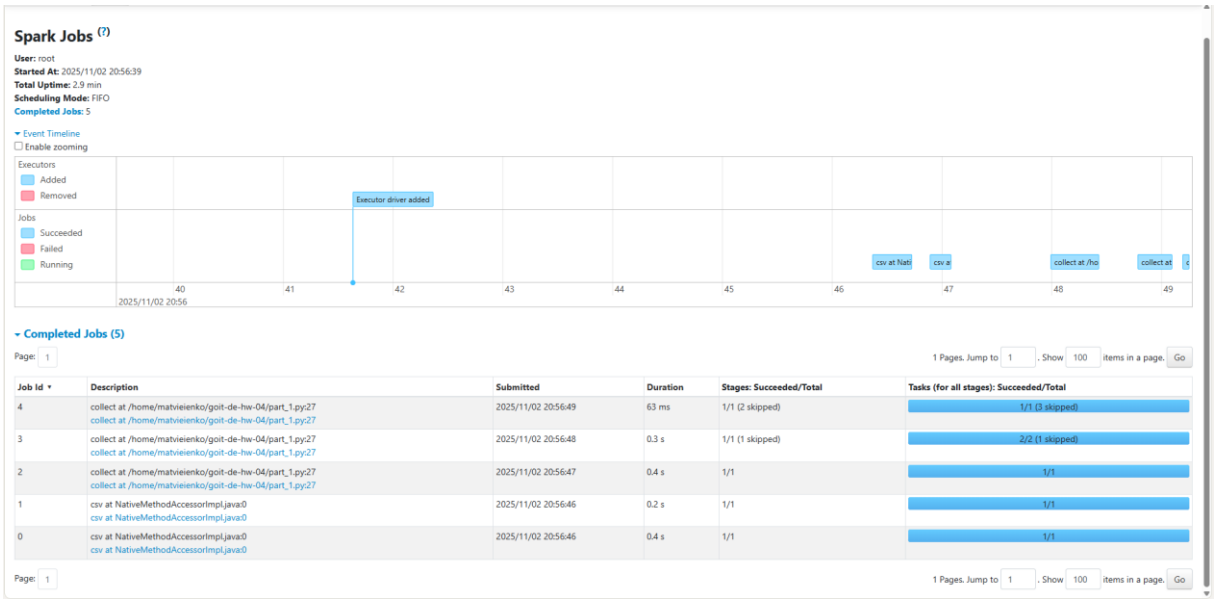
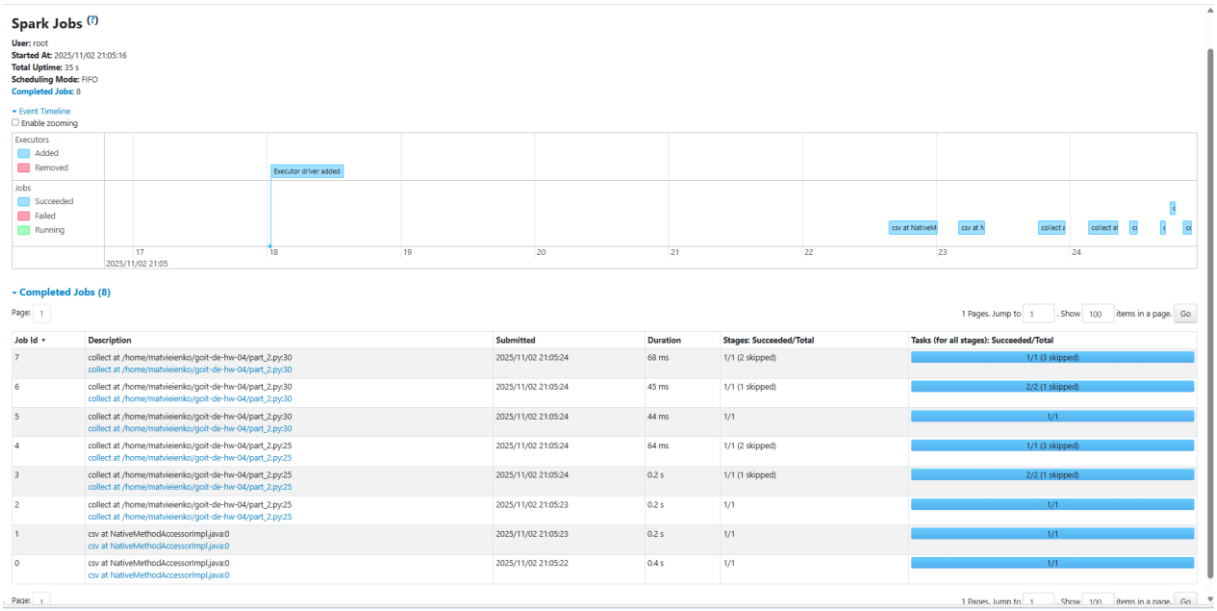
Домашнє завдання 4: Аналіз Spark Jobs



Частина 1 — (5 Jobs)

1. Job 0: Читання CSV-файлу у DataFrame (spark.read.csv), розподіл даних по партиціях.
2. Job 1: Репартіціонування даних (repartition(2)) — переналаштування розподілу між виконавцями.
3. Job 2: Вибірка потрібних колонок, фільтрація та агрегація (where, select, groupBy).
4. Job 3: Групування та підрахунок (операція count після groupBy).
5. Job 4: Дія collect() — виконання всіх трансформацій та збір результату у драйвер-програму.

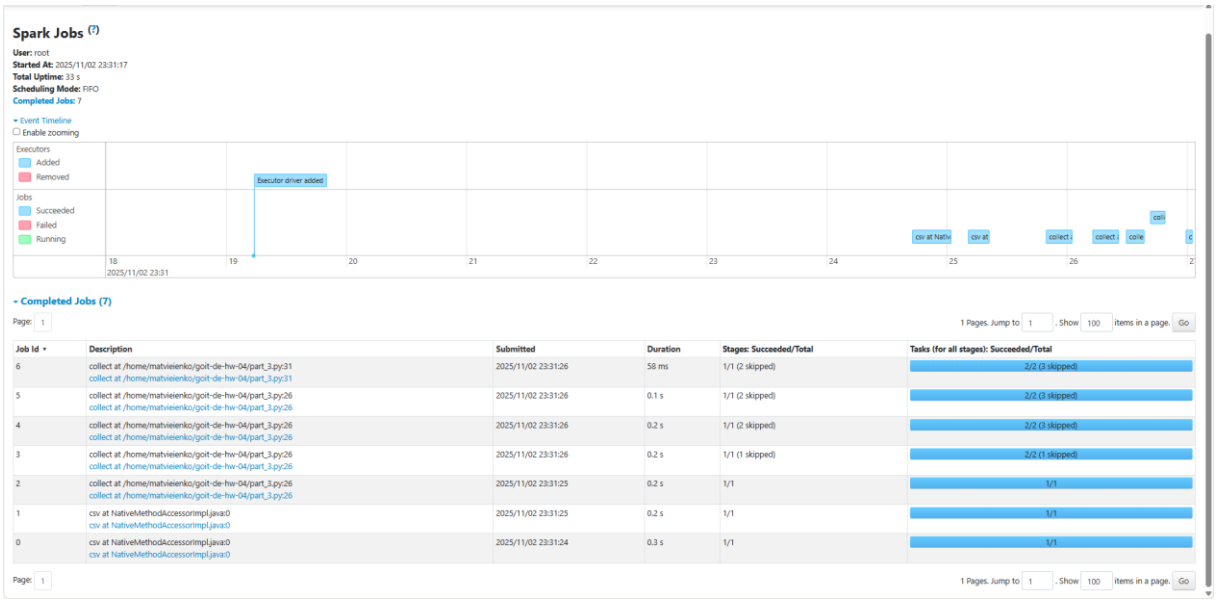
Оскільки в коді лише одна Action-операція, Spark виконує всі трансформації послідовно, створюючи 5 окремих Jobs.



Частина 2 — (8 Jobs)

1–5. Job 0–4: Повторюють усі етапи першої частини - читання, репартіціонування, трансформації, агрегація, збір даних. 6–8. Job 5–7: Друга Action (collect()) без кешу — Spark знову проводить усі стадії трансформацій, бо дані не збережені в оперативній пам’яті.

Дві Action-операції без cache змушують Spark двічі проганяти весь граф обчислень — це збільшує кількість Jobs.



Частина 3 — (7 Jobs)

1–5. Job 0–4: Виконання першої Action-операції та кешування DataFrame (cache()).
6–7. Job 5–6: Друга Action-операція (collect()) бере дані із кешу, тому повний ланцюжок DAG вже не проганяється заново.

Кешування дозволяє використовувати збережені в пам’яті дані для повторних дій — це зменшує кількість додаткових Jobs при наступному зверненні.

Висновок:

- Кожна Job у Spark відповідає окремому етапу плану виконання: читання даних, зміну розподілу, трансформації, агрегацію, збір результату.
- Повторні дії без кешу створюють нові jobs для кожної операції.
- Використання .cache() істотно скорочує кількість jobs при повторних зверненнях, оптимізуючи виконання.