quantco

Remote Engineering Challenge

One tool we use extensively is Pandas, a Python DataFrame library - and while it has a lot of features, it also comes with certain downsides, namely that it has a very loose uncontrollable way of acting with types and it doesn't really support null/None values in a good way (where an explicit None would e.g. have a different meaning than a NaN value, which denotes an invalid number, versus explicitly saying "we do not have a value here").

Overview

As a small Python programming exercise we would like you to implement a prototype of an alternative library that solves these two problems. We think this could be a good way of evaluating your programming skills and also give a bit of intuition what has to happen behind the scenes, for using Pandas and maybe in the future even fully implementing our own alternative. You can work like you normally would, meaning using your favorite editor and any kind of online documentation you need, but you should not copy extensive amounts of code and in this specific case since it is an evaluation of your skill level also generally do it yourself without asking others. There is no fixed time limit and you can take as long as you like, but we would recommend for your own sake that you don't spend significantly more than 3 hours on this task.

There is no problem if you have not heard of Pandas, or DataFrames in general before, as the following set of instructions will tell you exactly what to do. In fact we would like you to not use Pandas or NumPy and similar libraries internally to avoid staying within their way of thinking and limitations. Since this is going to be a prototype with focus on architecture you do not have to focus on performance - in fact we encourage you to use python lists and normal for loops for all internal data processing operations. Nevertheless we would like to hear a sentence or two about why this comes with performance issues and how you would do it instead for the real thing. And maybe add a comment/todo to all methods you would implement differently in this way.

Tasks

In general you can think of a DataFrame as a table with named columns, each of these columns is called Series, and in our case explicitly only holds values of a certain type (and the None value). A DataFrame only holds one Series for each string name, but there generally can be any number of Series in a DataFrame, with the limitation, that each Series has to have the same number of elements.

There are different types of Series, in this case a Series containing only string elements (and None), a boolean Series (with values True, False and None) and different numeric Series, namely one for floating point values and one for integer. In contrast to Pandas

we would rather fail hard and fast than having weird results, meaning you are supposed to raise exceptions for any kind of operations between Series of different types (unless explicitly mentioned otherwise) and if the lengths do not match. Generally a Series is constructed by handing it a list of the elements it is supposed to contain (and it should obviously perform a type check if all values conform to the created Series type or are None). To limit the scope we only ask you to implement read access, by overriding the square bracket access operator, which should when given an integer return the individual value at that position, and when given a boolean Series, a new Series with all rows/values where the boolean Series is True.

For all Series types you should implement the equality operation, which should fail hard if the Series types or lengths are different and otherwise perform an element-wise comparison returning a boolean Series. Additionally, the boolean Series should support element-wise and, or, xor and invert operations, and both numeric Series types should support element-wise math operations $(+, -, *, /)$, and inequalities $(>, <, >=, <=, !=)$, returning a boolean Series. All of these operations should return new Series instead of modifying the existing one. As a form of syntactic sugar, all these operations should also work when being handed an individual value on the right side, where it would be expanded automatically to a Series of its respective type and the same length as the other Series.

Finally, you should build the DataFrame itself, which gets a dictionary of Series in its constructor, where it has to perform the necessary checks, and additionally provides square bracket access either when given a string returning the respective Series, or when given a boolean Series returns another DataFrame containing only the rows with True values. It could also be helpful to add print operations/a string representation to both DataFrame and Series.

Experiments

To test your implementation you should be able to build the following table as a DataFrame:

SKU String	price Float	sales Integer	taxed Boolean
”X4E”	7.0	5	False
”T3B”	3.5	3	False
”F8D”	8.0	1	True
”C7X”	6.0	10	False

and run the following code on it:

```
df = ... # construct your DataFrame here

# let's find all our tax free products/SKUs where the price
    + our $5.0 shipping fee is more than $10 and we had
   more than 3 sales
result = df[(df["price"] + 5.0 > 10.0) & (df["sales"] > 3)
   & ~df["taxed"]]["SKU"]
print(result)
```

Feel free to try more examples and maybe bigger tables.