Правила написания преобразований

Преобразование Predicate - это класс Python с определенной структурой. Данный класс не содержит специфичных зависимостей, кроме `pandas`.

На данной странице описана структура класса с преобразованием. В дальнейшем возможно дополнение страницы рекомендациями по написанию преобразований.

Структура класса с преобразованием

У класса преобразования есть набор обязательных полей и методов. При этом нет ограничений на использование дополнительных полей или методов. Мы даже посоветуем использовать дополнительные методы и поля в разделе "Рекомендации по написанию преобразований".

Обязательные поля:

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

Обязательные методы:

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

В разделах ниже поля и методы описаны подробнее.

Поля

Название класса становится названием преобразования в системе

`__desc__` - понятное человеку описание преобразования.

`__tags__` - список тегов, присвоенных преобразованию в системе.

`__add_column__` - список названий столбцов, которые преобразование добавляет к исходному датасету.

`__del_column__` - список названий столбцов, которые преобразование удаляет из исходного датасета.

Таким образом, поля для преобразования, которое добавляет в датасет информацию о длительности аудиофайлов и ничего не удаляет из датасета, выглядят следующим образом:

```
class GetAudioDuration:
    __desc__ = "Сохранение длительности аудио-файлов по названиям из датасета"
    __tags__ = ["audio"]

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

Методы

__init__

Метод инициализации класса предназначен для ввода параметров преобразования. Именно в этот метод будут передаваться все параметры преобразования при вызове преобразования в проекте Predicate или в рамках библиотеки.

Базовая сигнатура:

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

Определение метода `__init__` в классе преобразования должно соответствовать следующим правилам:

1. В метод должен быть передан хотя бы один датасет.
2. Каждый параметр должен иметь аннотацию его типа согласно правилам параметризации, аналогичным правилам параметризации метрик.
3. Значение параметра по умолчанию не должно нарушать логику валидации указанного типа параметра.
4. Передавать значения параметров в следующие методы нужно через атрибуты экземпляра класса `self`.

Рекомендуем проводить все проверки введённых значений в рамках данного метода

Пример для аудио-преобразования:

```
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")
```

Таким образом, преобразование принимает на вход:

- `pandas` датафрэйм с данными для расчёта;
- имя колонки с информацией о пути до аудиофайла.

В теле метода экземпляру преобразования (`self`) присваиваются значения параметров для использования в остальных методах, и проводятся проверки правильности введенных данных.

> ***__call__***

Метод вызова класса содержит основные расчёты преобразования. Ограничений на содержимое нет, метод может содержать любые расчёты.

Базовая сигнатура:

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

Метод не имеет параметров, кроме `self`, и должен возвращать `pandas.DataFrame`.

Пример для аудио-преобразования:

```
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
```

Здесь для преобразования был обновлен исходный датафрейм, в который добавили новую колонку `duration_seconds`.

Порядок исполнения методов

Методы и в продукте Predicate, и в библиотеке Predicate исполняются в следующем порядке:

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

Учитывайте порядок исполнения методов при разработке преобразования. От него зависит, как можно передавать значения из метода в метод.