Coding test

Problem description

Given is a TSV file (sales_data.tsv) with dummy sales data. Each row contains a **transaction** record that describes how many units of a product have been sold at a specific store at a specific time. See excerpt below:

```
+----------+--------+-------------+-----+-------------+-----+--------------------------+
|product_id|store_id|product_name |units|transaction_id|price|timestamp                 |
+----------+--------+-------------+-----+-------------+-----+--------------------------+
|0         |1       |coffee large |3    |1            |1.0  |2021-12-01 17:48:41.569057|
|0         |2       |coffee large |1    |3            |1.0  |2021-12-01 21:42:11.569057|
|3         |1       |doughnut cold|1    |4            |1.0  |2021-12-01 23:10:41.569057|
|0         |1       |coffee large |1    |5            |1.0  |2021-12-01 23:41:46.569057|
|4         |2       |snickers 37g |1    |7            |3.1  |2021-12-02 02:14:41.569057|
```

We want to compute the **sales profiles** for a **subset of stores** and write those sales profiles to a **single JSON file** in the following format (here for the stores with ids 1 and 3):

```
{
  "1":{
    "bread 700g":0.15,
    "coffee large":0.32,
    "doughnut cold":0.18,
    "milk 2L":0.18,
    "snickers 37g":0.17
  },
  "3":{
    "bread 700g":0.19,
    "coffee large":0.17,
    "doughnut cold":0.37,
    "milk 2L":0.15,
    "snickers 37g":0.13
  }
}
```

A **sales profile** is computed by summing up the units sold for each product in each store and dividing by the total sum of units sold per store. These normalised unit sales must sum up to one (per store). This allows the comparison of stores independent of their total sales. For instance, the example above reveals that store 1 sells relatively more coffee than store 3, while store 3 sells more doughnuts.

Notes

- Assume that the data file (sales_data.tsv) is **too large** to be loaded into memory at once.
- However, the sales profiles for the selected stores can be assumed to fit into memory.
- There are invalid transactions with negative units that need to be filtered out
- There are different “spellings” for the same product that need to be unified, e.g. “coffee-large”, “coffee_large” and “coffee large”.

Tasks

1) Write a **PySpark pipeline** that takes a set of store ids as input, loads the sales_data.tsv file, and writes the sales profiles in JSON format to a file. The code must reproduce the sales profiles shown above for store ids 1 and 3.

Recommendation: use **Pandas** to compute the sales profiles and write the JSON output, once the data has been aggregated via PySpark.

2) Write **unit tests** using **pytest** to test the functions of the pipeline.
(https://docs.pytest.org/en/7.3.x/)

3) Run **pytest-cov** to ensure 100% test coverage.
(https://pypi.org/project/pytest-cov/)

Deliverables

Link to GitHub repo (or zipped code via email) with unit tests and coverage result.

Questions

stefan.maetschke@getqsic.com