Dataset Analyzer

The Problem

The Open University Learning Analytics Dataset (OULAD) contains real data collected from students in online courses at the Open University in the UK. The dataset includes information about each student's performance in the course and their interactions with online materials. This dataset has become a benchmark for learning analytics researchers interested in developing tools to support online learning, such as early warning systems to identify students at risk of failing a course.

Like many datasets gathered from online systems, some of the data files are huge, making them slow and cumbersome to analyze. For example, one file in the OULAD contains over 10.6 million rows and is so big it cannot be fully opened in standard spreadsheet software.

Your task is to write a program that will help automate analysis of the OULAD files. Download the OULAD files and review the data description at the link above. You will work with courses.csv (a small file) and studentVle.csv (a huge file). All files contain header rows.

The format of the data in courses.csv is: code_module,

code_presentation, code_presentation_length Where:

- code_module . A module is equivalent to a course code, in Northeastern terms. For example, CS5010 would be a course_module.
- code_presentation . A presentation is equivalent to an offering of a course. For example, Spring2023 would be a course_presentation.
- code_presentation_length is the length of the course offering in days. (courses at the Open University last a lot longer than courses at Northeastern).

The format of the data in studentVle is:

code_module, code_presentation, id_student, id_site, date, sum_click Where:

- code_module and code_presentation are the same as in courses.csv. studentVle.csv only contains data from courses included in courses.csv.
- id_student is the unique ID number of a student in the given course.
- id_site is the unique ID number of a particular online resource.

- date is a specific day within the course, relative to the course start date. For example, -10 means 10 days before the course started.
- sum_click is the number of times the given student clicked on the given resource on the given day.

For both parts below, you will need to write a program that will read in these files and use them to create new summary files so that each individual code_presentation of a code_module has a file listing the total number of clicks that occurred on a particular day. Each new file should be named code_presentation_code_module.csv and should contain two columns: date, and total_clicks.

For example, one code_module is "AAA". It has two code_presentations, one of which is "2013J" and the other is "2014J". Therefore, your program should produce two csv files for this module: AAA_2013J.csv and AAA_2014J.csv. Within each file, each row should contain one date and the total number of clicks that occurred in the presentation across all id_sites and all id_students. For example, one row in "AAA_2013J.csv" should contain -10, 11952, meaning that there were 11952 clicks in this course presentation on day -10.

Part 1: Sequential solution

Create a package called sequentialSolution and write a sequential solution to the above problem. Your main class should accept the directory containing the OULAD csv files as a command line argument. You may hardcode the names of the two OULAD csv files: courses.csv and studentVle.csv.

For testing, you will probably find it helpful to create a smaller version of studentVle.csv!

Part 2: Concurrent solution

Create a package called concurrentSolution. Redesign your solution to part 1 to use threads in a "producer-consumer" style. The inputs and outputs will be the same but your code now contains multiple threads.

For example, when you read from studentVle.csv, you can think of this as a producer that simply knows how to read this given file type and write each line "somewhere" for another thread, the consumer, to process. Consumers can read the contents and transform them into the results required and write these "somewhere" for maybe another thread to write the output.

Your solution must have producer and consumer threads running concurrently and communicating via some suitable shared data structure. Producers and consumers should be loosely coupled and communicate only through some queue or channel.

Part 3: Identify high activity days

Modify your concurrent solution to take an additional integer value on the command line, which we'll call an activity "threshold". The threshold is a value that the user wants to use to identify days when each course had the most activity.

You should use the threshold as follows:

1. Process the summary files produced above to find each day in each file where the total_clicks is >= the threshold value.
2. Create a new file called activity-threshold.csv, where "threshold" is the value entered by the user that contains all rows identified in step 1. Your file will also need to indicate which module and presentation the data came from, so it should contain the following columns:
   module_presentation, date, total_clicks
   So, for example, if the threshold entered by the user was 11000, the file created would be called activity-11000.csv and one row in the file would contain the following data:
   AAA_2013J, -10, 11952, where -10 is the date and 11952 is the total number of clicks on that day.