

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

The screenshot shows the Google Colab interface with a notebook titled 'Altask1-2.2.ipynb'. On the left, there's a sidebar for managing environment variables and secret keys. The main area contains Python code for filtering negative numbers from lists. The code includes several examples and demonstrates the function's behavior with various input lists. The output pane shows the results of each print statement, such as the original list and the filtered list (non-negative numbers). The bottom of the screen shows the 'Variables' and 'Terminal' tabs.

```
# --- Example usage ---
if __name__ == "__main__":
    list1 = [-1, 0, 5, 10, 2.5]
    list2 = [-10, -20, -0.5]
    list3 = [1, 2, 3, 4, 5]
    list4 = []

# Filter out negative numbers using the generated function
filter_function = filter_non_negatives(list1)
print("Original list: (list1)") # Expected: [-1, 0, 5, 10, 2.5]
print("Non-negative numbers: (filter_function(list1))") # Expected: [0, 5, 10, 2.5]

print("Original list: (list2)") # Expected: [-10, -20, -0.5]
print("Non-negative numbers: (filter_function(list2))") # Expected: [-0.5]

print("Original list: (list3)") # Expected: [1, 2, 3, 4, 5]
print("Non-negative numbers: (filter_function(list3))") # Expected: [1, 2, 3, 4, 5]

print("Original list: (list4)") # Expected: []
print("Non-negative numbers: (filter_function(list4))") # Expected: []

# Finally, let's demonstrate the function with an example list, showing the list before and after filtering.
sensor_data = [10, -5, 0, 25, -15, 30, -2]
print("Original sensor data: (sensor_data)") # Expected: [10, -5, 0, 25, -15, 30, -2]
cleaned_data = filter_non_negatives(sensor_data)
print("Cleaned sensor data (non-negative): (cleaned_data)") # Expected: [10, 0, 25, 30]
```

Altask1-22.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

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		
<input checked="" type="checkbox"/> GOOGLE_API_KEY	AlzaSyAHiTwf57				

+ Add new secret

Gemini API keys

Access your secret keys in Python via:

```
from google.colab import userdata
userdata.get('secretname')
```

I 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."
2
3 # The previous errors indicated that specific models ('gemini-1.5-flash', 'gemini-pro') were not found or supported for generateContent.
4 # To resolve this, we will dynamically find a model that supports 'generateContent'.
5
6 # Find an available model that supports generateContent
7 # Loop through all Gemini models
8 for m in general.list_models():
9 if 'generateContent' in m.supported_generation_methods:
10 selected_model_name = m.name
11 break
12
13 if selected_model_name:
14 gemini_model = general.GenerativeModel(selected_model_name)
15 print(f'Re-initialized Gemini model with dynamically selected model: "{selected_model_name}"')
16 else:
17 raise ValueError(f"No Gemini model supporting 'generateContent' found.")
18
19 response = gemini_model.generate_content(prompt)
20
21 # Extract the generated code from the response
22 generated_code = response.text.strip().strip('python').strip()
23
24 print(f'Generated Function:\n{generated_code}')
25
26 # Execute the generated code to define the Function
27 exec(generated_code)
28
29
30 print(f'Function "filter_negatives" defined in the current environment.')

Re-initialized Gemini model with dynamically selected model: 'models/gemini-1.5-flash'.

Generated Function:

```
def filter_negative(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, -2, 3]
    filtered_list = filter_negative(list1)
    print(f'Original list: {list1}')
    print(f'Non-negative numbers: {filtered_list}') # Expected: [1, 0, 5, 3]
```

Variables Terminal

12:12 PM Python

Altask1-22.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

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		
<input checked="" type="checkbox"/> GOOGLE_API_KEY	AlzaSyAHiTwf57				

+ Add new secret

Gemini API keys

Access your secret keys in Python via:

```
from google.colab import userdata
userdata.get('secretname')
```

I Start coding or generate with AI.

1 Import the Python SDK
2 Import google.generativeai as gemai
3 # Used to securely store your API key
4 from google.colab import userdatas
5
6 # Retrieve the API key from Colab's secrets manager
7 GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
8 gemai.configure(api_key=GOOGLE_API_KEY)
9
10 print("Gemini API configured successfully!")

Gemini API configured successfully!

Now, let's initialize the Generative Model to use it for generating the function.

I
1 # Initialize the Gemini API
2 gemini_model = gemai.GenerativeModel("gemini-1.5-flash")
3
4 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.

I 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."
2
3 # The previous errors indicated that specific models ('gemini-1.5-flash', 'gemini-pro') were not found or supported for generateContent.
4 # To resolve this, we will dynamically find a model that supports 'generateContent'.
5
6 # Find an available model that supports generateContent
7 selected_model_name = None
8 for m in general.list_models():
9 if 'generateContent' in m.supported_generation_methods:
10 selected_model_name = m.name
11 break
12
13 if selected_model_name:
14 gemini_model = general.GenerativeModel(selected_model_name)
15 print(f'Re-initialized Gemini model with dynamically selected model: "{selected_model_name}"')
16 else:
17 raise ValueError(f"No Gemini model supporting 'generateContent' found.")
18
19 response = gemini_model.generate_content(prompt)
20
21 # Extract the generated code from the response
22 generated_code = response.text.strip().strip('python').strip()
23
24 print(f'Generated Function:\n{generated_code}')
25
26 # Execute the generated code to define the Function
27 exec(generated_code)
28
29
30 print(f'Function "filter_negatives" defined in the current environment.')

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

12:21 PM Python

Altask1-2.2.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

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	AlzaSyANiTwtf57		

+ 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 = [-3, 0, 5, -2, 10, 2.5]
    filtered_list1 = filter_nonnegatives(list1)
    print("Original list: (list1)")
    print("Non-negative numbers: (filtered_list1)") # Expected: [0, 5, 10, 2.5]

    list2 = [1, 2, 3, 4, 5]
    filtered_list2 = filter_nonnegatives(list2)
    print("Original list: (list2)")
    print("Non-negative numbers: (filtered_list2)") # Expected: [1, 2, 3, 4, 5]

    list3 = [-30, -20, -0.5]
    filtered_list3 = filter_nonnegatives(list3)
    print("Original list: (list3)")
    print("Non-negative numbers: (filtered_list3)") # Expected: []

    list4 = [0, 0, 0]
    filtered_list4 = filter_nonnegatives(list4)
    print("Original list: (list4)")
    print("Non-negative numbers: (filtered_list4)") # Expected: [0, 0, 0]

    list5 = [0, 0, 5, -3, 10, 2.5]
    filtered_list5 = filter_nonnegatives(list5)
    print("Original list: (list5)")
    print("Non-negative numbers: (filtered_list5)") # Expected: [0, 0, 5, 10, 2.5]

    original_list = [1, 2, 3, 4, 5]
    non_negative_numbers = filter_nonnegatives(original_list)
    print("Original list: (original_list)")
    print("Non-negative numbers: (non_negative_numbers)")
```

Finally, let's demonstrate the function with an example list, showing the list before and after filtering.

```
(P) 1 # Example usage
2 sensor_data = [10, -5, 0, 25, -15, 30, -2]
3
4 print("Original sensor data: (sensor_data)")
5
6 # Filter out negative numbers using the generated function
7 cleaned_data = filter_nonnegatives(sensor_data)
8
9 print("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 ✓ 12:1PM Py

Altask1-2.2.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

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	AlzaSyANiTwtf57		

+ 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.

```
1 prompt = "Write a Python function called 'filter_nonnegatives' that takes a list of numbers as input and returns a new list containing only the non-negative numbers. Include a docstring."
2
3 # The previous errors indicated that specific models ('gemini-1.5-flash', 'gemini-pro') were not found or supported for generateContent.
4 # To resolve this, we will dynamically find a model that supports 'generateContent'.
5
6 # Find an available model that supports generateContent
7 selected_model_name = None
8 for m in gemini.list_models():
9     if 'generateContent' in m.supported_generation_methods:
10         selected_model_name = m.name
11         break
12
13 if selected_model_name:
14     gemini_model = gemini.GenerativeModel(selected_model_name)
15     print(f'Re-initialized Gemini model with dynamically selected model: "{selected_model_name}"')
16 else:
17     raise ValueError("No Gemini model supporting 'generateContent' found.")
18
19 response = gemini_model.generate_content(prompt)
20
21 # Extract the generated code from the response
22 generated_code = response.text.strip('`').strip('python').strip()
23
24 print("Generated Function:")
25 print(generated_code)
26
27 # Execute the generated code to define the function
28 exec(generated_code)
29
30 print("\nFunction 'filter_nonnegatives' defined in the current environment.")
```

Re-initialized Gemini model with dynamically selected model: 'models/gemini-2.5-flash'.

Generated Function:

```
def filter_nonnegatives(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.
    """
    non_negatives = []
    for number in numbers_list:
        if number >= 0:
            non_negatives.append(number)
    return non_negatives
```

--- Example usage ---

```
if __name__ == "__main__":
    list1 = [-3, 0, 5, -2, 10, 2.5]
    filtered_list1 = filter_nonnegatives(list1)
```

Variables Terminal ✓ 12:1PM Py

The screenshot shows a Google Colab notebook interface. On the left, there's a sidebar with 'Commands', 'Code', 'Text', and 'Run all'. Below it is a 'Secrets' panel where a 'GOOGLE_API_KEY' has been stored under 'Notebook access'. The main workspace contains Python code for setting up the Gemini API and initializing a generative model. The code uses the `google-gemini` library to import the API key from Colab's secrets manager and then initializes a Gemini model named 'gemini-1.5-flash'. A docstring is provided for the function being generated.

```

# Start coding or generate with AI.

Setting up the Gemini API

To use the Gemini API, you'll need an API key. If you don't already have one, create a key in Google AI Studio. In Colab, add the key to the secrets manager under the "Secrets" in the left panel. Give it the name "GOOGLE_API_KEY". Then pass the key to the SDK.

# Import the Python SDK
2 import google.generativeai as gemai
3 # Used to securely store your API key
4 from google.colab import userdata
5
6 # Retrieve the API key from Colab's secrets manager
7 GOOGLE_API_KEY=userdata.get("GOOGLE_API_KEY")
8 gemai.configure(api_key=GOOGLE_API_KEY)
9
10 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
2 gemini_model = gemai.GenerativeModel("gemini-1.5-flash")
3
4 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."
2
3 # The previous errors indicated that specific models ('gemini-1.5-pro') were not found or supported for generateContent.
4 # To resolve this, we will dynamically find a model that supports 'generateContent'.
5
6 # Find an available model that supports generateContent
7 selected_model_name = None
8 for m in gemai.list_models():
9   if "generateContent" in m.supported_generation_methods:
10     selected_model_name = m.name
11     break
12
13 if selected_model_name:
14   gemini_model = gemai.GenerativeModel(selected_model_name)
15   print(f"\nRe-initialized Gemini model with dynamically selected model: '{selected_model_name}'")
16 else:
17   raise ValueError("No Gemini model supporting 'generateContent' found.")
18
19 response = gemini_model.generate_content(prompt)
20

```

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

The screenshot shows a Jupyter Notebook interface with two panes. The left pane contains Python code for analyzing strings, and the right pane shows the interaction with the Gemini AI model.

```

# Attempt to generate content with the first found supported model
response = gemini_model.generate_content(contents=prompt)
break # Exit loop once content is successfully generated
except Exception as e:
    print(f"Error with model [{m.name}]: {e}")
else:
    print(f"Model [{m.name}] does not support generateContent.")

```

```

```python
import string

def analyze_string(input_string: str) -> dict:
 """
 Analyzes a given string to count vowels, consonants, and digits.

 Args:
 input_string (str): The string to be analyzed.

 Returns:
 dict: A dictionary containing the counts of 'vowels', 'consonants', and 'digits'.
 """
 vowel_count = 0
 consonant_count = 0
 digit_count = 0

 # Define a set of vowels for efficient case-insensitive checking
 vowels = {'a', 'e', 'i', 'o', 'u'}

 for char in input_string:
 if char.isalpha():
 char_lower = char.lower() # Convert to lowercase for consistent checking
 if char_lower in vowels:
 vowel_count += 1
 else:
 consonant_count += 1
 elif char.isdigit():
 digit_count += 1
 # Other characters (spaces, punctuation, symbols) are ignored

 return {
 "vowels": vowel_count,
 "consonants": consonant_count,
 "digits": digit_count
 }

--- Sample Inputs and Demonstration ---

print("--- Sample Input 1 ---")
sample_string_1 = "Hello World! 123"
result_1 = analyze_string(sample_string_1)
print(f"Input String: '{sample_string_1}'")
print(f"Vowels: {result_1['vowels']} (H, e, o, o), consonants: {result_1['consonants']} (l, l, l, W, r, d, !), digits: {result_1['digits']} (1, 2, 3)")

Expected: vowels: 3 (e, o, o), consonants: 7 (H, l, l, l, W, r, d), digits: 3 (1, 2, 3)

print("\n--- Sample Input 2 ---")
sample_string_2 = "Python Programming 2023"
result_2 = analyze_string(sample_string_2)
print(f"Input String: '{sample_string_2}'")

```

The right pane shows the Gemini interface with the following text:

Please explain this error:  
 ⓘ Network: 404 POST https://generativelanguage.googleapis.com/v1beta/models/gemini-1.0-pro  
 ⚡ The error indicates that the `gemini-1.0-pro` model is not found or supported for content generation. I'll update the code to use the `gemini-1.0-pro`, which is a commonly available and stable model.  
 The `gemini-1.0-pro` model seems to be unavailable or unsupported in your current setup. I'll modify the code to list all available models and then filter for those that support content generation. This way, we can identify a model that works in your environment and update the code accordingly.  
 It seems that the `gemini-1.0-pro` model is still not found or supported. To resolve this recurring issue, I'll modify the code to list all available models and then filter for those that support content generation. This way, we can identify a model that works in your environment and update the code accordingly.  
 The code has successfully generated a Python function named `analyze_string` using the `gemini-2.5-flash` model. This function counts vowels, consonants, and digits in given string. The output also includes several examples demonstrating how to use the function and the results for various inputs.

This screenshot shows the same Jupyter Notebook interface as the previous one, but the code has been modified to use the `gemini-1.0-pro` model.

```

1 def analyze_string(input_string):
2 vowel_count = 0
3 consonants_count = 0
4 digits_count = 0
5
6 vowels = {'a', 'e', 'i', 'o', 'u'}
7
8 for char in input_string:
9 char_lower = char.lower()
10 if 'a' <= char_lower <= 'z': # Check if it's an alphabet character
11 if char_lower in vowels:
12 vowel_count += 1
13 else:
14 consonants_count += 1
15 elif '0' <= char <= '9': # Check if it's a digit character
16 digits_count += 1
17
18 return {
19 "vowels": vowel_count,
20 "consonants": consonants_count,
21 "digits": digits_count
22 }
23
24 # Sample inputs and demonstration
25 sample1 = "Hello World!"
26 sample2 = "Python is fun 123"
27 sample3 = "XEDOU ERBD"
28 sample4 = "1234567890"
29
30 print("Analyzing '{sample1}': (analyze_string(sample1))")
31 print("Analyzing '{sample2}': (analyze_string(sample2))")
32 print("Analyzing '{sample3}': (analyze_string(sample3))")
33 print("Analyzing '{sample4}': (analyze_string(sample4))")
34

```

### 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 <code>string == string[::-1]</code> (slicing) for simplicity.	Often provides more "robust" versions, sometimes using <code>reversed()</code> 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

```
C:\> hubo > hp > AppData\Local\Packages\5119275A.What'sAppDesktop_cvggrngpm\LocalState\7\sessions\3CM9Y6C909CAF1F0480F7A2694FA88F4DC99C> transfers> 2028-02 > ⚡ assign-22303A3111B:copilot.py > ...
1 #Task 3: Palindrome Check
2 v def is_palindrome(s):
3 return s == s[::-1]
4
5 # Test cases
6 print(is_palindrome("man a plan a canal Panama")) # True
7 print(is_palindrome("racecar")) # False
8 Comparison: Gemini AI vs Copilot
9 1 Coding Performance:
10 >Copilot is generally better integrated into IDEs for fast code suggestions and developer workflows.
11 >Gemini can generate code too, but tends to be slower and broader-focused rather than as tightly embedded into editor workflows.
12 Winner for pure coding tools: Copilot
13
14 2 Context & Reasoning:
15 >Copilot excels at reasoning, large context understanding (docs + code + research), and handling multimodal inputs like images/diagrams.
16 >Copilot is optimized for shorter, code-centric tasks within text editors.
17 Winner for complex tasks: Gemini
18
19 3 Integration & Workflow:
20 >Copilot works directly in popular editors with minimal setup.
21 >Gemini can integrate through CLI or APIs and into Google's ecosystem, but isn't as IDE-native by default (though integrations are growing).
22 Winner for seamless coding experience: Copilot
23
24 4 Multimodal & General Intelligence:
25 >Gemini supports multimodal input (text, visuals) and broader tasks like research, explanations, documentation, strategy, etc.
26 >Copilot focuses on code generation and in-editor support only.
27 Winner for versatility: Gemini
28
29 5 Pricing & Value:
30 >Copilot is usually around $10/mo for individuals and integrated deeply with GitHub tools.
31 >Gemini varies by plan but often comes bundled with Google services (Cloud, Workspace), and free tiers like Gemini CLI offer generous daily usage.
32 Winner depends on budget & ecosystem needs
33
34 6 So Which One Is Better?
35 Use Case Better Choice:
36 >Daily coding, fast suggestions inside your editor GitHub Copilot
37 >Complex tasks: reasoning, docs, long projects, multimodal input Google Gemini
38 >Working within GitHub & developer workflow Copilot
39 >Handling diverse non-code tasks or big context workflows Gemini
40
41 7 Many developers actually use both:
42 >Copilot for IDE coding speed,
43 >Gemini for deep questions, research, design docs, and multimodal tasks.
44
45 8 Quick Summary:
46 >Copilot Best code-focused assistant with strong editor integration and real-time code generation.
47 >Gemini Best general AI partner for big-picture thinking, multimodal understanding, and broad tasks beyond code.
```