**Основы Pandas**

Темы на сегодня:

* Обзор библиотеки Pandas
* Базовые операции
* Дополнительно: режим презентации в Jupyter Notebook

ОБЗОР БИБЛИОТЕКИ PANDAS

Pandas (Python Data Analysis Library) – мощная библиотека для Python, которая предназначена для анализа больших данных. Вполне резонно возникает вопрос: «NumPy и SciPy являются отличными библиотеками с широкими возможностями – зачем же нужна ещё одна библиотека?» Дело в том, что Pandas импортирует данные иным способом – в т.н. датафреймы. Причём **импортирует практически любые данные**, например:

* CSV
* Excel
* Данные буфера обмена (Ctrl+C)
* HTML-таблицы (<table></table>)
* JSON
* Apache Parquet
* SQL-запросы

Сами по себе датафреймы **хранятся в оперативной памяти**. Таким образом, объём данных, с которыми можно работать, ограничен свободной оперативной памятью. Например, если из 4 ГБ ОЗУ доступно 2 ГБ, то можно работать с объёмом до 1,5 ГБ. При этом работает библиотека довольно шустро. Сфера применения – решение аналитических задач, в production-коде скорее всего она будет недостаточно быстра.

Данные в датафрейме **удобно просматривать** на любом этапе: при загрузке, во время вычислений, по окончании вычислений. Кроме того, часть функций по работе с датафреймами **схожи с синтаксисом SQL**, но при этом не требуется база данных. Например:

* JOIN - .join и .merge
* GROUPBY – .groupby
* ORDER BY – .sort\_values

Pandas поддерживает **цепочки вычислений** – не требуется сохранять промежуточные вычисления в отдельных переменных, можно применять методы друг за другом.

**Пример:**

*data.groupby(‘country’).count().sort\_values(‘quantity’)*

Тем не менее одно лишь умение работать с Pandas не поможет решать задачи. Понадобится также весь базовый функционал Python. Совместное использование позволит производить **вычисления любой сложности**.

БАЗОВЫЕ ОПЕРАЦИИ

Импортирование библиотеки происходит привычным образом. Общепринятое сокращение – pd. Далее необходимо загрузить данные. Для этого существует ряд методов вида .read\_источник. Одним из самых простых способов импорта является импорт с сайта:

**Пример:**

*import pandas as pd*

*pd.read\_html('http://www.cbr.ru')[2]*

*Unnamed: 0 31.12.2019 01.01.2020*

*0 Доллар США $ руб. 61,9057 руб. 61,9057*

*1 Евро € руб. 69,3406 руб. ↑69,3777*

Другим вариантом является импорт из файла. Рассмотрим на примере CSV-файла. После загрузки можно возвратить данные методом **.head(число\_строк)**, результатом работы которого будет датафрейм. Метод .head() возвращает данные с начала таблицы. Если необходимо взять данные с конца, то есть аналогичный метод **.tail(число строк)**.

**Пример:**

*import pandas as pd*

*ratings = pd.read\_csv(‘file.csv’)*

*ratings.head(10)*

*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*

*5 1 70 3.0 964982400*

*6 1 101 5.0 964980868*

*7 1 110 4.0 964982176*

*8 1 151 5.0 964984041*

*9 1 157 5.0 964984100*

Существуют дополнительные параметры, которые позволяют производить простейшее форматирование прямо на лету (загружать нужные элементы, менять имена столбцов, менять кодировку и т.д.).

**Пример:**

*data = pd.read\_csv('power.csv', names = ['страна', 'год', 'количество', 'категория'], sep = ',', header=0)*

*data.head()*

*страна год количество категория*

*0 Austria 1996 5.0 1*

*1 Austria 1995 17.0 1*

*2 Belgium 2014 0.0 1*

*3 Belgium 2013 0.0 1*

*4 Belgium 2012 35.0 1*

Метод **.info()** позволяет получить более подробную информацию о столбцах: длину, тип и т.д. Метод **.describe()** позволяет получить некоторую статистическую информацию.

**Пример:**

*ratings.info()*

*<class 'pandas.core.frame.DataFrame'>*

*RangeIndex: 100836 entries, 0 to 100835*

*Data columns (total 4 columns):*

*userId 100836 non-null int64*

*movieId 100836 non-null int64*

*rating 100836 non-null float64*

*timestamp 100836 non-null int64*

*dtypes: float64(1), int64(3)*

*memory usage: 3.1 MB*

*ratings.describe()*

*userId movieId rating timestamp*

*count 100836.000000 100836.000000 100836.000000 1.008360e+05*

*mean 326.127564 19435.295718 3.501557 1.205946e+09*

*std 182.618491 35530.987199 1.042529 2.162610e+08*

*min 1.000000 1.000000 0.500000 8.281246e+08*

*25% 177.000000 1199.000000 3.000000 1.019124e+09*

*50% 325.000000 2991.000000 3.500000 1.186087e+09*

*75% 477.000000 8122.000000 4.000000 1.435994e+09*

*max 610.000000 193609.000000 5.000000 1.537799e+09*

Обратиться к конкретному столбцу можно через точку или через квадратные скобки. В таком случае каждый столбец возвращается в виде типа Pandas Series – одномерного массива, который можно привести к списку.

**Пример:**

*ratings.movieId # Или ratings[‘movieId’]*

*0 1*

*1 3*

*2 6*

*3 47*

*4 50*

*...*

*# Количество фильмов в коллекции: берём столбец movieId, оставляем только уникальные значения и ищем длину полученного Series*

*len(ratings.movieId.unique()) # 9724*

*ratings.movieId.unique().shape [0] # Или так*

Несколько базовых методов:

* **.groupby(столбец)** – производит группировку по значениям определённого столбца
* **.count()** – производит подсчёт количества. Для сгруппированных датафреймов произведёт подсчёт строк для каждой группы
* **.sort\_values(столбец, направление)** – производит сортировку по заданному столбцу (столбцам). Параметр ascending (направление) принимает True, если сортировка по возрастанию или False, если по убыванию. По умолчанию – True

**Пример:**

*ratings.groupby('userId').count().sort\_values('movieId', ascending=False).head()*

*userId movieId rating timestamp*

*414 2698 2698 2698*

*599 2478 2478 2478*

*474 2108 2108 2108*

*448 1864 1864 1864*

*274 1346 1346 1346`*

* **.value\_counts()** – считает количество строк по значению столбца. Аналогично комбинации .groupby(столбец).count()

**Пример:**

*ratings['userId'].value\_counts().sort\_values(ascending=False).head()*

*414 2698*

*599 2478*

*474 2108*

*448 1864*

*274 1346*

*Name: userId, dtype: int64*

* **.filter(параметры) –** производит фильтрацию таблицы. Самый простой принцип – по столбцам. Условие можно задать регулярным выражением

**Пример:**

*ratings.filter(items = ['movieId', 'rating']).head()*

*# Аналогично: ratings[[‘movieId’, ‘rating’]]*

*movieId rating*

*0 1 4.0*

*1 3 4.0*

*2 6 4.0*

*3 47 5.0*

*4 50 5.0*

* **.query(условие) –** возвращает датафрейм из столбцов, удовлетворяющих условию в скобках

**Пример:**

*ratings.query('movieId == 4')*

*# Аналогично: ratings[ratings.movieId == 4]*

*userId movieId rating timestamp*

*562 6 4 3.0 845554349*

*1386 14 4 3.0 835441653*

*13075 84 4 3.0 858772461*

*23702 162 4 3.0 836684306*

*38179 262 4 1.0 840306203*

*62011 411 4 2.0 835533021*

*95103 600 4 1.5 1237760055*

* **.agg(параметры)** – функция высшего порядка, применяющая к датафрейму функции, помещённые в качестве параметров

**Пример:**

*ratings\_grouped = ratings.groupby('userId').agg([min, max]).head()*

*movieId rating timestamp*

*min max min max min max*

*userId*

*1 1 5060 1.0 5.0 964980499 965719662*

*2 318 131724 2.0 5.0 1445714835 1445715340*

*3 31 72378 0.5 5.0 1306463323 1306464293*

*4 21 4967 1.0 5.0 945078428 1007574542*

*5 1 608 1.0 5.0 847434747 847435337*

* **.loc[начало:конец]** – возвращает указанный диапазон строк таблицы

**Пример:**

*ratings.loc[5:10]*

*userId movieId rating timestamp*

*5 1 70 3.0 964982400*

*6 1 101 5.0 964980868*

*7 1 110 4.0 964982176*

*8 1 151 5.0 964984041*

*9 1 157 5.0 964984100*

*10 1 163 5.0 964983650*

* **.merge(датафрейм\_2, on=столбец, how=метод)** – возвращает датафрейм, составленный «склеиванием» датафреймов по указанному столбцу необходимым методом (аналогично JOIN в SQL)

**Пример:**

*movies = pd.read\_csv('ml-latest/movies.csv')*

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

*userId movieId rating timestamp title genres*

*0 1 1 4.0 964982703 Toy Story (1995) Animation*

*1 1 3 4.0 964981247 Grumpier Old Men (1995) Comedy*

*2 1 6 4.0 964982224 Heat (1995) Thriller*

*3 1 47 5.0 964983815 Seven (a.k.a. Se7en) (1995) Mystery*

*4 1 50 5.0 964982931 Usual Suspects, The (1995) Crime*

Арифметические операции (+, - и т.д.) выполняются построчно. Результат можно сохранить в новый столбец. Время работы функции по-прежнему можно узнать через %%time.

**Пример:**

*%%time*

*ratings\_grouped['diff'] = ratings\_grouped['timestamp']['max'] - ratings\_grouped['timestamp']['min']*

*ratings\_grouped.head()*

***Wall time: 12.7 ms***

*movieId rating timestamp diff*

*min max min max min max*

*userId*

*1 1 5060 1.0 5.0 964980499 965719662 739163*

*2 318 131724 2.0 5.0 1445714835 1445715340 505*

*3 31 72378 0.5 5.0 1306463323 1306464293 970*

*4 21 4967 1.0 5.0 945078428 1007574542 62496114*

*5 1 608 1.0 5.0 847434747 847435337 590*

Стоит заметить, что многие операции в Pandas имеют дополнительный параметр **inplace**. По умолчанию он равен False. При значении True каждая операция, применённая к данным будет их перезаписывать. Это позволяет сократить время выполнения цепочки действий. Использовать рекомендуется с осторожностью и лишь по необходимости.

РЕЖИМ ПРЕЗЕНТАЦИИ

Иногда файл .ipynb в Jupyter Notebook удобнее просматривать в режиме презентации. Данная функция не работает в других средах (PyCharm и тд.). Алгоритм запуска:

1. Открываем вкладку View на главной панели
2. Выбираем меню Cell Toolbar
3. Нажимаем на пункт Slideshow. После этого ячейки примут тип Slide. У каждой ячейки появится выпадающий список, в котором можно выбрать поведение конкретного слайда (Обычный, Пропуск слайда, Заметки и т.д.)
4. Выполняем команду:  
   *!путь\_до\_jupyter/jupyter nbconvert имя\_файла –to slides –post serve*
5. Произойдёт конвертирование в html-файл, который запустится также в браузере

*Более подробно можно прочесть в статье (англ.):* [*https://medium.com/@mjspeck/presenting-code-using-jupyter-notebook-slides-a8a3c3b59d67*](https://medium.com/@mjspeck/presenting-code-using-jupyter-notebook-slides-a8a3c3b59d67)