Software Developer (Data Pipelines)

PYTHON EXERCISE

The aim of this exercise is to make a Python script that can efficiently query large files and aggregate the results.

Assume that there is a directory called logs, inside you can find daily CSV files in the following naming scheme:

/logs/2025-01-01.log.csv
/logs/2025-01-02.log.csv
/logs/2025-01-03.log.csv
...
/logs/2025-02-01.log.csv

Note: Not all daily files will necessarily be present.

Each file will have the same columns:

timestamp (in the form "2025-01-01 00:23:45")
user (a string)
app (a string, can contain spaces)
metric1 to metric9 (integer values)

And will look like this:

timestamp	user	app	metric_1	metric_2	...	metric_9
2025-01-01 00:23:45	user1	facebook	55	23	...	31
2025-01-01 00:27:11	user1	youtube	53	442	...	112
2025-01-01 00:38:45	user1	facebook	66	245	...	11
2025-01-01 00:12:45	user2	twitter	33	212	...	12
2025-01-01 01:23:45	user3	facebook	622	23	...	42
...	...	...	...	...	...	...

Each user can have rows about their metrics for multiple apps, and can have multiple rows for the same app in the same day as well.

Write a python script that reads these logs as needed to answer to filter and aggregate the data with the options:

--from-datetime YYYY-mm-DD HH:MM:SS
--to-datetime YYYY-mm-DD HH:MM:SS
--user <user id>
--app <app name>
--granularity=<30m or 1day>
--dimensions=<user or app or user,app>

The --dimensions argument determines which of the two columns "user" and "app" to keep (like GROUP BY in SQL).

The granularity argument determines by which interval to aggregate. For example if --granularity=30m, you need to aggregate the data so that each row represents aggregated rows for all timestamps for each 30m. For an aggregated row, use the start of the 30m or 1d period (depending on granularity requested) as the timestamp.

timestamp	user	app	metric_1	metric_2	...	metric_9
2025-01-01 00:23:45	user1	facebook	55	23	...	31
2025-01-01 00:27:11	user1	youtube	53	442	...	112
2025-01-01 00:29:45	user1	facebook	66	245	...	11
2025-01-01 02:12:48	user2	twitter	33	212	...	12
2025-01-01 17:41:56	user3	facebook	622	23	...	42

For example if we have:

And are asked the query:

myscript --from-datetime="2025-01-01 00:00:00" --to-datetime="2025-01-01 09:00:00" --granularity=30m --dimensions user,app

the result should be:

timestamp,user,app,metric1,...,metric9
2025-01-01 00:00:00,user1,facebook,<sum of metric1>,...<sum of metric9>
2025-01-01 00:00:00,user1,youtube,<sum of metric1>,...<sum of metric9>
2025-01-01 00:02:00,user2,twitter,<sum of metric1>,...<sum of metric9>

Note:
-- the two facebook entries for user1 are both within the same 30m, so their metrics are aggegated (sum) into a single entry
-- the timestamp used for the aggregated entry for user1 are the start of that 30m range the values fall inside
-- the entry for user3 is discarded, since it falls after the --to-datetime

Similarly, if the query was:

myscript --from-datetime="2025-01-01 00:00:00" --to-datetime="2025-01-01 09:00:00" --granularity=30m --dimensions user

Since "app" is no longer a dimension we should keep, we should aggregate different apps within the same 30m into a single row, e.g. the output for user 1 would be like:

2025-01-01 00:00:00,user1,<sum of metric1>,...<sum of metric9>

Note how the app dimension was dropped from the output in this case.

Instructions:

Make the script fast and with good organization.

(a) Assume any log files your script will be used with have no header row, and are sorted by the user column.

(b) Use plain python and the standard library. Don't use libs like numpy or pandas.

(c) Assume the final log files the script will be tested with can have millions of users, but each user just has a few entries per 30m/day. Try to do the script as efficient as possible.

(d) Your script should be able to read all files (as needed) and answer such queries:

(1) myscript --from_datetime="2025-01-01 00:00:00" --to_datetime="2025-01-08 12:30:00" --user=user2 --granularity=30m --dimensions user,app

(2) myscript --from_datetime="2025-01-01 00:00:00" --to_datetime="2025-01-08 12:30:00" --user=user2 --granularity=30m --dimensions user

(3) myscript --from_datetime="2025-01-01 00:00:00" --to_datetime="2025-01-08 12:30:00" --user=user2 --granularity=30m --dimensions app

(4) myscript --from_datetime="2025-01-05 06:00:00" --to_datetime="2025-01-16 15:00:00" --user=user1,user2 --granularity=1day --dimensions user

(5) myscript --from_datetime="2025-01-05 06:00:00" --to_datetime="2025-01-16 15:00:00" --granularity=1day --dimensions user

(6) myscript --from_datetime="2025-01-05 06:00:00" --to_datetime="2025-01-16 15:00:00" --user=user1,user2 --app=facebook --granularity=1day --dimensions user

To test your script, you can generate sample logs files for multiple days, with users from user1 to user99, and a few app names like facebook, twitter, youtube, and such, with random timestamps belonging to the day. Not all users should have entries for all apps or all days. The sample logs should be sorted by the user column (like the actual logs will be).

SHELL EXERCIZES

Exercise 1

Provide a shell script that takes a path as argument and

- prints the size of each directory directly or nested under the path which (a) contains a .git directory (b) uses more than 1 GB
- Prints the time taken for the check per directory found

Exercise 2

Given 48 csv files with columns “user”,”application” and 48 csv files with columns “user”,”device” available per day, corresponding to 30 minute intervals, eg.

==> user.application.2020-01-16-00-00.csv <==
user1,app1
user1,app2

==> user.application.2020-01-16-00-30.csv <==
user1,app1
user2,app4
user4,app5

==> user.device.2020-01-16-00-00.csv <==
user1,device1
user1,device3

==> user.device.2020-01-16-00-30.csv <==
user9,decvice8
user1,devcice2

provide a shell script that takes a date YYYY-MM-DD as argument and

• writes 1 csv file with columns “application”,”number of unique users”

• writes 1 csv file with columns “device”,”number of unique users”

• writes 1 csv file with columns “application”,“device” with all possible unique application and device combinations found in those files.

Assume the input files are several GB each.

Exercise 3

Given daily csv files with columns “user”,”application” available for several consecutive days, eg.

==> user.application.2020-01-16.csv <==
user1,app1
user1,app2

==> user.application.2020-01-17.csv <==
user4,app5
user6,app3

==> user.application.2020-01-18.csv <==
user1,app1
user1,app2
user2,app2
user5,app2

==> user.application.2020-01-19.csv <==
user1,app1
user1,app2
user3,app3

==> user.application.2020-01-20.csv <==
user8,app9

provide a shell script that takes as argument a date and an application and prints the list of users who were using the provided application **every day** up to the provided date.

Assume the daily CSV files are several GB each.