

2303A51415

Batch-03

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 a Google Colab interface with a Jupyter notebook titled "Untitled8.ipynb". A single cell is active, containing the following Python code:

```
def clean_iot_sensor_data(sensor_data_list):
    """
    Cleans a list of IoT sensor data by removing all negative values.

    Args:
        sensor_data_list (list): A list of numerical sensor readings.

    Returns:
        list: A new list containing only non-negative sensor readings.

    cleaned_data = [value for value in sensor_data_list if value >= 0]
    return cleaned_data

    # Example usage:
    raw_sensor_readings = [23.5, -5.2, 24.1, 0.0, -1.8, 22.9, 30.7]
    cleaned_readings = clean_iot_sensor_data(raw_sensor_readings)

    print("Raw sensor readings:", raw_sensor_readings)
    print("Cleaned sensor readings:", cleaned_readings)

    ... Raw sensor readings: [23.5, -5.2, 24.1, 0.0, -1.8, 22.9, 30.7]
    cleaned sensor readings: [23.5, 24.1, 0.0, 22.9, 30.7]
```

The output of the cell shows the raw sensor readings and the cleaned sensor readings.

give me the function for cleaning iot sensor data where negative values are invalid it should filter all the negative values from the list

Here's a Python function that filters out negative values from a list of IoT sensor data. I've also included an example of how to use it.

The code in cell 9c65c983 successfully executed the

plain the `clean_iot_sensor_data` function

Plot the `cleaned_readings`

How can I app

Gemini 2.5 Flash ▾ ➤

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 Google Colab notebook window. The code cell contains the following Python function:

```
vowels = "aeiouAEIOU"
vowel_count = 0
consonant_count = 0
digit_count = 0

for char in input_string:
    if char.isalpha(): # check if the character is an alphabet
        if char in vowels:
            vowel_count += 1
        else:
            consonant_count += 1
    elif char.isdigit(): # check if the character is a digit
        digit_count += 1

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

# Example usage:
text = "Hello World 123!"
counts = count_chars(text)
print("Original string: '{text}'")
print(f"Counts: {counts}")

text_2 = "Python is fun 4 everyone!"
counts_2 = count_chars(text_2)
print("Original string: '{text_2}'")
print(f"Counts: {counts_2}")
```

The output cell shows the results of running the function with two examples:

```
... Original string: 'Hello World 123!'
counts: {'vowels': 3, 'consonants': 7, 'digits': 3}
Original string: 'Python is fun 4 everyone!'
Counts: {'vowels': 6, 'consonants': 12, 'digits': 4}
```



generate a python function that counts vowels consonets an d digits in a string



Here's a Python function that counts the number of vowels, consonants, and digits in a given string. I've also included an example of how to use it.

[Apply `count_chars` to a new string](#)

[Explain the output of `count_chars` function](#)

[How c](#)

What can I help you build?



Gemini 2.5 Flash ▾ ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

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

CODE

```
import google.generativeai as genai
import google.colab.userdata

try:
    # Configure the genai library with your API key
    GOOGLE_API_KEY = google.colab.userdata.get('GOOGLE_API_KEY')
    genai.configure(api_key=GOOGLE_API_KEY)

    # Initialize the Gemini model
    model = genai.GenerativeModel('gemini-pro')

    print("Gemini API configured and model initialized successfully.")

except google.colab.userdata.SecretNotFoundError:
    print("Error: GOOGLE_API_KEY not found in Colab secrets.")
    print("Please add your GOOGLE_API_KEY to Colab secrets to proceed.")
    print("Steps to add a secret:")
    print("1. Click on the 'key' icon (Secrets) in the left sidebar of Google Colab.")
    print("2. Click '+ New secret'.")
    print("3. For 'Name', enter 'GOOGLE_API_KEY'.")
    print("4. For 'Value', paste your Gemini API key.")
    print("5. Ensure 'Notebook access' is toggled ON for this notebook.")
except Exception as e:
    # Catch any other unexpected errors during configuration
    print(f"An unexpected error occurred during Gemini API configuration: {e}")
```

PROMPT

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

::::

palindrome_compare.py

Contains two palindrome checkers:

- gemini_is_palindrome: normalizes Unicode (NFKD), strips non-alphanumerics, compares reversed string
- copilot_is_palindrome: two-pointer scan, compares characters case-insensitively without Unicode normalization

A main() runs several test cases and reports where the functions disagree.

"""

```
from __future__ import annotations

import re
import unicodedata
from typing import List, Tuple

def gemini_is_palindrome(s: str) -> bool:
    """Gemini-style palindrome checker.

    - Normalizes Unicode using NFKD (so accented characters compare to their base letters)
    - Keeps only alphanumeric characters
    - Compares cleaned string to its reverse
    """

    if s is None:
        return False
```

```
# Unicode normalization separates base characters from diacritics
normalized = unicodedata.normalize("NFKD", s)
cleaned = "".join(ch for ch in normalized if ch.isalnum()).lower()
return cleaned == cleaned[::-1]
```

```

def copilot_is_palindrome(s: str) -> bool:
    """Copilot-style palindrome checker.

    - Uses two-pointer technique
    - Skips non-alphanumeric characters
    - Case-insensitive comparison
    - Does NOT perform Unicode decomposition/normalization
    """
    if s is None:
        return False
    i, j = 0, len(s) - 1
    while i < j:
        while i < j and not s[i].isalnum():
            i += 1
        while i < j and not s[j].isalnum():
            j -= 1
        if s[i].lower() != s[j].lower():
            return False
        i += 1
        j -= 1
    return True

```

```

TEST_CASES: List[Tuple[str, str]] = [
    ("", "empty string"),
    ("a", "single char"),
    ("Abba", "case-insensitive palindrome"),
    ("A man, a plan, a canal: Panama", "classic phrase"),
    ("No 'x' in Nixon", "punctuation and spaces"),
    ("Not a palindrome", "non-palindrome"),
    ("été", "accented palindrome (" + "NFKD-normalized -> 'ete'" + ")"),
    ("ÀbbaA", "accented A vs A"),
]

```

```

("😊a😊", "emoji around single letter"),
]

def main() -> None:
    print("Comparing Gemini vs Copilot palindrome checkers\n")

    mismatches = []
    for s, note in TEST_CASES:
        g = gemini_is_palindrome(s)
        c = copilot_is_palindrome(s)
        ok = g == c
        status = "MATCH" if ok else "DIFFER"
        print(f"{status:7} | Gemini: {g!s:5} | Copilot: {c!s:5} | {note:30} | {s!r}")
        if not ok:
            mismatches.append((s, note, g, c))

    print("\nSummary:")
    if not mismatches:
        print("✅ All outputs match for the test cases.")
    else:
        print(f"⚠️ Found {len(mismatches)} disagreement(s):")
        for s, note, g, c in mismatches:
            print(f" - {note}: Gemini={g} Copilot={c} for input {s!r}")

if __name__ == "__main__":
    main()

OUTPUT:
MATCH | Gemini: True | Copilot: True | empty string      |
MATCH | Gemini: True | Copilot: True | single char       | 'a'
MATCH | Gemini: True | Copilot: True | case-insensitive palindrome | 'Abba'

```

MATCH Gemini: True Copilot: True classic phrase Panama'	'A man, a plan, a canal:
MATCH Gemini: True Copilot: True punctuation and spaces	"No 'x' in Nixon"
MATCH Gemini: False Copilot: False non-palindrome	'Not a palindrome'
MATCH Gemini: True Copilot: True accented palindrome (NFKD-normalized -> 'ete') 'été'	
MATCH Gemini: False Copilot: False accented A vs A	'ÂbbaA'
MATCH Gemini: True Copilot: True emoji around single letter	'😊a😊'

Summary:

- ✓ All outputs match for the test cases.

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

CODE

```
def is_palindrome(text):
    """
    Checks if a given string is a palindrome.

    A palindrome is a word, phrase, number, or other sequence of
    characters
    which reads the same backward as forward.

    This function ignores spaces, punctuation, and capitalization.
    """
```

Args:

text (str): The string to check.

Returns:

bool: True if the string is a palindrome, False otherwise.

"""

```
# Remove non-alphanumeric characters and convert to lowercase
cleaned_text = ''.join(char.lower() for char in text if
char.isalnum())

# Compare the cleaned string with its reverse
return cleaned_text == cleaned_text[::-1]

# Example usage:
print(f"'madam' is a palindrome: {is_palindrome('madam')}")  
print(f"'A man, a plan, a canal: Panama' is a palindrome:  
{is_palindrome('A man, a plan, a canal: Panama')}")  
print(f"'hello' is a palindrome: {is_palindrome('hello')}")  
print(f"'Racecar' is a palindrome: {is_palindrome('Racecar')}")
```

PROMPT

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