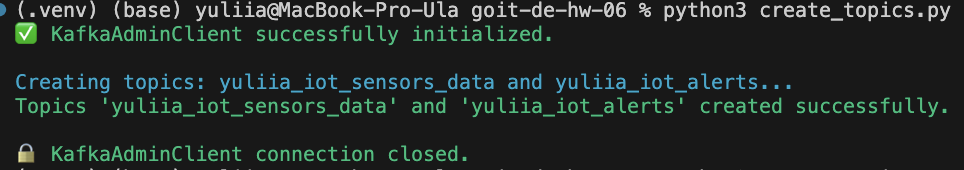
**Домашнє завдання до теми «Spark streaming»**

**Create\_topics.py  
*1. Ініціалізація KafkaAdminClient:***

* Код створює клієнт KafkaAdminClient для керування темами у Kafka.
* Параметри клієнта передаються з конфігураційного файлу kafka\_config (наприклад, bootstrap\_servers, security\_protocol, sasl\_mechanism тощо).
* Використовується обробка винятків для перевірки успішності підключення. Якщо підключення не вдалося, скрипт виводить червоне повідомлення про помилку та завершує виконання.

Результат: У виводі видно, що клієнт було успішно ініціалізовано (зелене повідомлення).

***2. Визначення імен тем:***

* Імена тем формуються динамічно на основі змінної my\_name ("yuliia"):
  + yuliia\_iot\_sensors\_data
  + yuliia\_iot\_alerts
* Встановлюється кількість партицій (2) та коефіцієнт реплікації (1).

***3. Перевірка існування тем:***

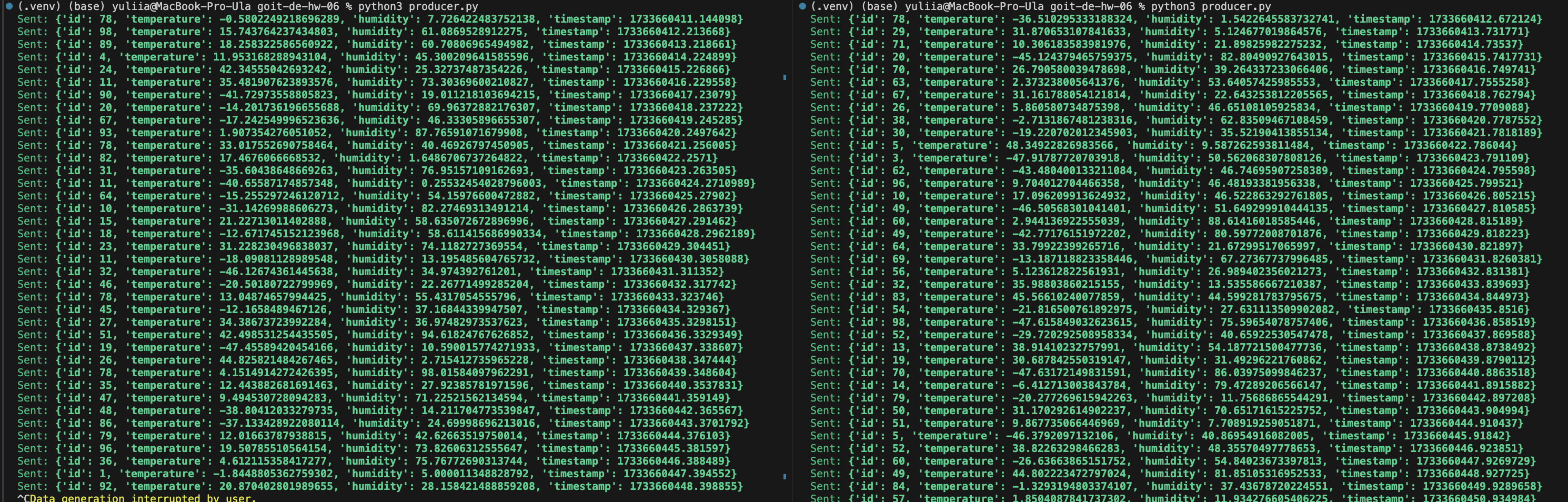
* Використовується метод list\_topics(), щоб отримати список уже створених тем.
* Якщо теми вже існують, скрипт виводить попередження жовтим кольором.

***4. Створення тем:***

* Якщо теми ще не існують, створюється список об'єктів NewTopic для нових тем.
* Метод create\_topics() створює теми на кластері Kafka.
* Результат створення відображається у виводі (зелений текст для успішного створення або червоний для помилок).

***5. Закриття клієнта:***

* Метод close() закриває підключення до Kafka, щоб звільнити ресурси.
* Успішне закриття підтверджується зеленим повідомленням.

**Producer.py  
*1. Ініціалізація Kafka Producer:***

* + Створюється об'єкт KafkaProducer для відправлення повідомлень до Kafka.
  + Параметри підключення (сервери, автентифікація, серіалізація) беруться з конфігураційного файлу kafka\_config.
  + Серіалізація повідомлень виконується за допомогою json.dumps, дані кодуються у формат UTF-8.

***2. Генерація даних:***

* + У нескінченному циклі генеруються випадкові дані:
    - id: випадкове число від 1 до 100.
    - temperature: випадкове число в діапазоні від -50 до 50 (°C).
    - humidity: випадкове число в діапазоні від 0 до 100 (%).
    - timestamp: поточний час у секундах із моменту Unix-епохи (time.time()).
  + Повідомлення надсилається до теми yuliia\_iot\_sensors\_data.

***3. Вивід даних у консоль:***

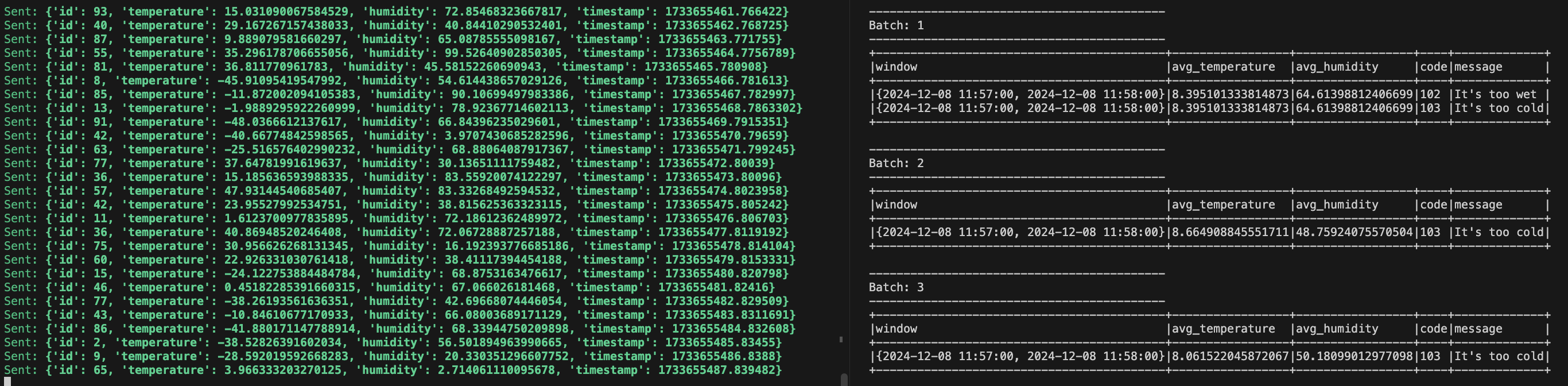
* + Для кожного повідомлення виводиться інформація у форматі JSON (зелений текст) з позначкою Sent:.

***4. Інтервал між повідомленнями:***

* + Інтервал у 1 секунду задається функцією time.sleep(1).

***5. Обробка завершення скрипту****:*

* + Скрипт завершується вручну (KeyboardInterrupt), після чого виводиться повідомлення про завершення (жовтий текст).
  + Після завершення з'єднання з Kafka закривається (producer.close()).

**Kafka\_iot\_alerts\_stream.py**Зображено вивід двох частин програми:

***1. Ліва частина****:*

* + Вивід із Kafka-продюсера (producer.py), який генерує повідомлення у JSON-форматі та надсилає їх у топік Kafka (yuliia\_iot\_sensors\_data).

***2. Права частина:***

* Вивід Spark-стримінгу, який обробляє дані із Kafka.
* **Агрегація**:
  + Дані групуються у вікна тривалістю 1 хвилина, для кожного з яких обчислюється:
    - avg\_temperature — середня температура.
    - avg\_humidity — середня вологість.
* **Алерти**:
  + Для кожного вікна перевіряються умови з файлу alerts\_conditions.csv. Наприклад:
    - Якщо значення температури або вологості виходять за межі заданих діапазонів, генерується алерт із кодом (code) і повідомленням (message).
  + У прикладі коди 102 і 103 відповідають повідомленням про надмірну вологість чи холод.

**Загальний опис роботи системи:**

1. **Джерело даних**:
   * Kafka-продюсер передає потік даних сенсорів (IoT).
2. **Обробка**:
   * Spark читає потік із Kafka, обробляє його у реальному часі та виконує агрегацію за допомогою вікон.
3. **Умови для алертів**:
   * Зчитуються із CSV-файлу і застосовуються до агрегаційних даних.
4. **Вивід результатів**:
   * Алерти виводяться у консоль і передаються до Kafka (в топік yuliia\_iot\_alerts).