Assignment – 2.2

Name:K.Manideep

Roll Number: 2303A51192

Batch - 03

AI Assisted Coding

13-01-2026

Task 1: Cleaning Sensor Data

❖ Scenario:

❖ You are cleaning IoT sensor data where negative values are invalid.

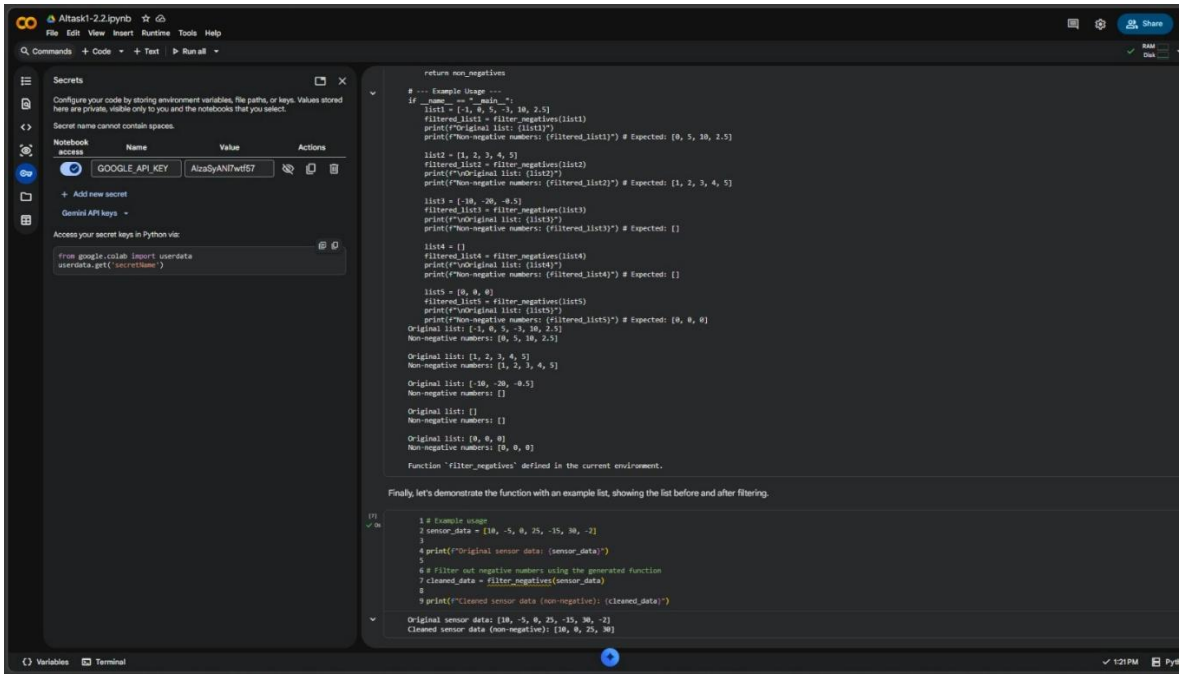
❖ Task:

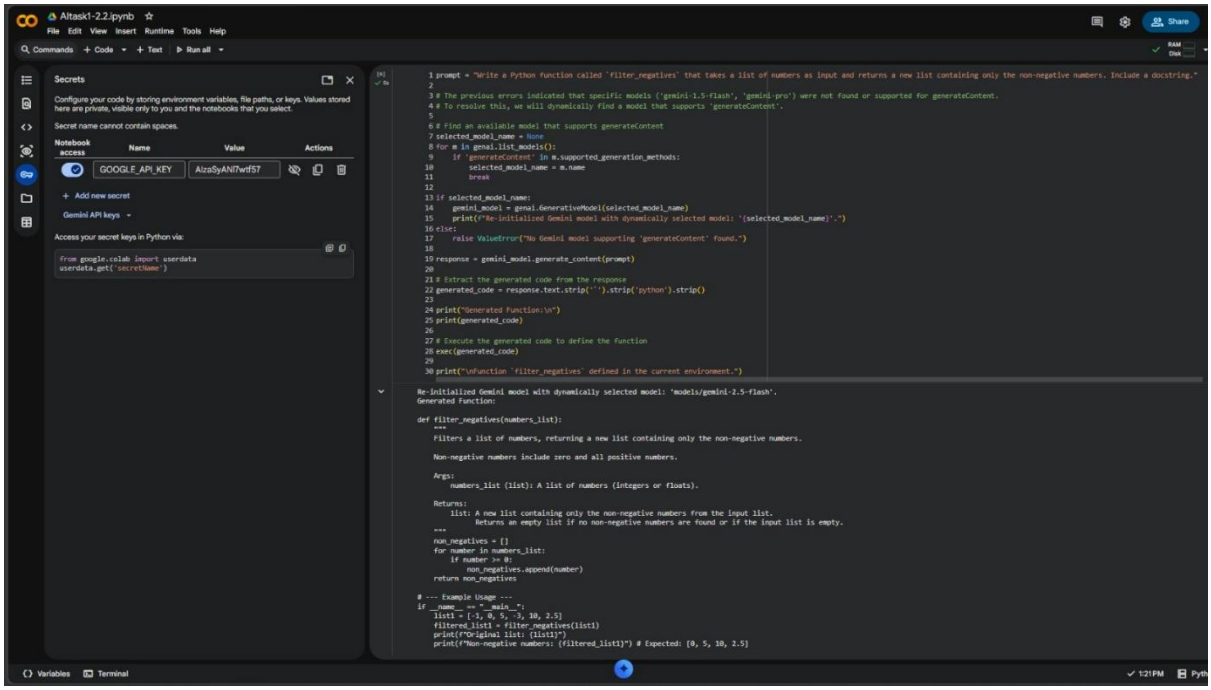
Use Gemini in Colab to generate a function that filters out all negative numbers from a list.

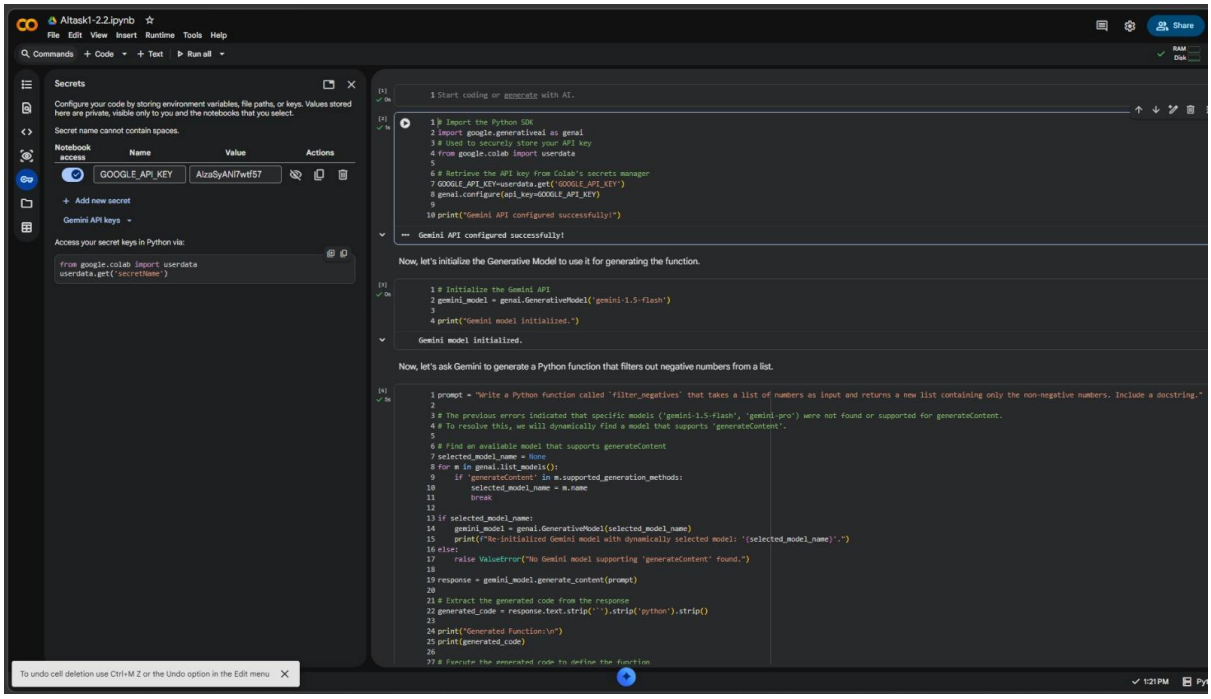
❖ Expected Output:

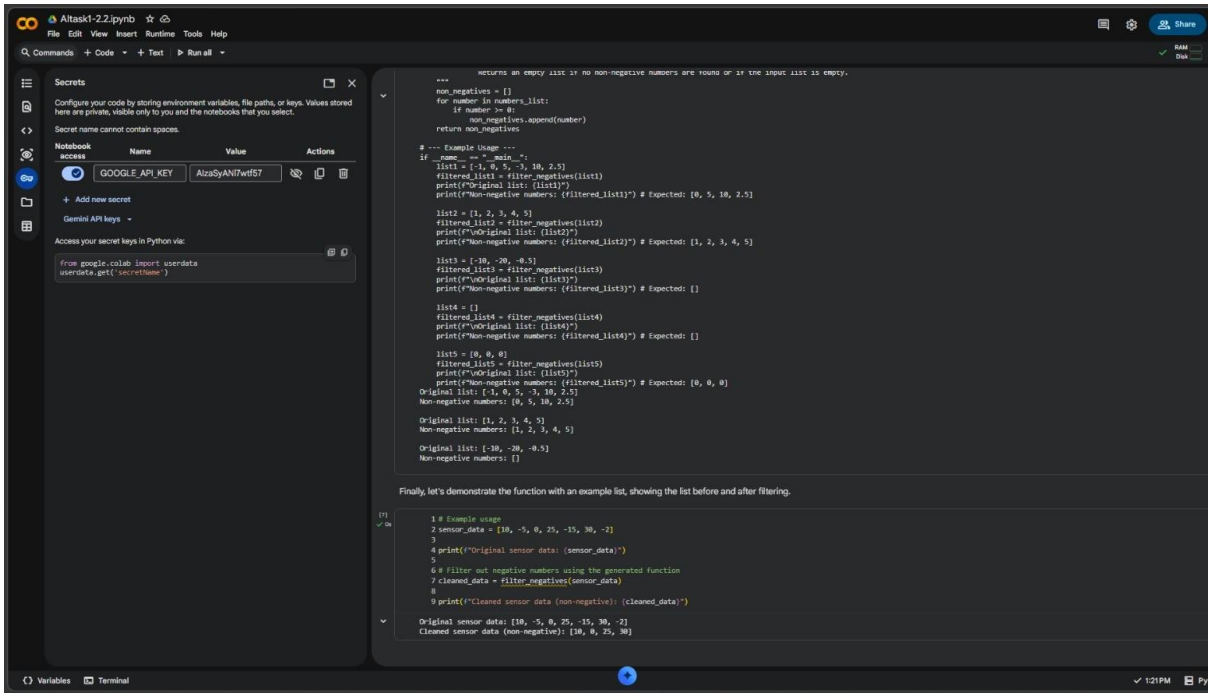
➢ Before/after list

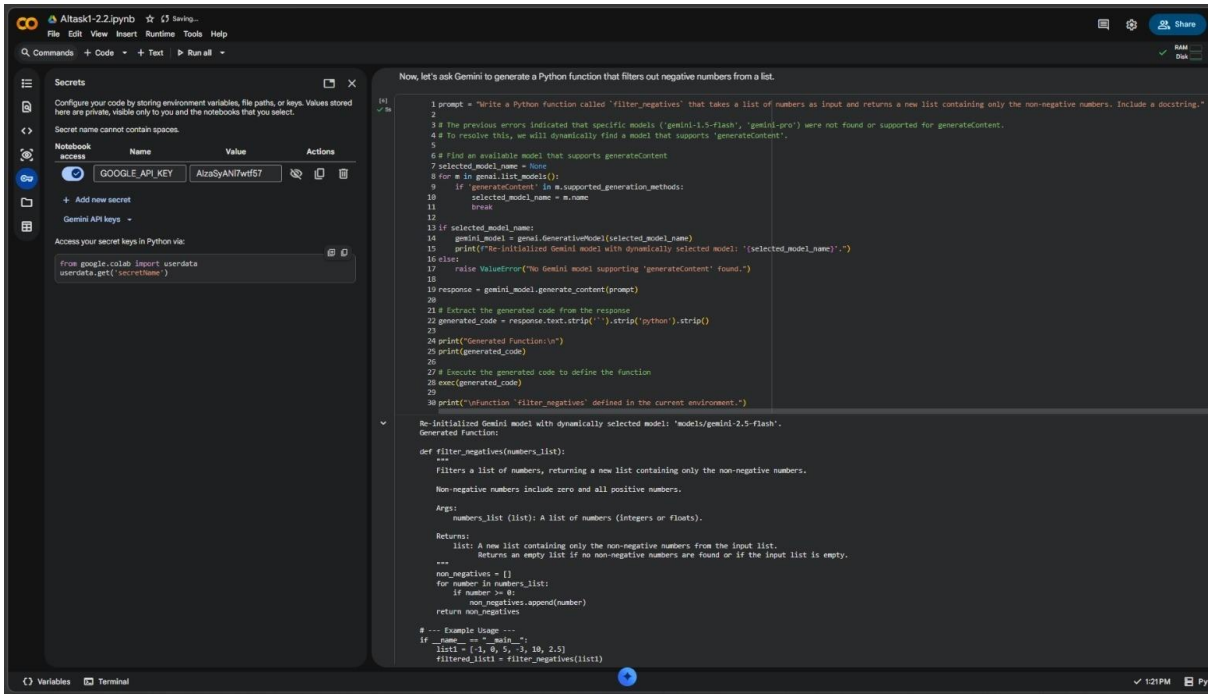
➢ Screenshot of Colab execution

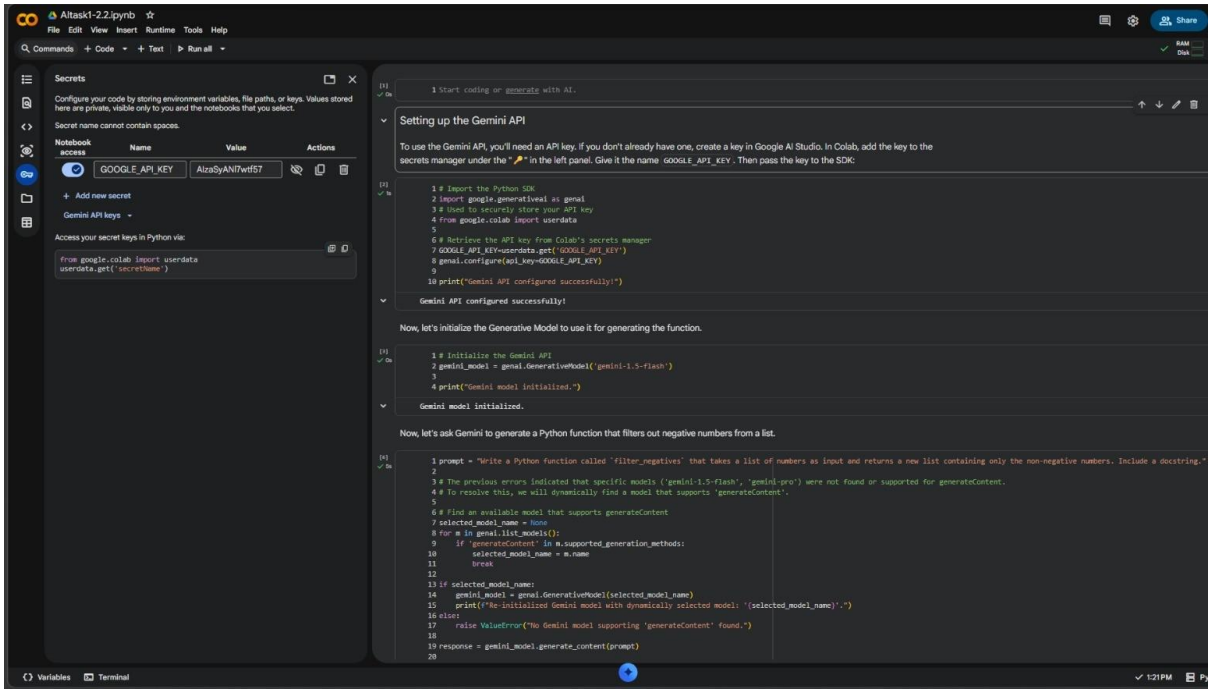



Altask1-2.2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Secrets
Configure your code by storing environment variables, file paths, or keys. Values stored here are private, visible only to you and the notebooks that you select.
Secret name cannot contain spaces.
Notebook access
Name
Value
Actions
GOOGLE_API_KEY
AIzaSyANl7wtf57
Add new secret
Gemini API keys
Access your secret keys in Python via:
from google.colab import userdata
userdata.get('secretName')
prompt = "Write a Python function called `filter_negatives` that takes a list of numbers as input and returns a new list containing only the non-negative numbers. Include a docstring."
# The previous errors indicated that specific models ('gemini-1.5-flash', 'gemini-pro') were not found or supported for generateContent.
# To resolve this, we will dynamically find a model that supports 'generateContent'.
# Find an available model that supports generateContent
selected_model_name = None
for m in genai.list_models():
if 'generateContent' in m.supported_generation_methods:
selected_model_name = m.name
break
if selected_model_name:
gemini_model = genai.GenerativeModel(selected_model_name)
print(f"Re-initialized Gemini model with dynamically selected model: '{selected_model_name}'.")
else:
raise ValueError("No Gemini model supporting 'generateContent' found.")
response = gemini_model.generate_content(prompt)
# Extract the generated code from the response
generated_code = response.text.strip('`').strip('python').strip()
print("Generated Function:\n")
print(generated_code)
# Execute the generated code to define the function
exec(generated_code)
print("\nFunction `filter_negatives` defined in the current environment.")
Re-initialized Gemini model with dynamically selected model: 'models/gemini-2.5-flash'.
Generated Function:
def filter_negatives(numbers_list):
"""
Filters a list of numbers, returning a new list containing only the non-negative numbers.
Non-negative numbers include zero and all positive numbers.
Args:
numbers_list (list): A list of numbers (integers or floats).
Returns:
list: A new list containing only the non-negative numbers from the input list.
Returns an empty list if no non-negative numbers are found or if the input list is empty.
"""
non_negatives = []
for number in numbers_list:
if number >= 0:
non_negatives.append(number)
return non_negatives
# --- Example Usage ---
if __name__ == "__main__":
list1 = [-1, 0, 5, -3, 10, 2.5]
filtered_list1 = filter_negatives(list1)
print(f"Original list: {list1}")
print(f"Non-negative numbers: {filtered_list1}") # Expected: [0, 5, 10, 2.5]
Variables Terminal
1:21PM


Altask1-2.2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Secrets
Configure your code by storing environment variables, file paths, or keys. Values stored here are private, visible only to you and the notebooks that you select.
Secret name cannot contain spaces.
Notebook access
Name
Value
Actions
GOOGLE_API_KEY
AIzaSyANl7wtf57
Add new secret
Gemini API keys
Access your secret keys in Python via:
from google.colab import userdata
userdata.get('secretName')
Start coding or generate with AI.
# Import the Python SDK
import google.generativeai as genai
# Used to securely store your API key
from google.colab import userdata
# Retrieve the API key from Colab's secrets manager
GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
print("Gemini API configured successfully!")
Gemini API configured successfully!
Now, let's initialize the Generative Model to use it for generating the function.
# Initialize the Gemini API
gemini_model = genai.GenerativeModel('gemini-1.5-flash')
print("Gemini model initialized.")
Gemini model initialized.
Now, let's ask Gemini to generate a Python function that filters out negative numbers from a list.
prompt = "Write a Python function called `filter_negatives` that takes a list of numbers as input and returns a new list containing only the non-negative numbers. Include a docstring."
# The previous errors indicated that specific models ('gemini-1.5-flash', 'gemini-pro') were not found or supported for generateContent.
# To resolve this, we will dynamically find a model that supports 'generateContent'.
# Find an available model that supports generateContent
selected_model_name = None
for m in genai.list_models():
if 'generateContent' in m.supported_generation_methods:
selected_model_name = m.name
break
if selected_model_name:
gemini_model = genai.GenerativeModel(selected_model_name)
print(f"Re-initialized Gemini model with dynamically selected model: '{selected_model_name}'.")
else:
raise ValueError("No Gemini model supporting 'generateContent' found.")
response = gemini_model.generate_content(prompt)
# Extract the generated code from the response
generated_code = response.text.strip('`').strip('python').strip()
print("Generated Function:\n")
print(generated_code)
# Execute the generated code to define the function
To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu
1:21PM


Altask1-2.2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Secrets
Configure your code by storing environment variables, file paths, or keys. Values stored here are private, visible only to you and the notebooks that you select.
Secret name cannot contain spaces.
Notebook access
Name
Value
Actions
GOOGLE_API_KEY
AIzaSyANl7wtf57
Add new secret
Gemini API keys
Access your secret keys in Python via:
from google.colab import userdata
userdata.get('secretName')
Returns an empty list if no non-negative numbers are found or if the input list is empty.
"""
non_negatives = []
for number in numbers_list:
if number >= 0:
non_negatives.append(number)
return non_negatives
# --- Example Usage ---
if __name__ == "__main__":
list1 = [-1, 0, 5, -3, 10, 2.5]
filtered_list1 = filter_negatives(list1)
print(f"Original list: {list1}")
print(f"Non-negative numbers: {filtered_list1}") # Expected: [0, 5, 10, 2.5]
list2 = [1, 2, 3, 4, 5]
filtered_list2 = filter_negatives(list2)
print(f"\nOriginal list: {list2}")
print(f"Non-negative numbers: {filtered_list2}") # Expected: [1, 2, 3, 4, 5]
list3 = [-10, -20, -0.5]
filtered_list3 = filter_negatives(list3)
print(f"\nOriginal list: {list3}")
print(f"Non-negative numbers: {filtered_list3}") # Expected: []
list4 = []
filtered_list4 = filter_negatives(list4)
print(f"\nOriginal list: {list4}")
print(f"Non-negative numbers: {filtered_list4}") # Expected: []
list5 = [0, 0, 0]
filtered_list5 = filter_negatives(list5)
print(f"\nOriginal list: {list5}")
print(f"Non-negative numbers: {filtered_list5}") # Expected: [0, 0, 0]
Original list: [-1, 0, 5, -3, 10, 2.5]
Non-negative numbers: [0, 5, 10, 2.5]
Original list: [1, 2, 3, 4, 5]
Non-negative numbers: [1, 2, 3, 4, 5]
Original list: [-10, -20, -0.5]
Non-negative numbers: []
Finally, let's demonstrate the function with an example list, showing the list before and after filtering.
# Example usage
sensor_data = [10, -5, 0, 25, -15, 30, -2]
print(f"Original sensor data: {sensor_data}")
# Filter out negative numbers using the generated function
cleaned_data = filter_negatives(sensor_data)
print(f"Cleaned sensor data (non-negative): {cleaned_data}")
Original sensor data: [10, -5, 0, 25, -15, 30, -2]
Cleaned sensor data (non-negative): [10, 0, 25, 30]
Variables Terminal
1:21PM


Altask1-2.2.ipynb
Saving...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Secrets
Configure your code by storing environment variables, file paths, or keys. Values stored here are private, visible only to you and the notebooks that you select.
Secret name cannot contain spaces.
Notebook access
Name
Value
Actions
GOOGLE_API_KEY
AIzaSyANl7wtf57
Add new secret
Gemini API keys
Access your secret keys in Python via:
from google.colab import userdata
userdata.get('secretName')
Now, let's ask Gemini to generate a Python function that filters out negative numbers from a list.
prompt = "Write a Python function called `filter_negatives` that takes a list of numbers as input and returns a new list containing only the non-negative numbers. Include a docstring."
# The previous errors indicated that specific models ('gemini-1.5-flash', 'gemini-pro') were not found or supported for generateContent.
# To resolve this, we will dynamically find a model that supports 'generateContent'.
# Find an available model that supports generateContent
selected_model_name = None
for m in genai.list_models():
if 'generateContent' in m.supported_generation_methods:
selected_model_name = m.name
break
if selected_model_name:
gemini_model = genai.GenerativeModel(selected_model_name)
print(f"Re-initialized Gemini model with dynamically selected model: '{selected_model_name}'.")
else:
raise ValueError("No Gemini model supporting 'generateContent' found.")
response = gemini_model.generate_content(prompt)
# Extract the generated code from the response
generated_code = response.text.strip('`').strip('python').strip()
print("Generated Function:\n")
print(generated_code)
# Execute the generated code to define the function
exec(generated_code)
print("\nFunction `filter_negatives` defined in the current environment.")
Re-initialized Gemini model with dynamically selected model: 'models/gemini-2.5-flash'.
Generated Function:
def filter_negatives(numbers_list):
"""
Filters a list of numbers, returning a new list containing only the non-negative numbers.
Non-negative numbers include zero and all positive numbers.
Args:
numbers_list (list): A list of numbers (integers or floats).
Returns:
list: A new list containing only the non-negative numbers from the input list.
Returns an empty list if no non-negative numbers are found or if the input list is empty.
"""
non_negatives = []
for number in numbers_list:
if number >= 0:
non_negatives.append(number)
return non_negatives
# --- Example Usage ---
if __name__ == "__main__":
list1 = [-1, 0, 5, -3, 10, 2.5]
filtered_list1 = filter_negatives(list1)
Variables Terminal
1:21PM



Task 2: String Character Analysis

❖ Scenario:

You are building a text-analysis feature.

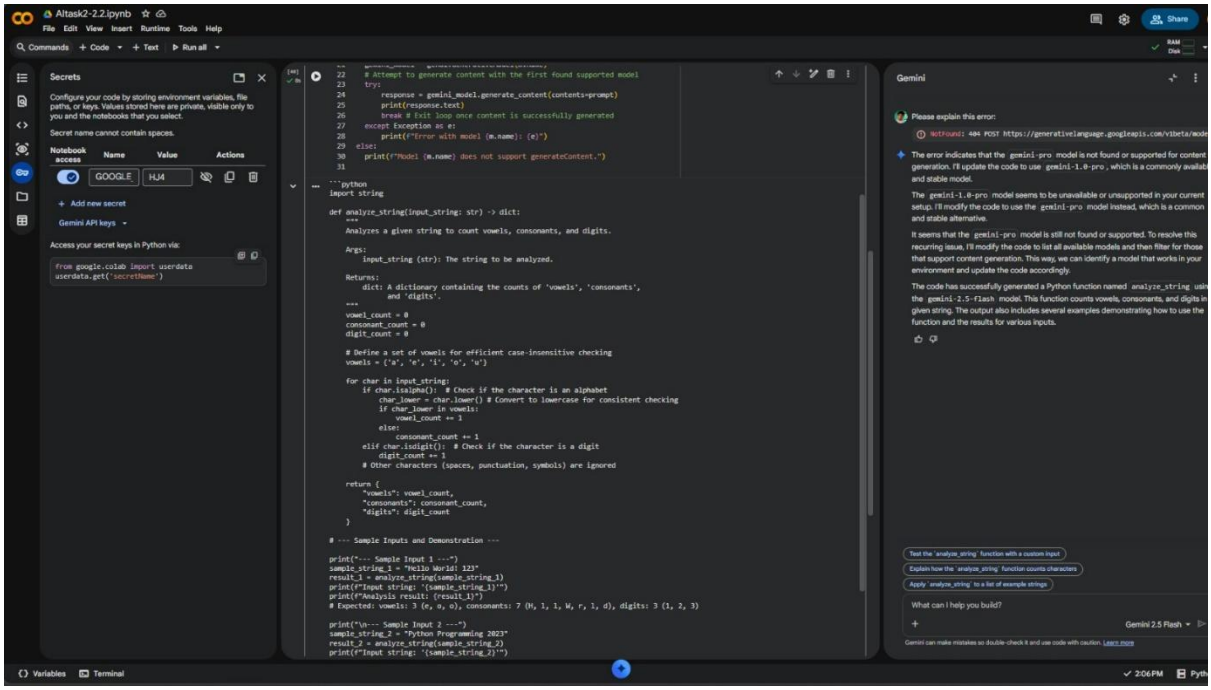
❖ Task:

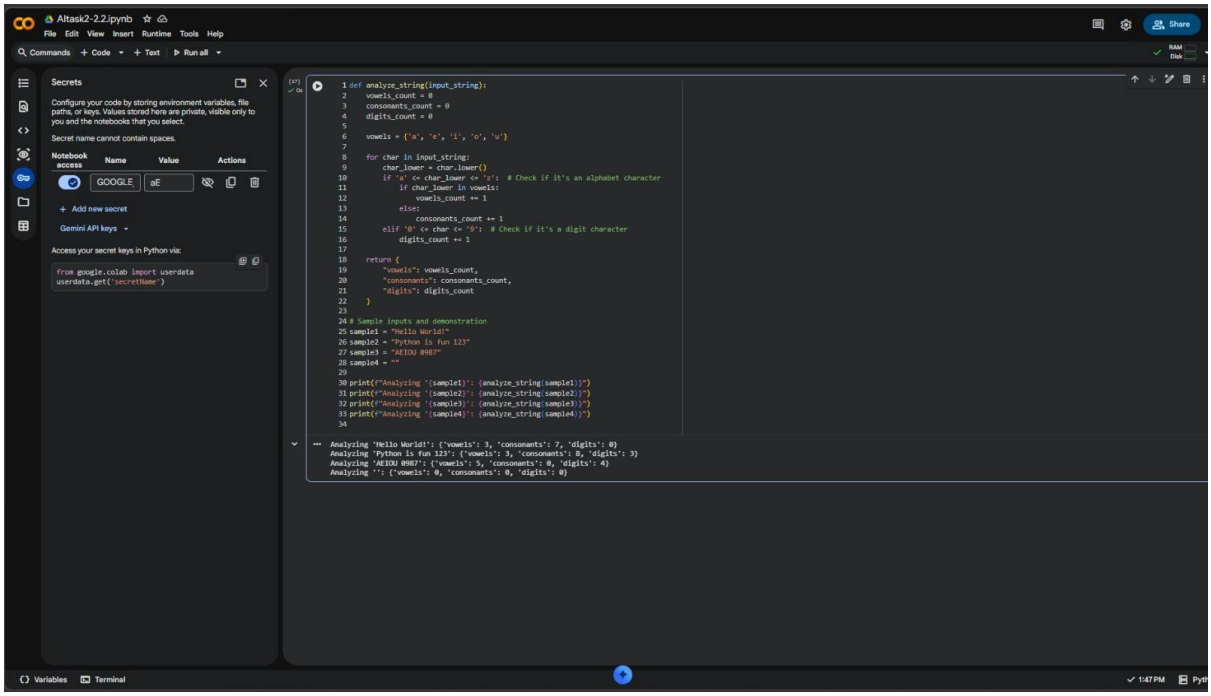
Use Gemini to generate a Python function that counts vowels, consonants, and digits in a string.

❖ Expected Output:

➢ Working function

➢ Sample inputs and outputs





Task 3: Palindrome Check – Tool Comparison

❖ Scenario:

You must decide which AI tool is clearer for string logic.

❖ **Task:**

Generate a palindrome-checking function using Gemini and Copilot, then compare the results.

❖ **Expected Output:**

➢ **Side-by-side code comparison**

➢ **Observations on clarity and structure**

Feature	Google Gemini (Colab)	GitHub Copilot / Cursor AI
Logic Approach	Often uses `string == string[::-1]` (slicing) for simplicity.	Often provides more "robust" versions, sometimes using `reversed()` or loops.
Documentation	Provides conversational explanations and usage examples.	Provides inline docstrings and focuses on integration within the IDE.
Conciseness	Very concise, aims for readability.	Aims for production-readiness (handling edge cases like spaces/case).

Task 4: Code Explanation Using AI

❖ **Scenario:**

You are reviewing unfamiliar code written by another developer.

❖ **Task:**

Ask Gemini to explain a Python function (prime check OR palindrome check) line by line.

❖ **Expected Output:**

➢ **Code snippet**

➢ **AI explanation**

➢ **Student comments on understanding**

```
#Task 3: Palindrome Check
def is_palindrome(s):
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned == cleaned[::-1]
# Test cases
print(is_palindrome("A man a plan a canal Panama"))  # True
print(is_palindrome("race a car"))  # False
Comparison: Gemini AI vs Copilot:.
1.Coding Performance:
->Copilot is generally better integrated into IDEs for fast code suggestions and developer workflows.
->Gemini can generate code too, but tends to be slower and broader-focused rather than as tightly embedded into editor workflows.
Winner for pure coding speed -> Copilot

2.Context & Reasoning:
->Gemini excels at deep reasoning, large context understanding (docs + code + research), and handling multimodal inputs like images/diagrams.
->Copilot is optimized for shorter, code-centric tasks within text editors.
Winner for complex tasks -> Gemini

3.Integration & Workflow:
->Copilot works directly in popular editors with minimal setup.
->Gemini can integrate through CLI or APIs and into Google's ecosystem, but isn't as IDE-native by default (though integrations are growing).
Winner for seamless coding experience -> Copilot

4.Multimodal & General Intelligence:
->Gemini supports multimodal input (text, visuals) and broader tasks like research, explanations, documentation, strategy, etc.
->Copilot focuses on code generation and in-editor support only.
Winner for versatility -> Gemini

5.Pricing & Value

->Copilot is usually around $10/mo for individuals and integrated deeply with GitHub tools.
->Gemini varies by plan but often comes bundled with Google services (Cloud, Workspace), and free tiers like Gemini CLI offer generous daily usage.
Winner depends on budget & ecosystem needs

6.So Which One Is Better?
Use Case    Better Choice:
->Daily coding, fast suggestions inside your editor GitHub Copilot
->Complex tasks -> reasoning, docs, long projects, multimodal input  Google Gemini
->Working within GitHub & developer workflow    Copilot
->Handling diverse non-code tasks or big context workflows  Gemini

7.Many developers actually use both:
->Copilot for IDE coding speed,
->Gemini for deep questions, research, design docs, and multimodal tasks.

8.Quick Summary

->Copilot -> Best code-focused assistant with strong editor integration and real-time code generation.
->Gemini -> Best general AI partner for big-picture thinking, multimodal understanding, and broad tasks beyond code.
```