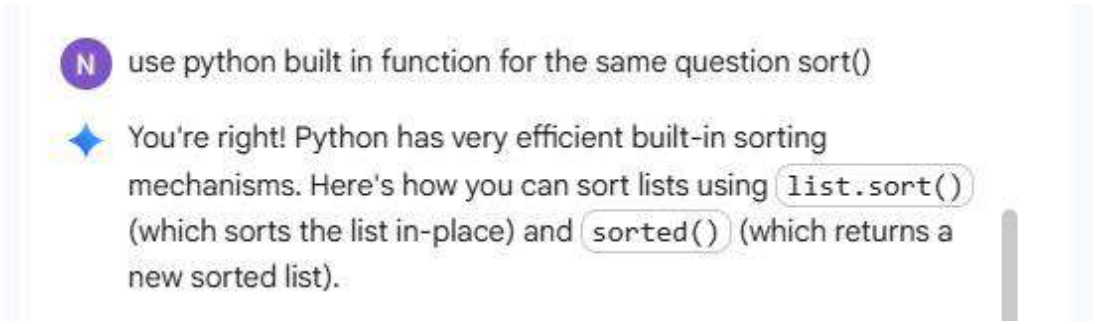
❖ Expected Output:

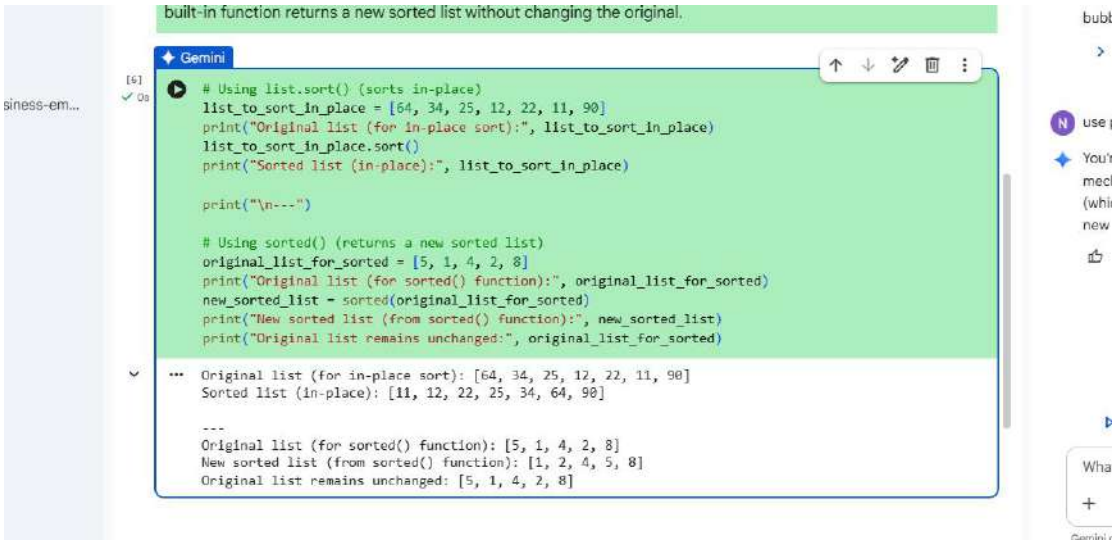
➢ Two versions of code

➢ Short comparison


New Tab
CSV files
N
Share
RAM
Disk
Gemini
read a CSV file and calculate mean, min, and max.
Okay, I can help with that! Here's the Python code to read a CSV file, then calculate and display the mean, minimum, and maximum values for a numerical column. I'll include a step to create a sample CSV first.
Accept and run
Accept
Cancel
give a python code for bubble sort
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more
10:35
Python 3







Short comparision

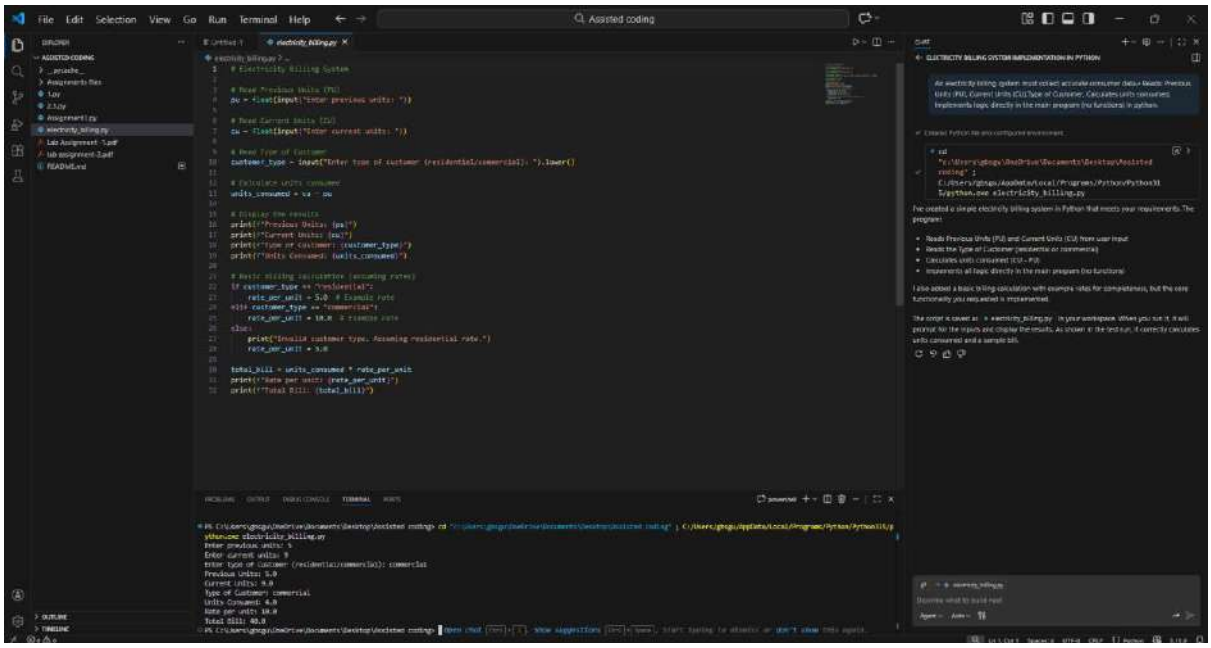
Bubble Sort: It's a simple sorting algorithm, but generally inefficient for large datasets (O(n^2) complexity). It sorts by repeatedly stepping through the list, comparing adjacent elements and swapping them.

Python's list.sort() and sorted(): These are highly optimized and implemented in C, using Timsort (a hybrid stable sorting algorithm with O(n log n) average and worst-case complexity). They are significantly faster and more memory-efficient for most use cases. list.sort() sorts in-place, while sorted() returns a new sorted list.

Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

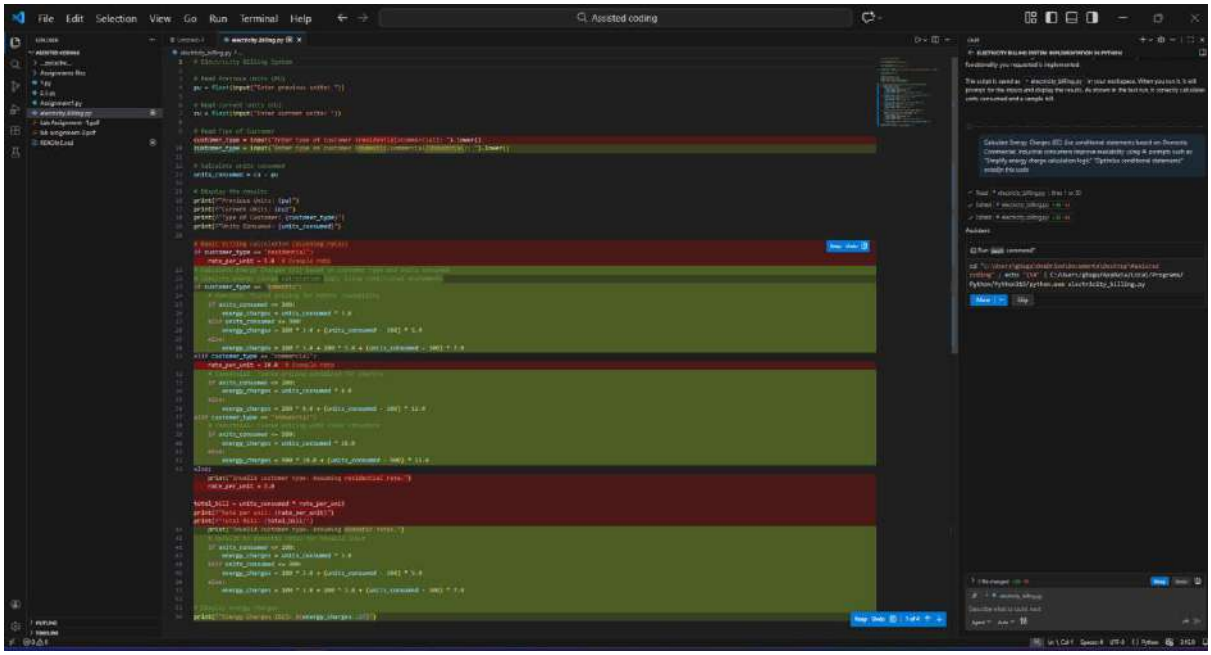
An electricity billing system must collect accurate consumer data.

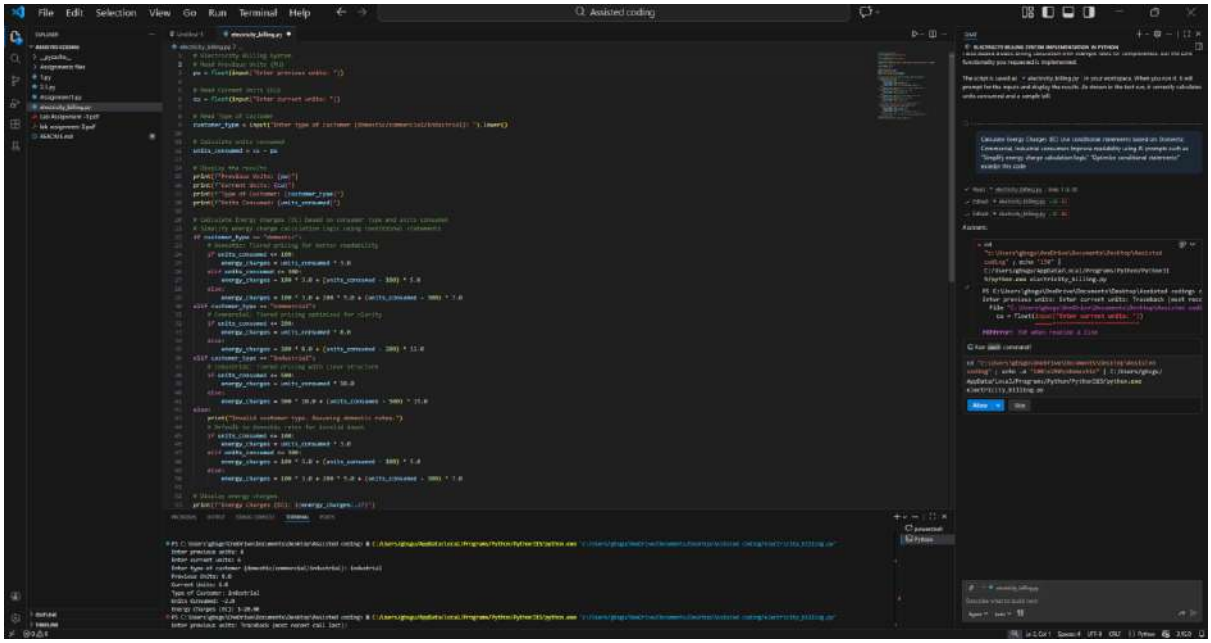


Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

Energy charges depend on the number of units consumed and customer type.

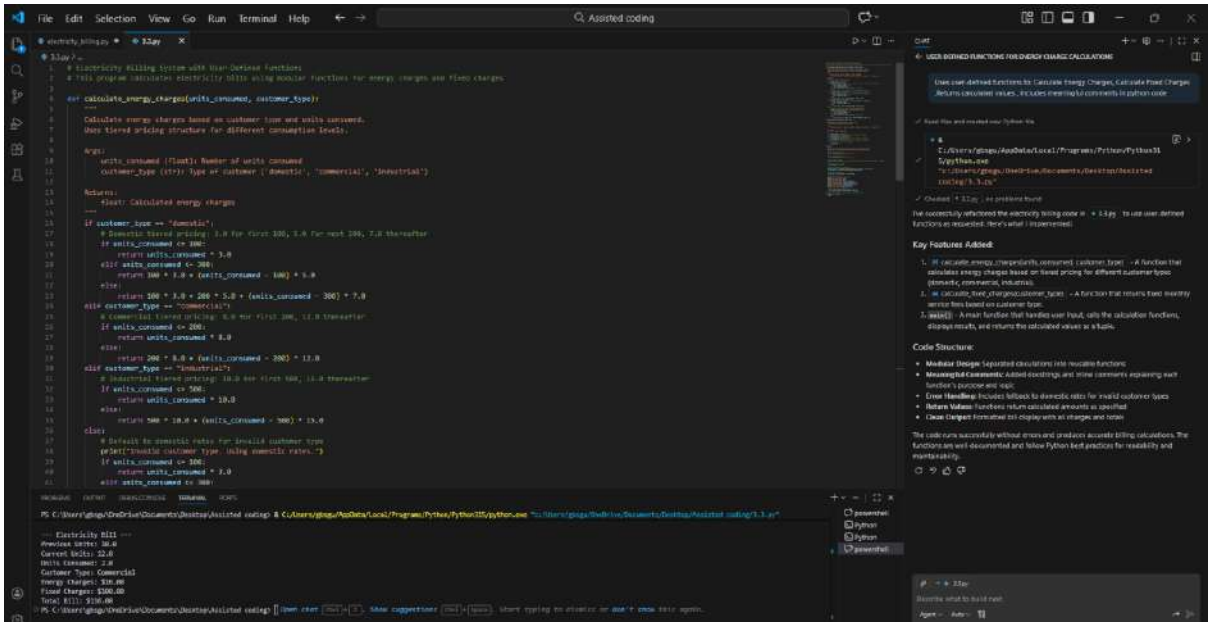


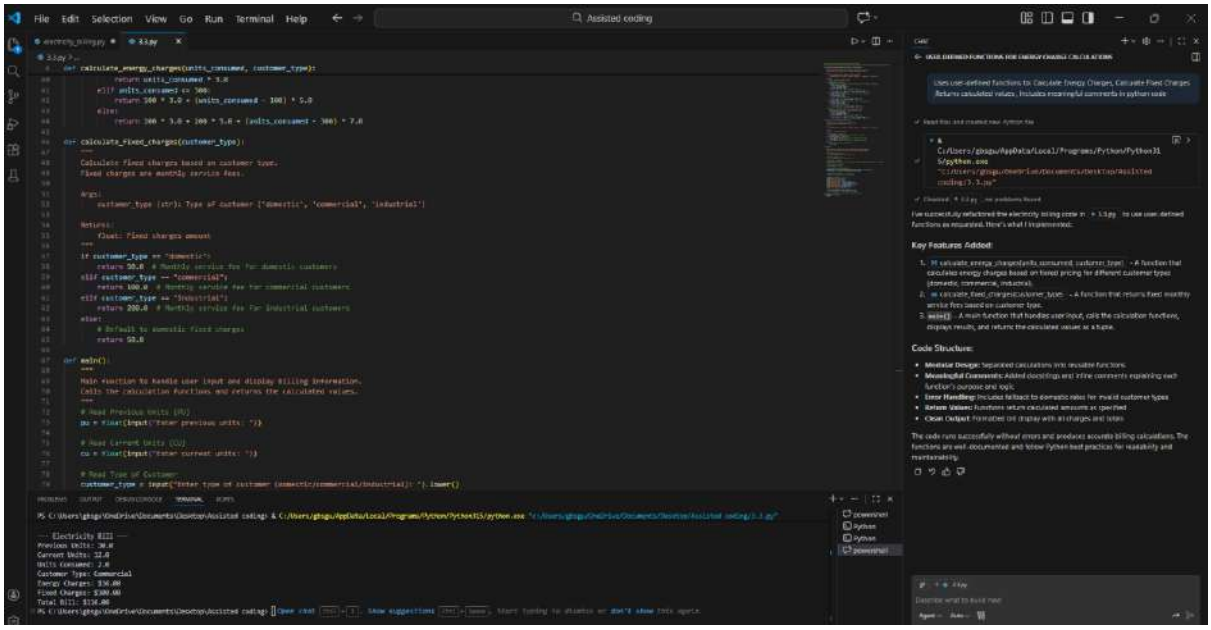


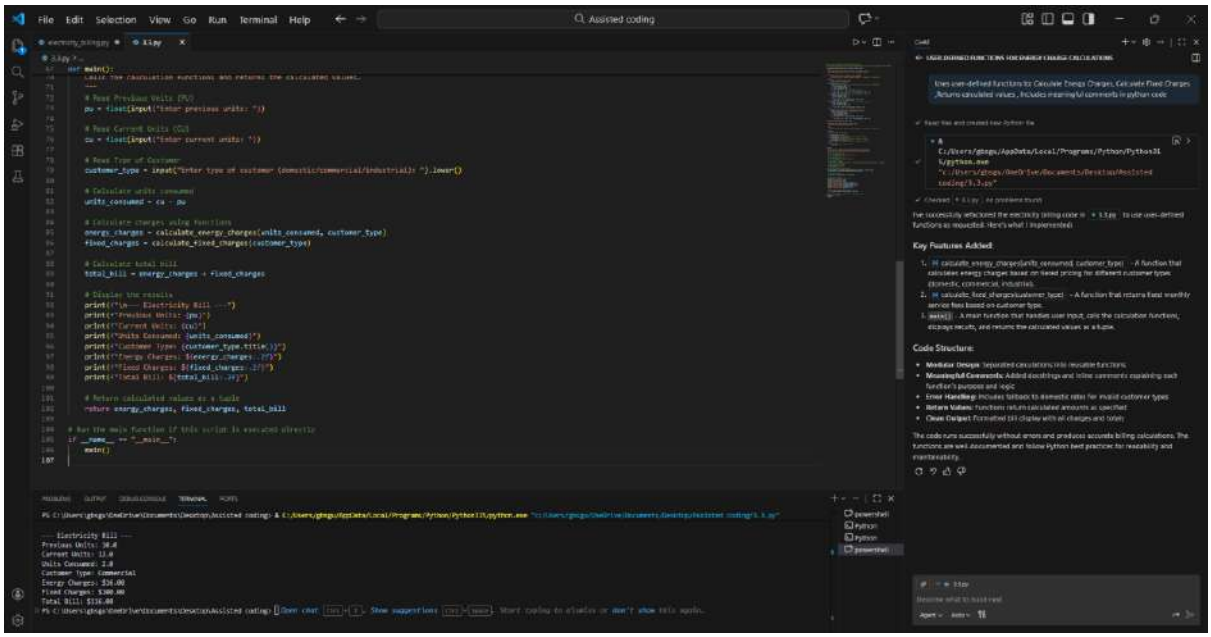
Task 3: Modular Design Using AI Assistance (Using Functions)

Scenario

Billing logic must be reusable for multiple consumers.



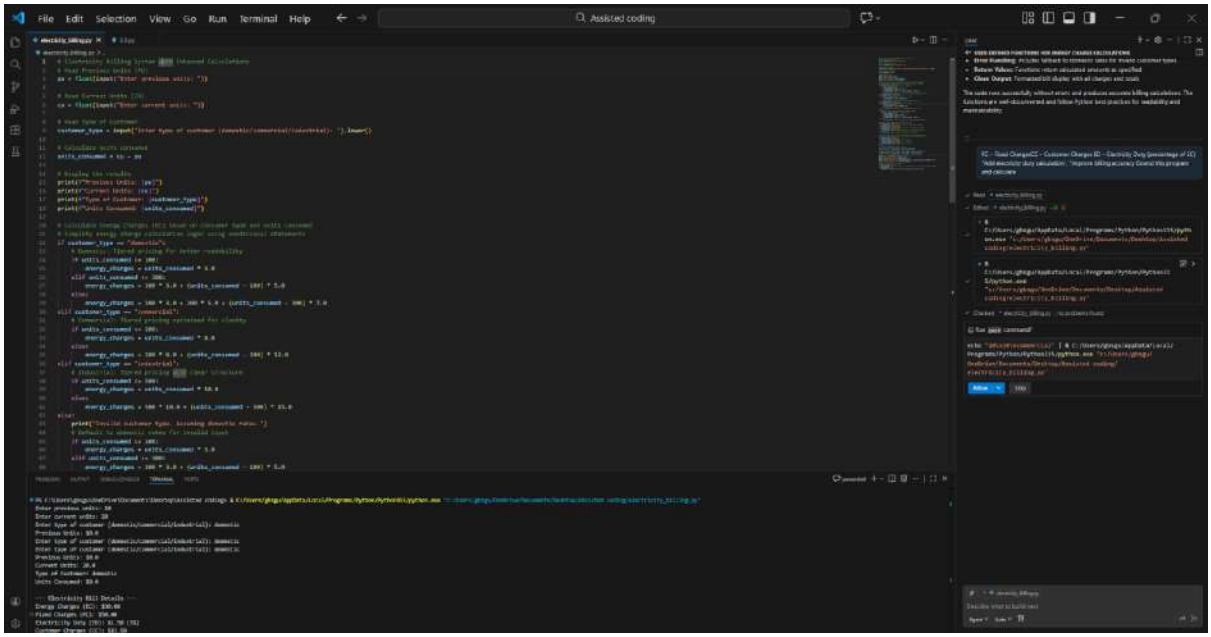


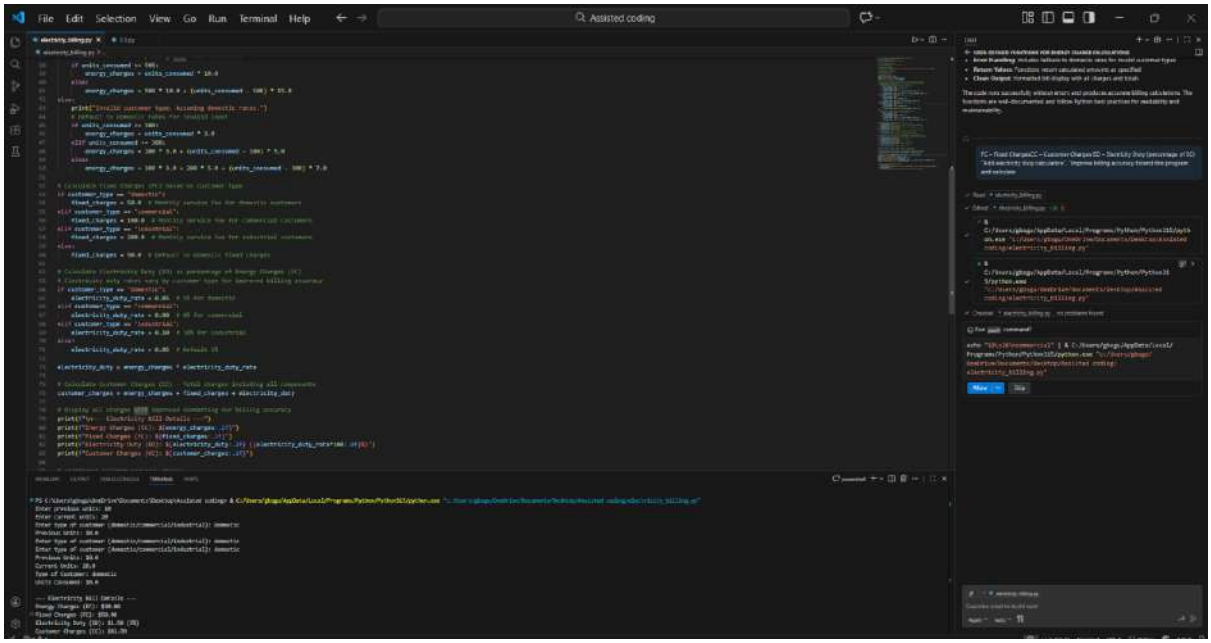


Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.

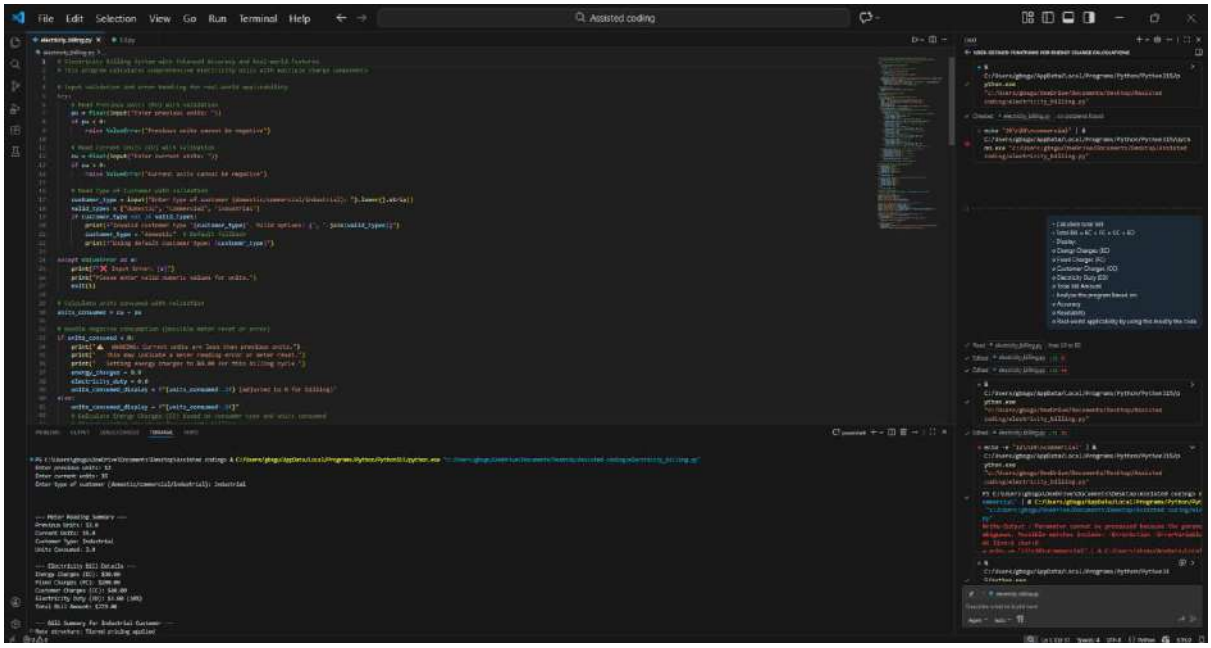


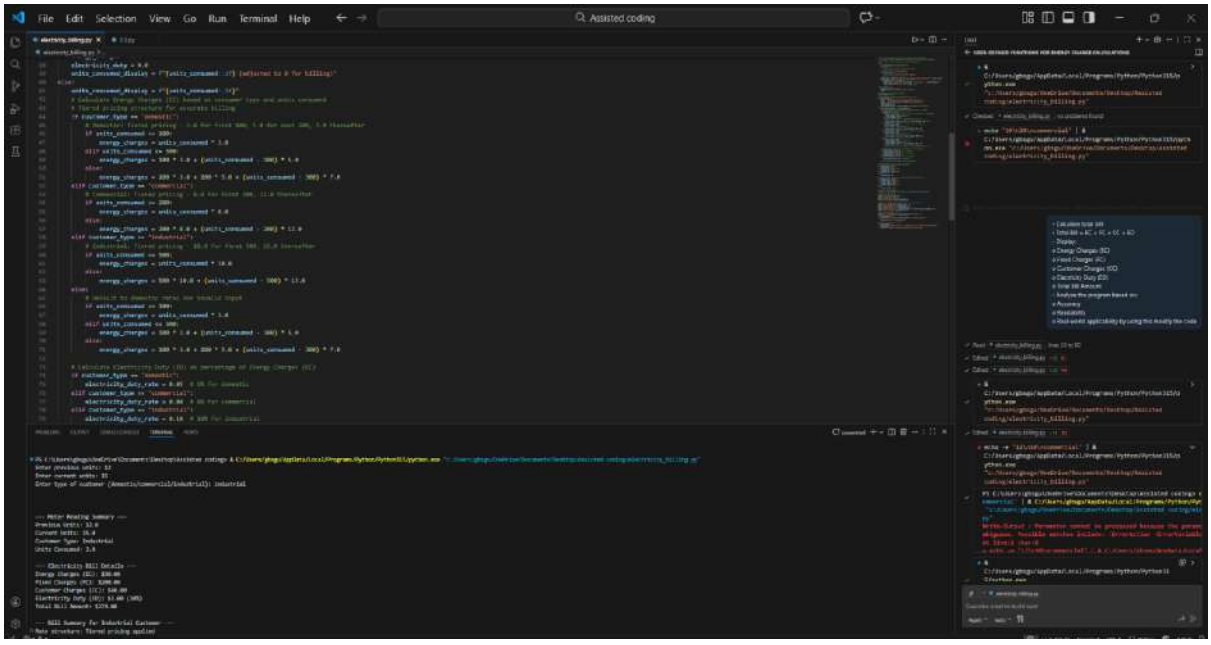


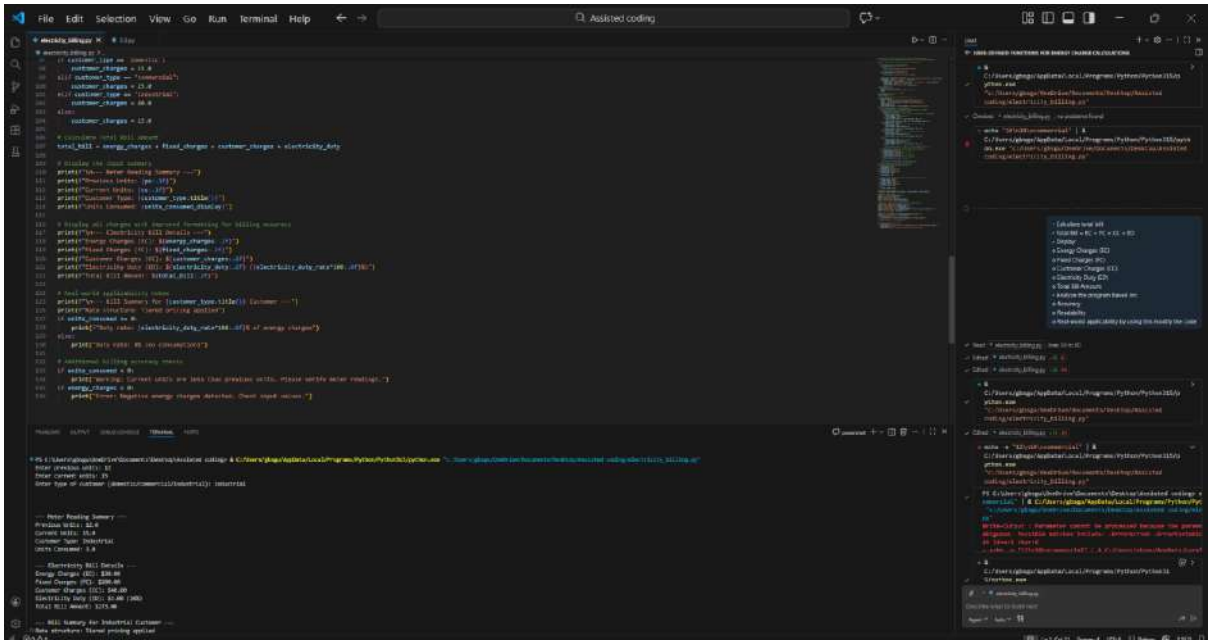
Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.


File Edit Selection View Go Run Terminal Help
Assisted coding


File Edit Selection View Go Run Terminal Help
Assisted coding



This program accurately calculates the electricity bill by using basic arithmetic formulas. The code is easy to read because of meaningful variable names and clear print statements. It is applicable in real-world situations as it follows the standard electricity billing structure used by power departments. The formatted output helps users understand each charge clearly.

Assignment-3.4

Task 1: Zero-shot Prompt – Fibonacci Series Generator

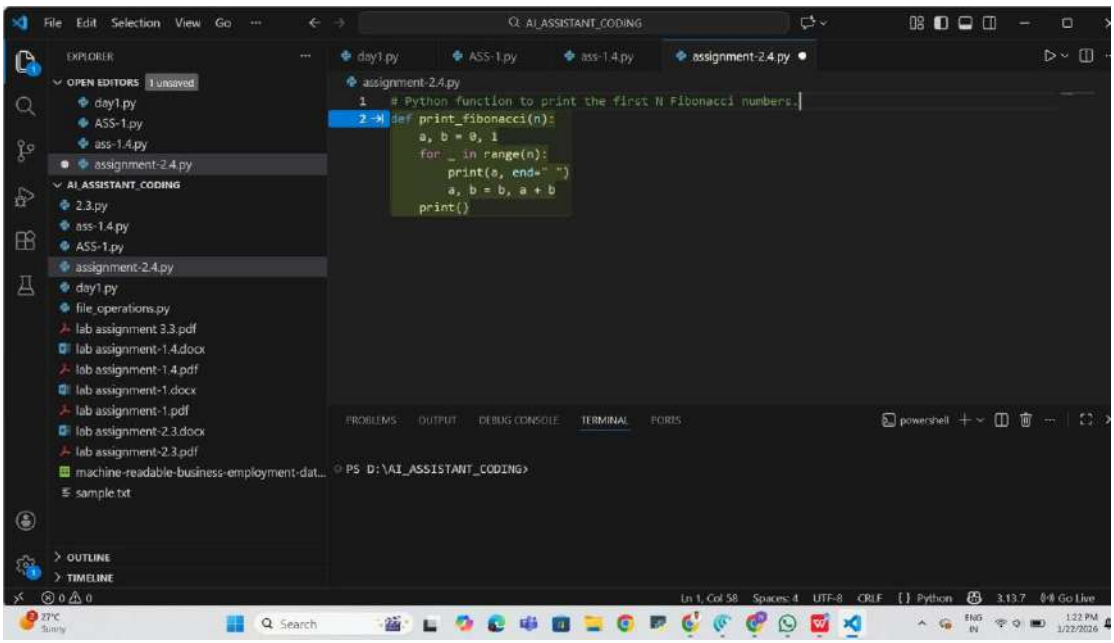
Task Description #1

- Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.

Expected Output #1

- A complete Python function generated by Copilot without any example provided.
- Correct output for sample input N = 7 → 0 1 1 2 3 5 8
- Observation on how Copilot understood the instruction with zero Context

PROMPT:

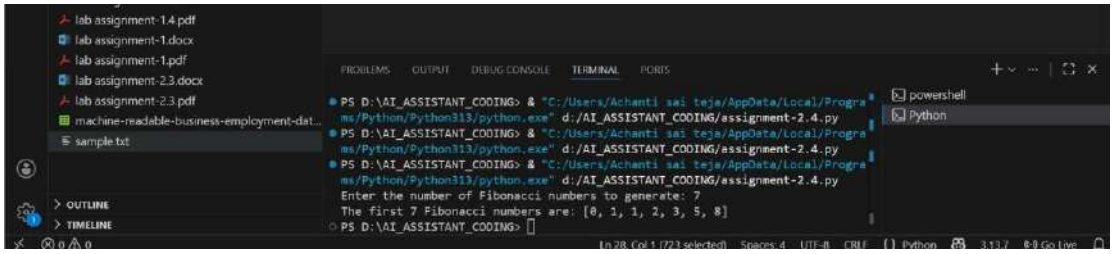


CODE:

```
def fibonacci_sequence(n):
    """
    Generates the first N Fibonacci numbers.

    Args: n: Integer, number of Fibonacci numbers to
        generate

    Returns:
        List of first N Fibonacci numbers
    """ fib_sequence
    = []
    a, b = 0, 1 for _ in range(n):
    fib_sequence.append(a) a, b = b, a +
    b return fib_sequence for i in
    range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
```

```
    return True return True # Test the function num = int(input("Enter
the number of Fibonacci numbers to generate: ")) fib_numbers =
fibonacci_sequence(num)
print(f"The first {num} Fibonacci numbers are: {fib_numbers}")
```

OUTPUT :



Task2: One-shot Prompt – List Reversal Function

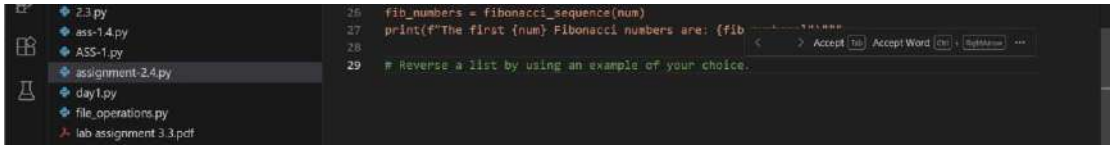
Task Description #2

- Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

Expected Output #2

- Copilot-generated function to reverse a list using slicing or loop.
- Output: [3, 2, 1] for input [1, 2, 3]
- Observation on how adding a single example improved Copilot's accuracy.

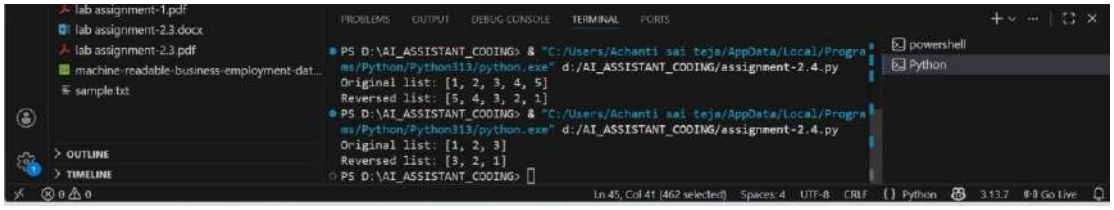
PROMPT:



CODE:

```
# Reverse a list by using an example of your choice
def reverse_list(input_list): """
    Reverses the given list.

    Args: input_list: List of elements to be
        reversed

    Returns:
        List: Reversed list
    """ return input_list[::-1] # Test
the reverse_list function sample_list =
[1, 2, 3] reversed_list =
reverse_list(sample_list)
print(f"Original list: {sample_list}")
print(f"Reversed list: {reversed_list}")
```

OUTPUT:



Adding **just one clear example** helps Copilot quickly infer:

what the function should do the

expected input and output format

This improves **accuracy and reduces ambiguity**, even with a minimal prompt.

Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

• Write a comment with 2–3 examples to help Copilot understand how

to check if a string starts with a capital letter and ends with a period.

Expected Output #3

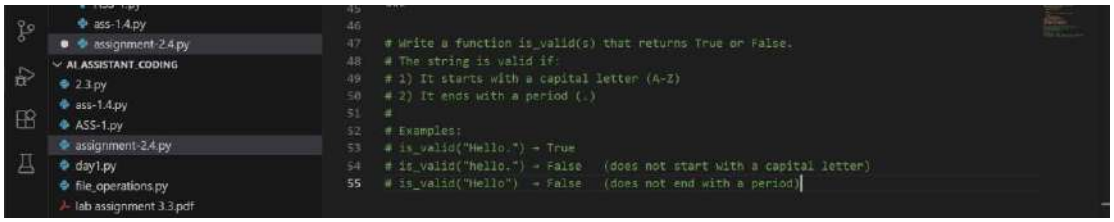
• A function is_valid() that checks the pattern.

• Output: True or False based on input.

• Students reflect on how multiple examples guide Copilot to generate

more accurate code.

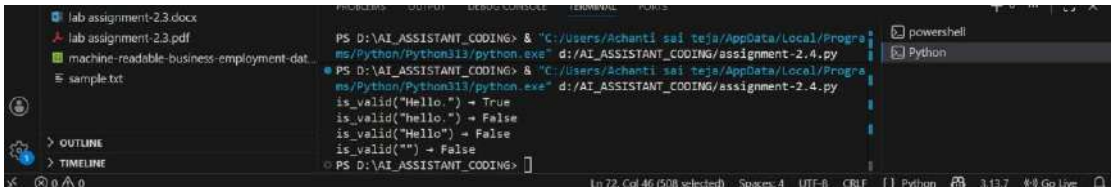
PROMPT:



CODE:

```
def is_valid(s):
    """
    Checks if the string starts with a capital letter and ends with a period.

    Args: s: Input string to
    validate Returns:
        Boolean: True if valid, False otherwise
    """ if len(s)
    == 0:
        return False
    return s[0].isupper() and s[-1] == '.'
# Test the is_valid function test_strings =
["Hello.", "hello.", "Hello", ""] for string in
test_strings:
    result = is_valid(string)
    print(f'is_valid("{string}") → {result}')
```

OUTPUT:



Providing **multiple examples in comments** helps Copilot clearly understand:

what counts as valid

what should be rejected
This leads to **more accurate and relevant code generation**.

Task 4: Zero-shot vs Few-shot – Email Validator

Task Description #4

• First, prompt Copilot to write an email validation function using zero-

shot (just the task in comment).

• Then, rewrite the prompt using few-shot examples.

Expected Output #4

• Compare both outputs:

Zero-shot may result in basic or generic validation.

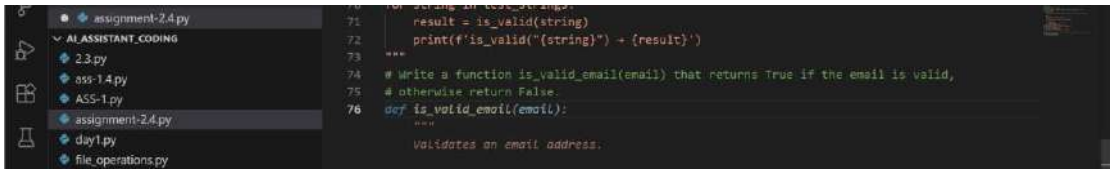
Few-shot gives detailed and specific logic (e.g., @ and domain

checking).

• Submit both code versions and note how few-shot improves

reliability.

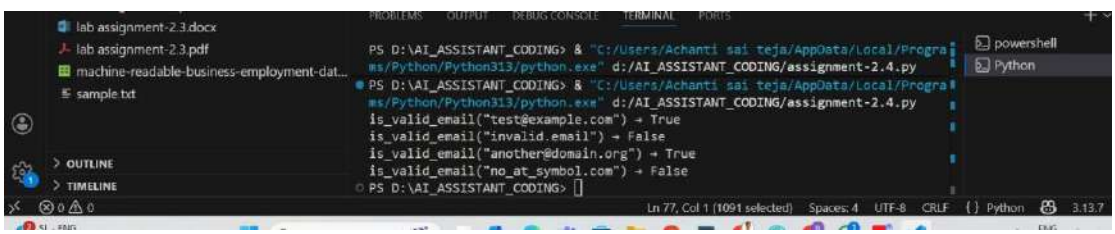
PROMPT:



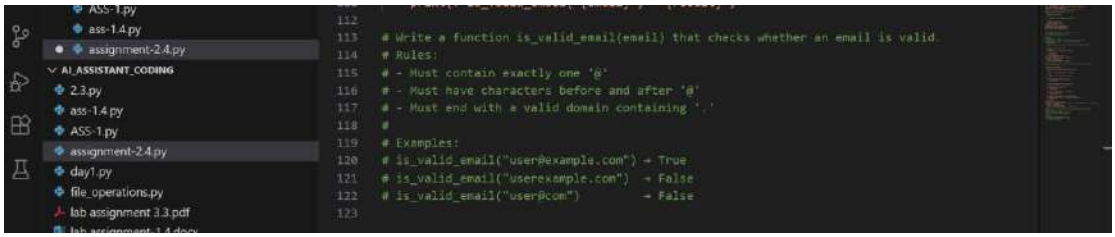
CODE:

```
def is_valid_email(email):
    """
    Validates an email address based on basic criteria.

    Args: email: String, email address to
        validate

    Returns:
        Boolean: True if valid, False otherwise
    """
    # Basic checks for presence of '@' and '.'
    if "@" not in email or "." not in email:
    return False

    # Split the email into local and domain parts
    local_part, domain_part = email.split("@", 1)

    # Local part should not be empty
    if len(local_part) == 0:
        return False

    # Domain part should contain at least one dot
    if "." not in domain_part:
        return False

    # Domain part should not start or end with a dot if
    domain_part.startswith(".") or domain_part.endswith("."):
        return False

    return True
# Test the is_valid_email function
test_emails = ["test@example.com", "invalid.email", "another@domain.org",
"no_at_symbol.com"] for
email in test_emails:
    result = is_valid_email(email)
    print(f'is_valid_email("{email}") → {result}')
```

OUTPUT:



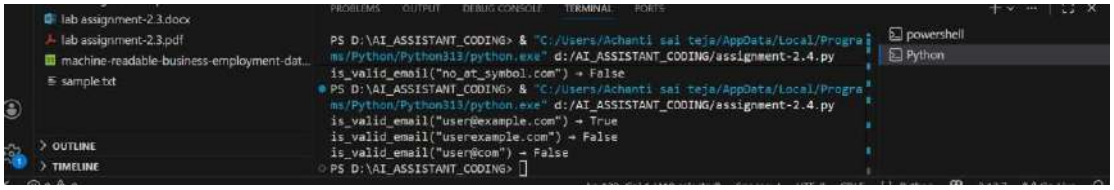
PROMPT:



CODE:

```
def is_valid_email(email):
    if email.count("@") != 1:
        return False

    local, domain = email.split("@")
    if not local or "." not in domain:
        return False
    return True
# Test the is_valid_email function test_emails =
["user@example.com", "userexample.com", "user@com"] for email in
test_emails:
    result = is_valid_email(email)
    print(f'is_valid_email("{email}") → {result}')
```

OUTPUT:



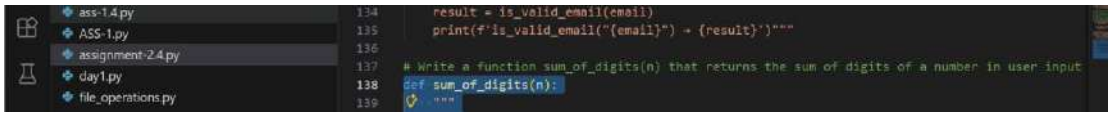
Few-shot prompting **guides Copilot's reasoning** by showing:

valid vs invalid patterns

edge cases

This results in **more reliable, accurate, and meaningful code** compared to zero-shot prompts.

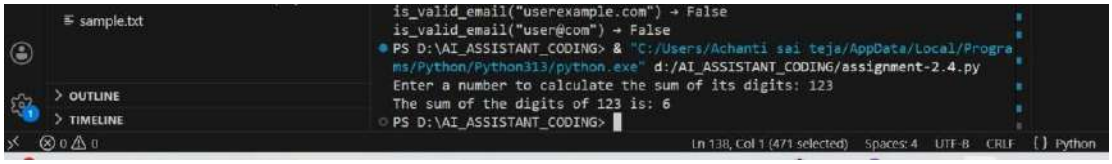
PROMPT:



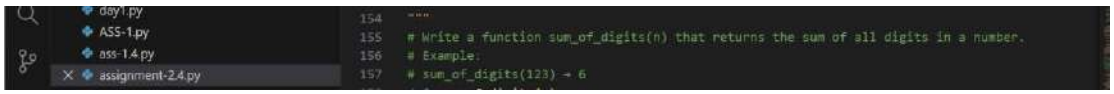
CODE:

```
def sum_of_digits(n):
    """
    Calculates the sum of digits of a given number.

    Args: n: Integer whose digits are to be
        summed

    Returns:
        Integer: Sum of the digits
    """ return sum(int(digit) for digit in
    str(abs(n)))
# Test the sum_of_digits function number = int(input("Enter a number to
calculate the sum of its digits: ")) result = sum_of_digits(number)
print(f"The sum of the digits of {number} is: {result}")
```

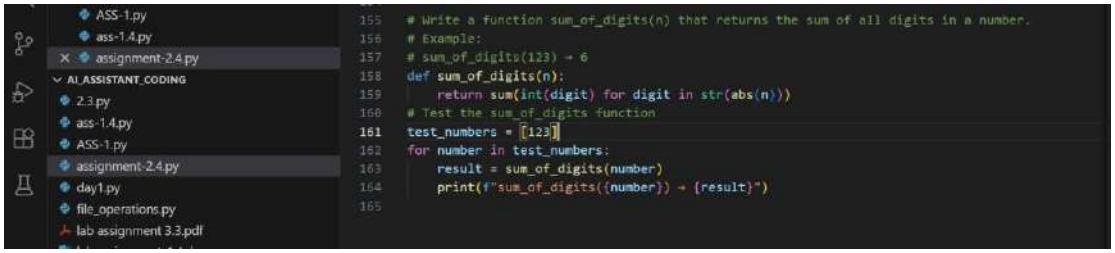
OUTPUT:



PROMPT:



CODE:



OUTPUT: