Mind AI
https://about.mind.ai/careers

Cinema Screening Validation Challenge

Overview

It's the year 2040. There is a cinema located in Myeongdong, Seoul, which has been operating for 15 years and is sold out every day. It opens 5 days a week and has two screening rooms. The first one shows 3 movies a day, and the second one shows 2 screenings a day. It only shows movies from the year 2024 and earlier, yet the cinema is sold out every day. Not much is known about how the cinema decides which movie to screen on which day in which screening room, only that the screening schedule is unique every week. A team of data scientists is working tirelessly to uncover the rules that contribute to this successful screening plan. Your task, however, is to validate how well the proposed rules align with the historical data.

What the data scientists will deliver

The data scientists work on two sources of data:

1. The pool of movies including their characteristics are documented in the file `movies.csv`.
2. The weekly screening schedules from the past 15 years are also known and documented in the file `past_screenings.csv`.

They have been asked to determine the set of rules the cinema has followed in creating the screening schedule over the last 15 years. The rules must adhere to this pattern:

Movies `<movie_filter?>` `<characteristics?>` `<limit_function>` `<limit>` time(s) every `<frequency>` day(s).

movie_filter

A `<movie_filter>` is optional in the rule and can be:

1. A **filter expression**: An atomic condition such as `genre=Drama`, `director=Ingmar Bergman`, or `year=1968`.
2. A **boolean expression**: An expression that combines one or more filter expressions using boolean operators (`and`, `or`, `not`). Boolean expressions can be nested and combined to form more complex conditions.

All column names can be used as keys for the filter expression, while the values of the filter expressions should be one of the values of the column.

Examples for movie_filter

1. Movies of the same name must not be shown more than 1 time every 5 days.

2. Movies `where (genre=Drama) and (director=Ingmar Bergman)` of the same name must not be shown more than 1 time every 5 days.

3. Movies `where not(genre=Drama)`of the same name must not be shown more than 1 time every 5 days.

4. Movies `where (((genre=Drama) and (director=Ingmar Bergman)) or (year=1968))` of the same name must not be shown more than 1 time every 5 days.

5. Movies `where not(((genre=Drama) or (director=Ingmar Bergman)))` of the same name must not be shown more than 1 time every 5 days.

characteristics

Rules can optionally be applied to clusters of movies with shared characteristics. These shared characteristics are defined by the column names in the movie dataset. If a cluster is defined by a single characteristic, it is written as `of the same characteristic_a.`If a cluster is defined by multiple characteristics, they are written as `of the same characteristic_a and characteristic_b`.

Examples for characteristics

1. Movies must not be shown more than 1 time every 5 days.
2. Movies `of the same genre` must not be shown more than 1 time every 5 days.

3. Movies `of the same genre and director` must not be shown more than 1 time every 5 days.

4. Movies `of the same genre and director and year` must not be shown more than 1 time every 5 days.

limit_function

The limit function is either `must not be shown more than` or `must be shown at least`.

limit and frequency

Both limit and frequency are integer values.

What you will be given

You too will be given the data sources `movies.csv` and `past_screenings.csv`. Additionally, the validator we expect you to write must have an API to also receive a file path to a JSON file. This JSON file is the result of an automatic process that converts the set of rules in natural language discovered by the data scientists. This file has a JSON array as its root data structure, with each element representing a rule in the form of another JSON array. The JSON representation of that pattern follows this schema:

```
["count_limit_template",
    {"movie_filter":
    null |
    ["and" | "or" , <movie_filter>+ ] |
    ["not", <movie_filter>] |
    ["=", <characteristic:str>, <characteristic_value>:str|int],
    "characteristics": null | ["characteristics=", <characteristic:str>+],
    "limit_function": ["<at_most|at_least>", <limit:int>],
    "frequency": ["every_n_days", <n:int>]}]
```

Examples of JSON rules

Natural Language: ***Movies of the same title must not be shown more than 1 time every 5 days.***

JSON:

```
["count_limit_template",
    {"movie_filter": null,
    "characteristics": ["characteristics=", "title"],
    "limit_function": ["at_most", 1],
    "time_grouping": ["every_n_days", 5]}]
```

Natural Language: ***Movies where ((country="South Korea") or (country="Germany")) must be shown more than 2 times every 3 days.***

JSON:

```
["count_limit_template",
    {"movie_filter": ["or", ["=", "country", "South Korea"], ["=", "country", "Germany"]],
    "characteristics": null,
    "limit_function": ["at_least", 2],
    "time_grouping": ["every_n_days", 3]}]
```

These rules are also included in the provided sample JSON file named `sample_rules.json`.

Expected solution

The expected results are:

Executable Python code

Write a Python program that can be executed from the command line. The program should take three arguments:

1. A filesystem path to a CSV file containing the movie dataset
2. A filesystem path to a CSV file containing the historical screening data
3. A filesystem path to a JSON file that represents the set of rules of the data scientists

The program should validate the historical data based on the specified rules and output only the number of weeks of historical data that would have been valid according to those rules. No additional text or formatting should be included in the output; just the number itself.

If no week would be valid according to the rules, output: 0

Tests and Documentation

If you have written any unit tests, please include them with your submission. Additionally, provide documentation explaining your design choices and the reasoning behind them.

Evaluation criteria

Your solution will be assessed based on the following criteria:

- The correctness of your implementation
- The runtime performance of your implementation
- The quality of your tests and documentation