Transformation Writing Rules

The Predicate transformation is a Python class with a defined structure. This class does not contain specific dependencies other than `pandas`.

This page describes the structure of the transformation class. In the future, this page may be supplemented with recommendations for writing transformations.

Structure of the Transformation Class

The transformation class has a set of mandatory fields and methods. There are no restrictions on the use of additional fields or methods. We even recommend using additional methods and fields in the "Recommendations for Writing Transformations" section.

Mandatory fields:

1. `__desc__`;
2. `__tags__`;
3. `__add_column__`;
4. `__del_column__`.

Mandatory methods:

1. `__init__`;
2. `__call__`.

The sections below describe the fields and methods in more detail.

Fields

The class name becomes the name of the transformation in the system.

`__desc__` - a human-readable description of the transformation.

`__tags__` - a list of tags assigned to the transformation in the system.

`__add_column__` - a list of column names that the transformation adds to the original dataset.

`__del_column__` - a list of column names that the transformation removes from the original dataset.

Thus, the fields for a transformation that adds information about the duration of audio files to the dataset and does not remove anything from the dataset look as follows:

```
class GetAudioDuration:
    __desc__ = "Saving the duration of audio files by names from the dataset"
    __tags__ = ["audio"]

    __add_column__ = ["duration_seconds"]
    __del_column__ = []
```

Methods

__init__

The class initialization method is intended for inputting transformation parameters. All transformation parameters will be passed to this method when calling the transformation in the Predicate project or within the library.

Basic signature:

```
def __init__(self, *, **kwargs: typing.Any) -> None:
    ...
```

The definition of the `__init__` method in the transformation class must comply with the following rules:

1. At least one dataset must be passed to the method.
2. Each parameter must have a type annotation according to the parameterization rules, similar to the parameterization rules for metrics.
3. The default value of the parameter must not violate the logic of validating the specified parameter type.
4. Parameter values must be passed to subsequent methods through the instance attributes of the class `self`.

It is recommended to perform all checks of the entered values within this method.

Example for audio transformation:

```
def __init__(self, df: pd.DataFrame, path_colomn:str = 'path'):
        self.df = df
        self.path_colomn = path_colomn

        if self.df.empty:
```

```
            raise Exception("Dataframe is empty")
        if self.path_colomn not in self.df:
            raise ValueError(f"Field {self.path_colomn} does not exist in the
dataframe")
```

Thus, the transformation takes as input:

- A `pandas` dataframe with data for calculation;
- The name of the column containing the path information to the audio file.

In the body of the method, the transformation instance (`self`) is assigned parameter values for use in other methods, and checks for the correctness of the entered data are performed.

> ***__call__***

The class call method contains the main calculations of the transformation. There are no restrictions on the content; the method can contain any calculations.

Basic signature:

```
def __call__(self) -> pd.DataFrame:
    ...
```

The method has no parameters other than `self` and must return a `pandas.DataFrame` .

Example for audio transformation:

```
def __call__(self) -> pd.DataFrame:
        durations = [self.get_audio_duration(f'/tmp/data/audio/{path}') for
path in self.df[self.path_colomn]]
        self.df['duration_seconds'] = durations

        return self.df
```

Here, the original dataframe was updated, adding a new column `duration_seconds` .

Order of Method Execution

Methods in both the Predicate product and the Predicate library are executed in the following order:

1. `__init__` ;
2. `__call__` .

Consider the order of method execution when developing the transformation. It affects how values can be passed from method to method.