**Завдання 1: Базовий код з однією дією collect()**

**Опис:**

* Код виконує базову обробку датасету nuek-vuh3.csv, включаючи завантаження, фільтрацію, групування та підрахунок.
* В кінці додається умова фільтрації та викликається метод collect().

A screenshot of a computer

Description automatically generated

**Кількість Jobs:**

* **5 Jobs**.
* Кожна операція, яка виконується в Spark, створює нове завдання (Job). Виконання collect() примушує Spark обчислити всі необхідні трансформації для отримання результатів.

A screenshot of a computer

Description automatically generated

A white background with many small colored lines

Description automatically generated with medium confidence

**Завдання 2: Додана проміжна дія collect()**

**Опис:**

* Додано проміжну дію collect() перед фільтрацією результатів за умовою count > 2.
* Цей код спочатку отримує всі попередньо обчислені результати, а потім повторно виконує додаткові обчислення. A screenshot of a computer program

  Description automatically generated

**Кількість Jobs:**

* **8 Jobs**.
* Додавання проміжного collect() змусило Spark виконати нові обчислення для отримання даних перед фільтрацією. Це призвело до збільшення кількості Job.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Завдання 3: Використання cache()**

**Опис:**

* Додано функцію cache() для проміжного DataFrame. Це зберігає результати обчислень у пам’яті для повторного використання.
* Після кешування виконується дві дії: перше collect() і додаткове фільтрування з повторним викликом collect().A screenshot of a computer screen

  Description automatically generated

**Кількість Jobs:**

* **7 Jobs**.
* Кешування дозволяє уникнути повторного обчислення проміжних даних, що зменшує кількість Job.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Висновок:**

1. Використання cache() оптимізує виконання, оскільки зберігає результати проміжних обчислень у пам’яті.
2. Додавання проміжної дії, такої як collect(), збільшує кількість Job, оскільки змушує Spark виконувати додаткові обчислення.
3. Правильне використання трансформацій та кешування дозволяє мінімізувати кількість обчислень і підвищити продуктивність.