**Продвинутый Pandas. Часть 1**

Темы на сегодня:

* Сводные таблицы
* Ещё немного о фильтрации
* Даты и строки в Pandas

СВОДНЫЕ ТАБЛИЦЫ

Довольно часто в задачах по анализу требуется считать количество строк, соответствующих определённому параметру. Уже известно, что для отдельного параметра такая задача решается 2 способами:

* датафрейм.groupby(‘столбец’).count()
* датафрейм[‘столбец’].value\_counts()

При этом существует вариант построения целой таблицы подобных подсчётов по всем параметрам: так называемой сводной таблицы. В Pandas за это отвечает метод **.pivot\_table()**. Его параметры:

* index – столбец, который будет использован для строк
* columns – столбец, который будет использован для столбцов
* values – столбец обрабатываемых значений
* aggfunc – функция, применяемая к values
* fill\_value – значение по умолчанию
* margins – если True, то добавляется столбец All (Итого). По умолчанию: False

**Пример:**

*import pandas as pd*

*ratings = pd.read\_csv('ratings.csv')*

*ratings.head()*

*userId movieId rating timestamp*

*0 1 1 4.0 964982703*

*1 1 3 4.0 964981247*

*2 1 6 4.0 964982224*

*3 1 47 5.0 964983815*

*4 1 50 5.0 964982931*

*ratings.pivot\_table(index = 'userId', columns = 'rating', values = 'timestamp', aggfunc = 'count', fill\_value = 0, margins = True).head()*

*rating 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 All*

*userId*

*1 0 1 0 5 0 26 0 76 0 124 232*

*2 0 0 0 1 1 4 4 9 4 6 29*

*3 20 0 0 1 0 1 1 1 5 10 39*

*4 0 23 0 26 0 39 0 64 0 64 216*

*5 0 1 0 3 0 17 0 13 0 10 44*

ЕЩЁ НЕМНОГО О ФИЛЬТРАЦИИ

До этого фильтрация рассматривалась как способ фактического отсеивания ненужных элементов датафрейма. Но часто задача сводится к более простой: отображению лишь необходимых элементов без различного рода преобразований. Это можно решить с помощью уже известного метода **.loc()**, который позволял отобразить необходимый диапазон строк. Однако, с его помощью возможно управлять и отображением столбцов.

**Пример:**

*log = pd.read\_csv('visit\_log.csv', sep=';')*

*log.head()*

*timestamp visit\_id url region user\_id traffic\_source*

*0 1549980692 e3b0c44298 https://host.ru/3c19 Russia b1613cc09f yandex*

*1 1549980704 6e340b9cff https://host.ru/c8d9 Russia 4c3ec14bee direct*

*2 1549980715 96a296d224 https://host.ru/b8b5 Russia a8c40697fb yandex*

*3 1549980725 709e80c884 https://host.ru/b8b5 Russia 521ac1d6a0 yandex*

*4 1549980736 df3f619804 https://host.ru/b8b5 Russia d7323c571c yandex*

*log.loc[:, ['user\_id', 'region']].head()*

*# Двоеточие без указания строк говорит о том, что нужно использовать все строки*

*user\_id region*

*0 b1613cc09f Russia*

*1 4c3ec14bee Russia*

*2 a8c40697fb Russia*

*3 521ac1d6a0 Russia*

*4 d7323c571c Russia*

Кроме того существует похожий метод **.iloc()**, который в отличие от предыдущего принимает номера столбцов.

**Пример:**

*log.iloc[:, [1, -3]].head()*

*visit\_id region*

*0 e3b0c44298 Russia*

*1 6e340b9cff Russia*

*2 96a296d224 Russia*

*3 709e80c884 Russia*

*4 df3f619804 Russia*

Можно также прописать условие фильтрации.

**Пример:**

*log.loc[log.region == 'China'].head()*

*timestamp visit\_id url region user\_id traffic\_source*

*8 1549980760 af5570f5a1 https://host.ru/3004 China 45664f7af2 direct*

*13 1549980792 dd46c3eebb https://host.ru/3004 China 7e301c8c41 direct*

*24 1549980842 9d908ecfb6 https://host.ru/ae8a China 583b1a1636 direct*

*68 1549981106 1751ac12e7 https://host.ru/c238 China 81c4917b0d direct*

*73 1549981142 9c56f48ae9 https://host.ru/108c China 4b07d164e7 direct*

При этом можно создавать новые столбцы.

**Пример:**

*log.loc[log.region == 'Russia', 'VAT'] = 1.2*

*log.head(10)*

*timestamp visit\_id url region user\_id traffic\_source VAT*

*0 1549980692 e3b0c44298 https://host.ru/3c19 Russia b1613cc09f yandex 1.2*

*1 1549980704 6e340b9cff https://host.ru/c8d9 Russia 4c3ec14bee direct 1.2*

*2 1549980715 96a296d224 https://host.ru/b8b5 Russia a8c40697fb yandex 1.2*

*3 1549980725 709e80c884 https://host.ru/b8b5 Russia 521ac1d6a0 yandex 1.2*

*4 1549980736 df3f619804 https://host.ru/b8b5 Russia d7323c571c yandex 1.2*

*5 1549980742 8855508aad https://host.ru/df64 Russia fc43898e47 yandex 1.2*

*6 1549980742 b0f66adc83 https://host.ru/b8b5 Russia 13fc55e781 paid 1.2*

*7 1549980754 837885c8f8 https://host.ru/108c Russia cb5082b6f6 direct 1.2*

*8 1549980760 af5570f5a1 https://host.ru/3004 China 45664f7af2 direct NaN*

*9 1549980765 3e7077fd2f https://host.ru/df64 Russia 6f9de8c8b6 email 1.2*

А также задавать функции в качестве фильтров.

**Пример:**

*log.loc[lambda row: row.region == 'China'].head(10)*

*timestamp visit\_id url region user\_id traffic\_source VAT*

*8 1549980760 af5570f5a1 https://host.ru/3004 China 45664f7af2 direct NaN*

*13 1549980792 dd46c3eebb https://host.ru/3004 China 7e301c8c41 direct NaN*

*24 1549980842 9d908ecfb6 https://host.ru/ae8a China 583b1a1636 direct NaN*

*68 1549981106 1751ac12e7 https://host.ru/c2382 China 81c4917b0d direct NaN*

*73 1549981142 9c56f48ae9 https://host.ru/108c China 4b07d164e7 direct NaN*

*78 1549981170 5552748b5a https://host.ru/186c China da8a1c12a1 email NaN*

*90 1549981255 988a09c9a6 https://host.ru/8e88 China 3aee8c305f email NaN*

*117 1549981446 fee3d3a171 https://host.ru/df64 China 934c45c561 direct NaN*

*121 1549981476 d8129de428 https://host.ru/05e4 China 4a5d3c1068 email NaN*

*141 1549981610 dc2021c180 https://host.ru/186c China bdee21b60c yandex NaN*

Пустые значения можно фильтровать с помощью аналогичных методов **.isnull()** и **.isna()**

**Пример:**

*import numpy as np*

*df = pd.DataFrame({'value': [123, None, np.nan, np.NaN, np.NAN, 456]})*

*print(df)*

*value*

*0 123.0*

*1 NaN*

*2 NaN*

*3 NaN*

*4 NaN*

*5 456.0*

*df.loc[pd.isnull(df.value) == False, :]*

*value*

*0 123.0*

*5 456.0*

ДАТЫ И СТРОКИ В PANDAS

До этого момента время фигурировало в формате UNIX, что очень удобно по причине того, что это всегда целое число. Тем не менее такой формат неудобен для понимания. Поэтому его можно преобразовать к стандартному виду YYYY-MM-DD-… при помощи метода **.to\_datetime(число, unit=точность\_преобразования)**. При этом тип данных также меняется на datetime.

**Пример:**

*log['date'] = pd.to\_datetime(log['timestamp'], unit='s')*

*log.loc[0 : 4, ['timestamp', 'date']].head()*

*timestamp date*

*0 1549980692 2019-02-12 14:11:32*

*1 1549980704 2019-02-12 14:11:44*

*2 1549980715 2019-02-12 14:11:55*

*3 1549980725 2019-02-12 14:12:05*

*4 1549980736 2019-02-12 14:12:16*

Обращаться к отдельным элементам даты можно через свойства аксессора **.dt()**.

**Пример:**

*log['hour'] = log.date.dt.hour*

*log.loc[0 : 4, ['timestamp', 'date', 'hour']].head()*

*timestamp date hour*

*0 1549980692 2019-02-12 14:11:32 14*

*1 1549980704 2019-02-12 14:11:44 14*

*2 1549980715 2019-02-12 14:11:55 14*

*3 1549980725 2019-02-12 14:12:05 14*

*4 1549980736 2019-02-12 14:12:16 14*

Что касается работы со строками, то чаще всего возникает задача поиска строки в датафрейме. Для этого есть метод **.contains()** аксессора .str(), который является аналогом оператора in. Метод принимает как обычные строки, так и регулярные выражения.

**Пример:**

*stats = pd.read\_csv('keywords.csv')*

*stats.head()*

*keyword shows*

*0 вк 64292779*

*1 одноклассники 63810309*

*2 порно 41747114*

*3 ютуб 39995567*

*4 вконтакте 21014195*

*stats[stats.keyword.str.contains('охотник')].head()*

*keyword shows*

*3072 сумеречные охотники 2 сезон 71965*

*3474 сумеречные охотники 66083*

*3654 белоснежка и охотник 2 фильм 2016 63154*

*4178 последний охотник на ведьм 57560*

*6127 последний охотник на ведьм фильм 2015 42213*

*# В том числе поиск возможен по одному из нескольких слов*

*stats[stats.keyword.str.contains('охотник|фильм|2016')].head()*

*keyword shows*

*20 фильмы 2016 4486635*

*51 фильмы 2156641*

*54 фильмы онлайн 2305540*

*68 смотреть фильмы онлайн 1928484*

*86 порно фильмы 1458031*

Помимо поиска возможна также и замена (аналог метода .replace()). Чтобы при поиске не было проблем из-за регистра, можно привести все строки к верхнему или нижнему регистру методами **.str.upper()** и **.str.lower()**.

**Пример:**

*serial = stats[stats.keyword.str.contains('сериалы')]*

*serial.head()*

*keyword shows*

*246 сериалы 587779*

*304 сериалы тут 503967*

*555 турецкие сериалы на русском языке 296403*

*881 русские сериалы 234262*

*890 сериалы онлайн 204812*

*serial.keyword.str.replace('сериалы', 'книги').head()*

246 книги

304 книги тут

555 турецкие книги на русском языке

881 русские книги

890 книги онлайн

*serial.keyword.str.upper().head()*

246 СЕРИАЛЫ

304 СЕРИАЛЫ ТУТ

555 ТУРЕЦКИЕ СЕРИАЛЫ НА РУССКОМ ЯЗЫКЕ

881 РУССКИЕ СЕРИАЛЫ

890 СЕРИАЛЫ ОНЛАЙН

*serial.keyword.str.lower().head()*

246 сериалы

304 сериалы тут

555 турецкие сериалы на русском языке

881 русские сериалы

890 сериалы онлайн

Для того, чтобы учесть все формы написания желательно использовать регулярные выражения. Более простой вариант, поиск по основе слова, не является надёжным, поскольку не сможет отфильтровать слова, содержащие в себе ту же основу.

Кроме того можно использовать методы анализа, содержащиеся в библиотеках nltk и pymystem (не входят в Anaconda, необходимо установить отдельно). Последняя особенно хорошо работает с русским языком.

**Пример:**

*from pymysystem import Mystem*

*search = ‘курс евро к доллару доллара долларов’*

*m = Mystem()*

*lemmas = m.lemmatize(search)*

*print(lemmas) # [‘курс’, ‘ ’, ‘евро’, ‘ ’, ‘к’, ‘ ’, ‘доллар’, ‘ ’, ‘доллар’, ‘ ’, ‘доллар’, ‘\n’]*

Подробнее о pymystem:

* [*https://pypi.org/project/pymystem3/*](https://pypi.org/project/pymystem3/)
* [*https://yandex.ru/dev/mystem/*](https://yandex.ru/dev/mystem/)