Python technical exercise #3 - Aggregations

Introduction

Purpose of this exercise is to test candidate's problem-solving and software development skills on a real-world example. Exercise is split into 3 parts, which are meant to build on top of each other, producing a single feature.

Please note: goal of the exercise is to provide a *robust*, *tested* and *easily extensible* implementation of the described feature. In case the allotted time is not enough, please priorize quality over quantity (e.g. two high-quality tasks are better than three low-quality ones).

Guidelines

1. *All function signatures mentioned in the exercise have to be adhered to. Other functions can be introduced, depending on the need.*
2. *Each function mentioned below should be properly documented (docstring) and have a test case that covers it.*
3. *In case of invalid input, lack of data, or any other errors, inform the user by raising an exception.*

Implementation language: Python 3.7+
Duration: 4 hours
Output: ZIP archive with necessary `.py` files, together with a `README` file with instructions on how to execute the code.

Background

After the input data has been validated and normalized, derived data can be generated by executing formula-based aggregations.

Part 1 - Retrieving transaction data

Background

Transaction data can be retrieved from the REST API of the European Central Bank, using the URL below:

`https://sdw-wsrest.ecb.europa.eu/service/data/BP6/Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N?detail=dataonly`

In the URL above, `Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N` is a transaction identifier and can be replaced with any other (valid) identifier.

Response, in XML format, contains multiple children of the following format:

```
<generic:Obs>
    <generic:ObsDimension value="1999-Q1"/>
    <generic:ObsValue value="55420.1818623299"/>
</generic:Obs>
```

Each of these represents the transaction amount at a certain point in time.

Task

Implement the function with the following signature:

```
def get_transactions(identifier: str) -> pd.DataFrame
```

Function should fetch the transaction data from the appropriate URL and convert it to a pandas DataFrame, with columns `IDENTIFIER`, `TIME_PERIOD` and `OBS_VALUE`, corresponding to values of the identifier parameter, `generic:ObsDimension` tag and `generic:ObsValue` tag from the XML. `OBS_VALUE` should be converted to `float`.

Example

Running: `get_transactions("Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N")` should produce the following:

IDENTIFIER	TIME_PERIOD	OBS_VALUE
Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N	1999-Q1	5420.181862
Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N	1999-Q2	87003.970961

...

Part 2 - Retrieving data for the formula

Background

An aggregation formula is given as an *assignment* expression involving transaction identifiers. For example:

```
Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N =
    Q.N.I8.W1.S1P.S1.T.A.FA.D.F._Z.EUR._T._X.N +
    Q.N.I8.W1.S1Q.S1.T.A.FA.D.F._Z.EUR._T._X.N
```

is a valid formula. For simplicity, expression to the right of `=` will only include addition (`+`) and substraction (`-`) operators. While there is a guarantee that the left-hand side of the `=` operator will contain only one identifier, number of identifiers on the right-hand side is not fixed.

Task

Implement the function with the following signature:

```
def get_formula_data(formula: str) -> DataFrame
```

The function should parse the formula and fetch the data (using the function from part #1) for the identifiers on the right-hand side of the `=` operator.

Resulting DataFrame should use `TIME_PERIOD` column as the index, and have as many columns as there are identifiers, each containing values of the `OBS_VALUE` column from the appropriate DataFrame. Names of the columns are the names of the identifiers, in the order they appear in the formula.

In case some identifier have data for a `TIME_PERIOD`, but others don't, fill the blanks with 0.

Example

Running:

```
get_formula_data("Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N =
Q.N.I8.W1.S1P.S1.T.A.FA.D.F._Z.EUR._T._X.N +
Q.N.I8.W1.S1Q.S1.T.A.FA.D.F._Z.EUR._T._X.N")
```

should produce the following:

TIME_PERIOD	Q.N.I8.W1.S1P.S1.T.A.FA.D.F._Z.EUR._T._X.N	Q.N.I8.W1.S1Q.S1.T.A.FA.D.F._Z.EUR._T._X.N
2008-Q1	158849.702543	158813.745366
2008-Q2	55317.060368	55136.813191

...

Part 3 - Computing the aggregates

Task

Implement the function with the following signature:

```
def compute_aggregates(formula: str) -> pd.DataFrame
```

The function should load the data using the function from part #2 and return a DataFrame, indexed by `TIME_PERIOD`, with a single column, whose name is the identifier on the left-hand side of the `=` operator. Values of this column are obtained by applying the appropriate operations, as indicated by the formula, on the retrieved data.

Example

Running:

```
compute_aggregates("Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N =
Q.N.I8.W1.S1P.S1.T.A.FA.D.F._Z.EUR._T._X.N +
Q.N.I8.W1.S1Q.S1.T.A.FA.D.F._Z.EUR._T._X.N")
```

should produce the following:

TIME_PERIOD	Q.N.I8.W1.S1.S1.T.A.FA.D.F._Z.EUR._T._X.N
2008-Q1	317663.447909
2008-Q2	110453.873559

...

Note: Values above are obtained by summing up the values (as indicated by the formula) of the DataFrame in part #2.