**Продвинутый Pandas. Часть 2**

Темы на сегодня:

* Объединение датафреймов: что не так?
* Типы объединений
* Борьба с дубликатами

ОБЪЕДИНЕНИЕ ДАТАФРЕЙМОВ: ЧТО НЕ ТАК?

Проанализировать таблицу на предмет определённых данных возможно различными способами, не только при помощи Pandas. Сложности возникают в тех случаях, когда стоит задача сделать выводы, исходя из данных, содержащихся в разных таблицах.

**Пример:**

*visits = pd.DataFrame(*

*{*

*'user\_id': [11, 22, 55, 11, 77],*

*'source': ['ad', 'yandex', 'email', 'google', 'ad']*

*}*

*)*

*visits = visits[['user\_id', 'source']]*

*print(visits)*

*user\_id source*

*0 11 ad*

*1 22 yandex*

*2 55 email*

*3 11 google*

*4 77 ad*

*purchases = pd.DataFrame(*

*{*

*'user\_id': [11, 22, 55, 11, 99],*

*'category': ['Спорт', 'Авто', 'Дача', 'Спорт', 'Авто'],*

*}*

*)*

*purchases = purchases[['user\_id', 'category']]*

*print(purchases)*

*user\_id source*

*0 11 ad*

*1 22 yandex*

*2 55 email*

*3 11 google*

*4 77 ad*

Если предстоит склеить подобные таблицы, то возникает следующая проблема: они редко бывают готовы к объединению по причине неполного соответствия друг другу. Или неоднозначного. Какие-то проблемы можно решить с помощью группировок, сводных таблиц, фильтров, но наслоение методов предварительной обработки усложняет анализ.

Именно для этого были созданы 2 метода, которые решают проблему объединения датафреймов без лишних действий:

* **.merge()** – склеивание по столбцам. С ним уже немного знакомились ранее
* **.join()** – склеивание по индексу

То есть, .join() подходит для таблиц, где в индексе находится какой-либо столбец. Обычно такая ситуация происходит в результате группировки.

**Пример:**

*visits\_grouped = visits.groupby('user\_id').count()*

*visits\_grouped.rename(columns={'source': 'visits'}, inplace=True)*

*print(visits\_grouped)*

*visits*

*user\_id*

*11 2*

*22 1*

*55 1*

*77 1*

В противном случае используется .merge()

**Пример:**

*visits\_grouped.reset\_index()*

*user\_id visits*

*0 11 1*

*1 22 1*

*2 55 1*

*3 77 1*

Соответственно, желательно привести таблицы к одному виду.

**Пример:**

*purchases\_pivot = purchases.pivot\_table(index='user\_id', columns='category', values='user\_id', aggfunc='size', fill\_value=0)*

*print(purchases\_pivot)*

*category Авто Дача Спорт*

*user\_id*

*11 0 0 2*

*22 1 0 0*

*55 0 1 0*

*99 1 0 0*

*# Через .join()*

*visits\_grouped.join(purchases\_pivot)*

*visits Авто Дача Спорт*

*user\_id*

*11 2 0.0 0.0 2.0*

*22 1 1.0 0.0 0.0*

*55 1 0.0 1.0 0.0*

*77 1 NaN NaN NaN*

# Через .merge()

*visits\_grouped.reset\_index().merge(purchases\_pivot.reset\_index())*

*user\_id visits Авто Дача Спорт*

*0 11 2 0 0 2*

*1 22 1 1 0 0*

*2 55 1 0 1 0*

Может показаться, что .join() однозначно лучше, чем .merge(), но всё определяется решаемой задачей и тем, в каком виде находятся датафреймы.

Стоит упомянуть, что считается плохим тоном хранить объединённые таблицы, поскольку они содержат различные данные в большом количестве, а потому их обработка может идти долго. Разумнее разделить на несколько таблиц, в части которых будут исключительно числовые данные, а в других – строковые в меньшем количестве.

ТИПЫ ОБЪЕДИНЕНИЙ

Из последнего примера можно видеть, что по умолчанию .merge() и .join() сработали по-разному в случае отсутствия соответствия. Первый удалил те строки, которым не смог найти соответствие, второй же эти строки заполнил None-значениями. Дело в том, что поведение методов определяется их дополнительными параметрами, важнейшим из которых является тип объединения.

Типы объединения:

* left join: за основу берётся первая таблица, каждой её строке ищется соответствие во второй таблице. В случае, если его нет, в соответствие таким строкам ставится None-значение. Используется чаще всего.

**Пример:**

*visits\_grouped.join(purchases\_pivot, how='left')*

*visits Авто Дача Спорт*

*user\_id*

*11 2 0.0 0.0 2.0*

*22 1 1.0 0.0 0.0*

*55 1 0.0 1.0 0.0*

*77 1 NaN NaN NaN*

* right join: за основу берётся вторая таблица, каждой её строке ищется соответствие в первой таблице. В случае, если его нет, в соответствие таким строкам ставится None-значение. Используется редко, по особой необходимости.

**Пример:**

*visits\_grouped.join(purchases\_pivot, how='right')*

*visits Авто Дача Спорт*

*user\_id*

*11 2.0 0 0 2*

*22 1.0 1 0 0*

*55 1.0 0 1 0*

*99 NaN 1 0 0*

* inner join: в результирующий датафрейм помещаются лишь те значения обеих таблиц, которым есть соответствие в другой таблице. Используется по необходимости, поскольку часть необходимых значений может быть потеряна.

**Пример:**

*visits\_grouped.join(purchases\_pivot, how='inner')*

*visits Авто Дача Спорт*

*user\_id*

*11 2.0 0 0 2*

*22 1.0 1 0 0*

*55 1.0 0 1 0*

* outer join: в результирующий датафрейм помещаются значения обеих таблиц. Тем из них, которым нет соответствия в другой таблице, ставится в соответствие None-значение. Используется по необходимости, поскольку содержит в себе все данные и чаще всего избыточна.

**Пример:**

*visits\_grouped.join(purchases\_pivot, how='outer')*

*visits Авто Дача Спорт*

*user\_id*

*11 2.0 0.0 0.0 2.0*

*22 1.0 1.0 0.0 0.0*

*55 1.0 0.0 1.0 0.0*

*77 1.0 NaN NaN NaN*

*99 NaN 1.0 0.0 0.*

БОРЬБА С ДУБЛИКАТАМИ

При объединении таблиц нужно также держать в уме тот факт, что в таблицах могут быть дубликаты. В таком случае, особенно при outer join, появляется избыточная информация, которая может повлиять на дальнейший анализ. Например, если потребуется сделать подсчёт значений.

**Пример:**

*ratings = pd.read\_csv('ratings\_example.txt', sep = '\t')*

*ratings.head()*

*userId movieId rating timestamp*

*0 1 31 2.5 1260759144*

*movies = pd.read\_csv('movies\_example.txt', sep = '\t')*

*movies.head()*

*movieId title genres*

*0 31 Dangerous Minds (1995) Drama*

*1 32 Twelve Monkeys (a.k.a. 12 Monkeys) (1995) Mystery|Sci-Fi|Thriller*

*2 31 Dangerous Minds (1995) Drama*

*# При объединении есть дубликаты*

*ratings.merge(movies, how='left', on='movieId')*

*userId movieId rating timestamp title genres*

*0 1 31 2.5 1260759144 Dangerous Minds (1995) Drama*

*1 1 31 2.5 1260759144 Dangerous Minds (1995) Drama*

Для избавления от дубликатов следует использовать метод **.drop\_duplicates()**. У метода есть параметры:

* subset – столбец, по которому нужно искать дубли. По умолчанию – все столбцы
* keep – как поступать с дубликатами. First (по умолчанию) – удалить все экземпляры, кроме первого. Last – удалить все экземпляры, кроме последнего. False – удалить все экземпляры без исключений

**Пример:**

*movies.drop\_duplicates(subset = ['movieId'], keep = 'first', inplace = True)*

*movies.head()*

*movieId title genres*

*0 31 Dangerous Minds (1995) Drama*

*1 32 Twelve Monkeys (a.k.a. 12 Monkeys) (1995) Mystery|Sci-Fi|Thriller*

*ratings.merge(movies, how = 'left', on = 'movieId')*

*userId movieId rating timestamp title genres*

*0 1 31 2.5 1260759144 Dangerous Minds (1995) Drama*

*ratings.merge(movies, how = 'right', on = 'movieId')*

*ratings.merge(movies, how = 'right', on = 'movieId')*

*userId movieId rating timestamp title genres*

*0 1.0 31 2.5 1.260759e+09 Dangerous Minds (1995) Drama*

*1 NaN 32 NaN NaN Twelve Monkeys Mystery|Sci-Fi|*

*(a.k.a. 12 Monkeys) (1995) Thriller*

По окончании процесса объединения и устранения дубликатов полезно проверять правильность с помощью функции длины **len**. Алгоритм для left join:

* Ищем длину первого датафрейма
* Объединяем датафреймы
* Ищем длину объединённого датафрейма
* Проверяем: длина полученного датафрейма должна быть равна длине первого датафрейма

Для right join алгоритм меняется соответственно.