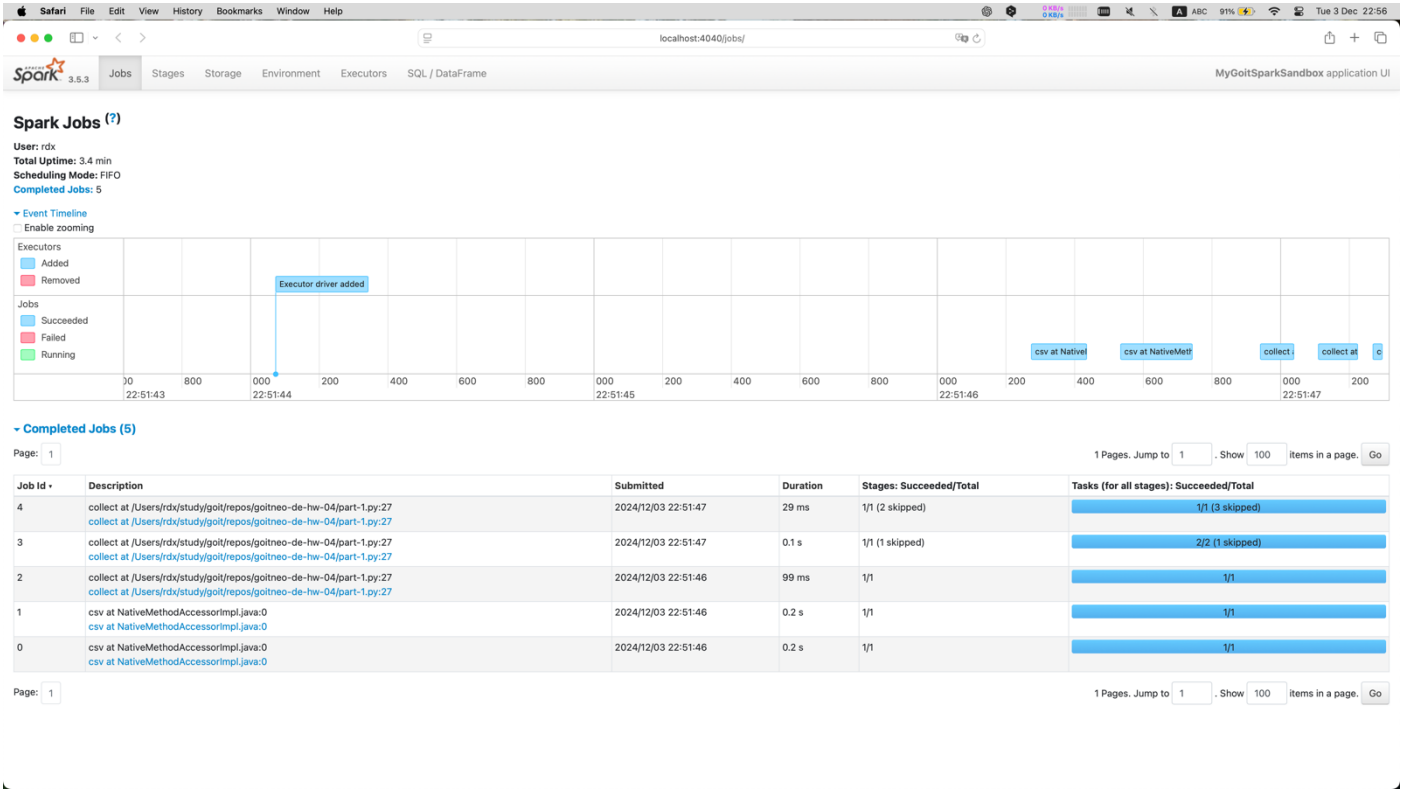
1. **Аналіз кількості Jobs у кожному скрипті:**

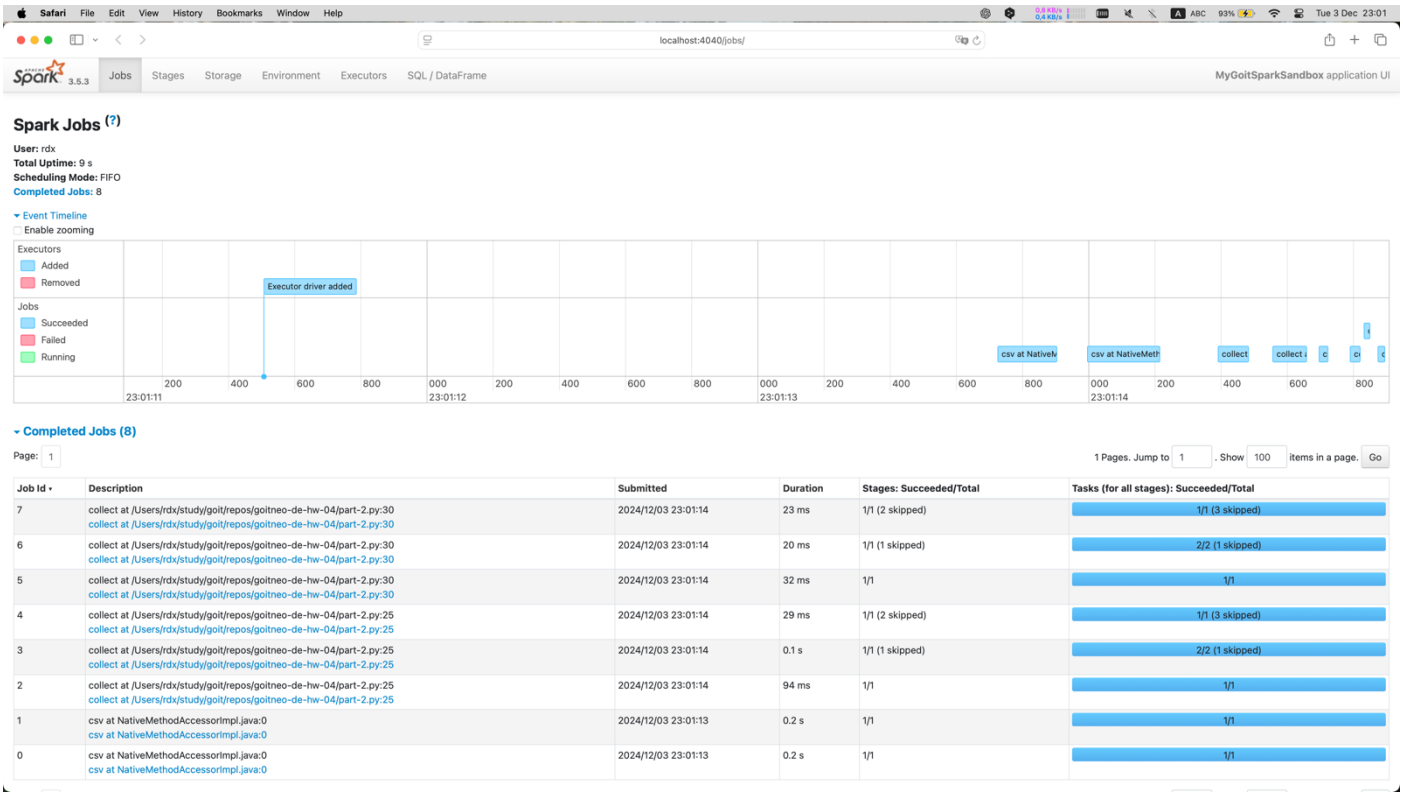
   - **part-1.py**:

У скрипті виконується лише один action (collect) після обробки даних (nuek_processed). Цей action запускає обчислення всього ланцюжка трансформацій (наприклад, where, select, groupBy, count). Тому кількість Jobs мінімальна (5 Jobs, як на скриншоті).



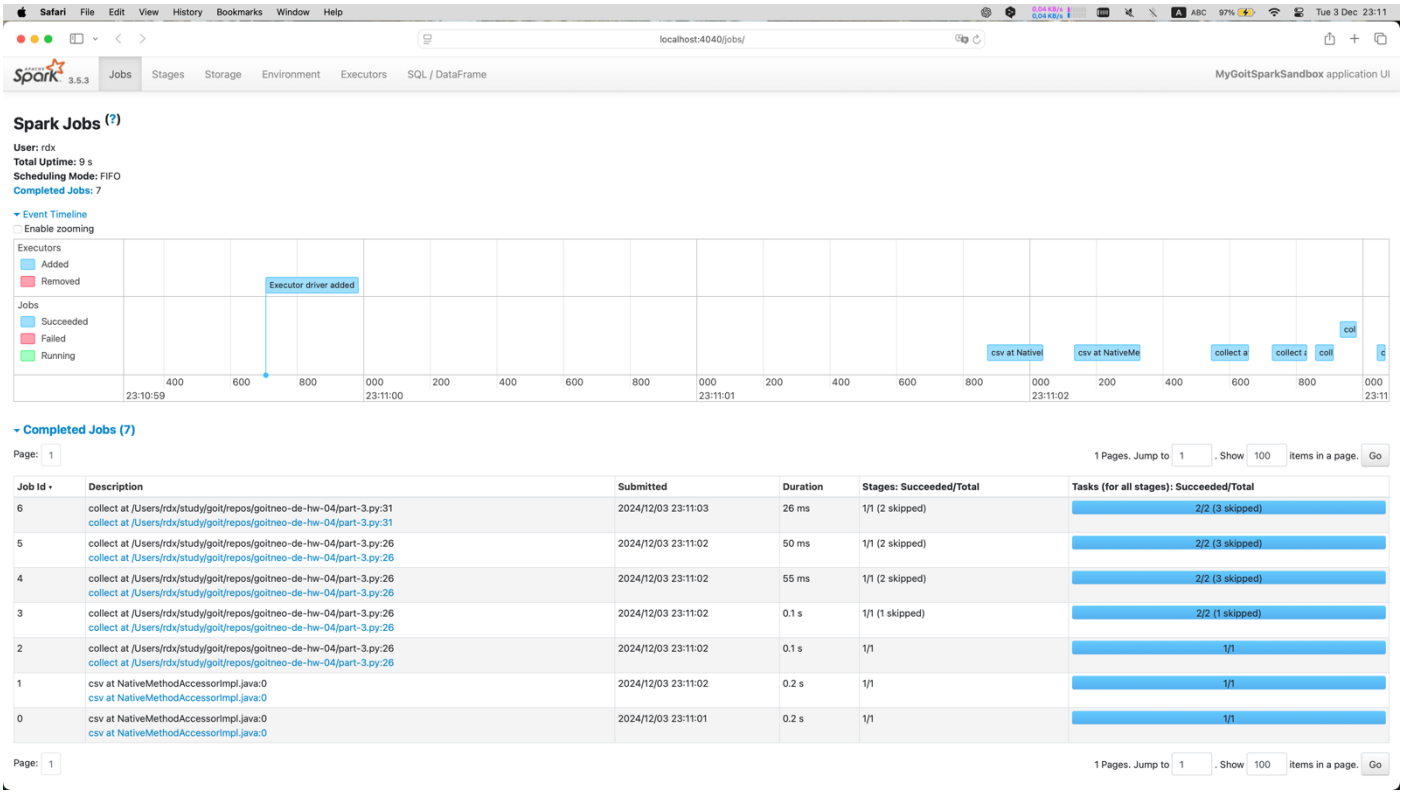
- **part-2.py**:

У цьому скрипті додано проміжну дію collect перед додаванням фільтру where("count > 2"). Це викликає два окремих обчислення (дві дії collect), що розбиває ланцюжок виконання на два етапи. Кожен collect запускає новий Job, тому загальна кількість Jobs зросла до 8.



- **part-3.py**:

У цьому скрипті використовується функція cache(), яка кешує результати DataFrame (nuek_processed_cached) в пам’яті. Завдяки кешуванню повторне використання цього DataFrame (включно з другим collect) виконується без необхідності перераховувати попередні трансформації, що зменшує кількість Job до 7.



2. **Функція *cache* та її використання:**

- **Що робить *cache*:**

cache зберігає результати обчислень DataFrame у пам’яті (або на диску, якщо пам’яті недостатньо). Це дозволяє уникнути повторного виконання всієї послідовності трансформацій під час повторного звернення до DataFrame.

- **Навіщо її використовувати:**

Функція корисна, коли:

- Один і той самий DataFrame використовується декілька разів у різних частинах коду (наприклад, двічі викликається collect).
- Потрібно зменшити витрати на обчислення, особливо якщо трансформації дорогі.
- Зниження кількості Jobs підвищує продуктивність і зменшує затримки.

У прикладі part-3.py, завдяки кешуванню, трансформації до nuek_processed_cached виконуються лише один раз. Це значно знижує кількість Jobs у порівнянні з part-2.py, де ті самі обчислення виконувались повторно.