Работа с временными рядами

https://dfedorov.spb.ru

Нативные даты и время языка Python: пакеты datetime и dateutil

Базовые объекты Python для работы с датами и временем располагаются во встроенном пакете **datetime**. Его, вместе со сторонним модулем **dateutil**, можно использовать для быстрого выполнения множества удобных операций над датами и временем.
Например, можно вручную сформировать дату с помощью типа **datetime** :

```
>>> from datetime import datetime
>>> datetime(year=2015, month=7, day=4)
datetime.datetime(2015, 7, 4, 0, 0)
```

Или, воспользовавшись модулем **dateutil** , можно выполнять синтаксический разбор дат, находящихся во множестве строковых форматов:

```
>>> from dateutil import parser
>>> date = parser.parse("4th of July, 2015")
>>> date
datetime.datetime(2015, 7, 4, 0, 0)
```

При наличии объекта **datetime** можно делать вывод дня недели:

```
>>> date.strftime('%A')
'Saturday'
```

Сила пакетов datetime и dateutil заключается в их гибкости и удобном синтаксисе: эти объекты и их встроенные методы можно использовать для выполнения практически любой интересующей вас операции. Единственное, в чем они работают плохо, это работа с большими массивами дат и времени: подобно спискам числовых переменных языка Python, работающим неоптимально по сравнению с типизированными числовыми массивами в стиле библиотеки NumPy, списки объектов даты/времени Python работают с меньшей производительностью, чем типизированные массивы кодированных дат.

Модуль для работы с часовыми поясами

Типизированные массивы значений времени: тип datetime64 библиотеки NumPy

Указанная слабая сторона формата даты/времени языка Python побудила команду разработчиков библиотеки NumPy добавить набор нативных типов данных временных рядов.
Тип (dtype) datetime64 кодирует даты как 64-битные целые числа, так что представление массивов дат оказывается очень компактным.
Для типа datetime64 требуется очень точно заданный формат входных данных:

```
>>> import numpy as np
>>> date = np.array('2015-07-04', dtype=np.datetime64)
>>> date
```

```
array('2015-07-04', dtype='datetime64[D]')
```

Но как только дата отформатирована, можно быстро выполнять над ней различные векторизованные операции:

```
>>> date + np.arange(12)
```

```
array(['2015-07-04', '2015-07-05', '2015-07-06', '2015-07-07',
       '2015-07-08', '2015-07-09', '2015-07-10', '2015-07-11',
       '2015-07-12', '2015-07-13', '2015-07-14', '2015-07-15'],
      dtype='datetime64[D]')
```

Поскольку datetime64-массивы библиотеки NumPy содержат данные одного типа, подобные операции выполняются намного быстрее, чем если работать непосредственно с объектами datetime языка Python, особенно если речь идет о больших массивах.

Библиотека NumPy определяет требуемую единицу на основе входной информации; например, вот дата/время на основе единицы в один день:

```
>>> np.datetime64('2015-07-04')
```

```
numpy.datetime64('2015-07-04')
```

Вот дата/время на основе единицы в одну минуту:

```
>>> np.datetime64('2015-07-04 12:00')
```

```
numpy.datetime64('2015-07-04T12:00')
```

Отметим, что, хотя тип данных datetime64 лишен некоторых недостатков встроенного типа данных datetime языка Python, ему недостает многих предоставляемых datetime и особенно dateutil удобных методов и функций.

В табл. перечислены доступные для использования коды форматирования, а также относительные и абсолютные промежутки времени, которые можно кодировать с их помощью.

Код	Значение	Промежуток времени (относительный)	Промежуток времени (абсолютный)
Y	Год	±9.2e18 лет	[9.2e18 до н. э., 9.2e18 н. э.]
M	Месяц	±7.6e17 лет	[7.6e17 до н. э., 7.6e17 н. э.]
W	Неделя	±1.7e17 лет	[1.7e17 до н. э., 1.7e17 н. э.]
D	День	±2.5e16 лет	[2.5e16 до н. э., 2.5e16 н. э.]
h	Час	±1.0e15 лет	[1.0e15 до н. э., 1.0e15 н. э.]
m	Минута	±1.7e13 лет	[1.7e13 до н. э., 1.7e13 н. э.]
s	Секунда	±2.9e12 лет	[2.9e9 до н. э., 2.9e9 н. э.]
ms	Миллисекунда	±2.9e9 лет	[2.9e6 до н. э., 2.9e6 н. э.]
us	Микросекунда	±2.9e6 лет	[290301 до н. э., 294241 н. э.]
ns	Наносекунда	±292 лет	[1678 до н. э., 2262 н. э.]
ps	Пикосекунда	±106 дней	[1969 до н. э., 1970 н. э.]
fs	Фемтосекунда	±2.6 часов	[1969 до н. э., 1970 н. э.]
as	Аттосекунда	±9.2 секунды	[1969 до н. э., 1970 н. э.]

Даты и время в библиотеке Pandas: избранное из лучшего

Библиотека Pandas предоставляет объект **Timestamp** , сочетающий удобство использования **datetime** и **dateutil** с эффективным хранением и векторизованным интерфейсом типа **numpy.datetime64** . Библиотека Pandas умеет создавать из нескольких таких объектов **Timestamp** объект класса **DatetimeIndex** , который можно использовать для индексации данных в объектах **Series** или **DataFrame** . Можно применить инструменты библиотеки Pandas для воспроизведения вышеприведенной наглядной демонстрации. Можно выполнить синтаксический разбор строки с датой в гибком формате и воспользоваться кодами форматирования, чтобы вывести день недели:

```
>>> import pandas as pd
>>> date = pd.to_datetime("4th of July, 2015")
>>> date
Timestamp('2015-07-04 00:00:00')

>>> date.strftime('%A')
'Saturday'
```

Кроме этого, можно выполнять векторизованные операции в стиле библиотеки NumPy непосредственно над этим же объектом:

```
>>> pd.to_timedelta(np.arange(12), 'D')
TimedeltaIndex([ '0 days',  '1 days',  '2 days',  '3 days',  '4 days',
                 '5 days',  '6 days',  '7 days',  '8 days',  '9 days',
                '10 days', '11 days'],
               dtype='timedelta64[ns]', freq=None)

>>> date + pd.to_timedelta(np.arange(12), 'D')
DatetimeIndex(['2015-07-04', '2015-07-05', '2015-07-06', '2015-07-07',
               '2015-07-08', '2015-07-09', '2015-07-10', '2015-07-11',
               '2015-07-12', '2015-07-13', '2015-07-14', '2015-07-15'],
              dtype='datetime64[ns]', freq=None)
```

Временные ряды библиотеки Pandas: индексация по времени

Инструменты для работы с временными рядами библиотеки Pandas особенно удобны при необходимости индексации данных по меткам даты/времени. Например, создадим объект **Series** с индексированными по времени данными:

```
>>> index = pd.DatetimeIndex(['2014-07-04', '2014-08-04', '2015-07-04', '2015-08-04'])
>>> index
DatetimeIndex(['2014-07-04', '2014-08-04', '2015-07-04', '2015-08-04'], dtype='datetime64[ns]', freq=None)

>>> data = pd.Series([0, 1, 2, 3], index=index)
>>> data
2014-07-04    0
2014-08-04    1
2015-07-04    2
2015-08-04    3
dtype: int64

>>> data.loc['2014-07-04':'2015-07-04']
2014-07-04    0
2014-08-04    1
2015-07-04    2
dtype: int64

>>> data.loc['2015']
2015-07-04    2
2015-08-04    3
dtype: int64
```

Библиотека Pandas была разработана в расчете на построение финансовых моделей, так что, как вы могли и ожидать, она содержит весьма широкий набор инструментов для работы с датой, временем и индексированными по времени данными. Данные о дате и времени могут находиться в нескольких видах.

- **Метки даты/времени** ссылаются на конкретные моменты времени (например, 4 июля 2015 года в 07:00 утра).
- **Временные интервалы и периоды** ссылаются на отрезки времени между конкретными начальной и конечной точками (например, 2015 год). Периоды обычно представляют собой особый случай интервалов, с непересекающимися интервалами одинаковой длительности (например, 24-часовые периоды времени, составляющие сутки).
- **Временная дельта** (она же продолжительность) относится к отрезку времени конкретной длительности (например, 22,56 с).

https://nagornyy.me/courses/data-science/dates-and-time-in-python-and-pandas/

Структуры данных для временных рядов библиотеки Pandas

- Для меток даты/времени библиотека Pandas предоставляет тип данных **Timestamp**. Этот тип является заменой для нативного типа данных **datetime** языка Python, он основан на более эффективном типе данных **numpy.datetime64**. Соответствующая индексная конструкция — **DatetimeIndex**.

- Для периодов времени библиотека Pandas предоставляет тип данных **Period**. Он на основе типа данных **numpy.datetime64** кодирует интервал времени фиксированной периодичности. Соответствующая индексная конструкция — **PeriodIndex** .

- Для временных дельт (продолжительностей) библиотека Pandas предоставляет тип данных **Timedelta** . **Timedelta** — основанная на типе **numpy.timedelta64** более эффективная замена нативного типа данных datetime.timedelta языка Python. Соответствующая индексная конструкция — **TimedeltaIndex** .

Самые базовые из объектов даты/времени — объекты **Timestamp** и **DatetimeIndex**. Хотя к ним и можно обращаться непосредственно, чаще используют функцию **pd.to_datetime()** , умеющую выполнять синтаксический разбор широкого диапазона форматов. При передаче в функцию **pd.to_datetime()** отдельной даты она возвращает **Timestamp** , при передаче ряда дат по умолчанию возвращает **DatetimeIndex** :

```
>>> from datetime import datetime
>>> import pandas as pd
>>> dates = pd.to_datetime([datetime(2015, 7, 3), '4th of July, 2015', '2015-Jul-6', '07-07-2015', '20150708'])
>>> dates
DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-06', '2015-07-07', '2015-07-08'],
              dtype='datetime64[ns]', freq=None)
```

Любой объект **DatetimeIndex** можно с помощью функции **to_period()** преобразовать в объект **PeriodIndex**, указав код для периодичности интервала. В данном случае мы использовали код 'D' , означающий, что периодичность интервала — один день:

```
>>> dates.to_period('D')
PeriodIndex(['2015-07-03', '2015-07-04', '2015-07-06', '2015-07-07', '2015-07-08'],
            dtype='period[D]', freq='D')
```

Объект **TimedeltaIndex** создается, например, при вычитании одной даты из другой:

```
>>> dates - dates[0]
TimedeltaIndex(['0 days', '1 days', '3 days', '4 days', '5 days'], dtype='timedelta64[ns]', freq=None)
```

Регулярные последовательности: функция pd.date_range()

Чтобы облегчить создание регулярных последовательностей, Pandas предоставляет несколько функций:

pd.date_range() — для меток даты/времени,
pd.period_range() — для периодов времени и
pd.timedelta_range() — для временных дельт.

Функция pd.date_range() создает регулярную последовательность дат, принимая на входе начальную дату, конечную дату и необязательный код периодичности. По умолчанию период равен одному дню:

```
>>> pd.date_range('2015-07-03', '2015-07-10')
DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06',
               '2015-07-07', '2015-07-08', '2015-07-09', '2015-07-10'],
              dtype='datetime64[ns]', freq='D')
```

В качестве альтернативы можно также задать диапазон дат с помощью не начальной и конечной точек, а посредством начальной точки и количества периодов времени:

```
>>> pd.date_range('2015-07-03', periods=8)
DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06',
               '2015-07-07', '2015-07-08', '2015-07-09', '2015-07-10'],
              dtype='datetime64[ns]', freq='D')
```

Можно изменить интервал времени, поменяв аргумент freq , имеющий по умолчанию значение 'D' . Например, в следующем фрагменте мы создаем диапазон часовых меток даты/времени:

```
>>> pd.date_range('2015-07-03', periods=8, freq='H')
DatetimeIndex(['2015-07-03 00:00:00', '2015-07-03 01:00:00',
               '2015-07-03 02:00:00', '2015-07-03 03:00:00',
               '2015-07-03 04:00:00', '2015-07-03 05:00:00',
               '2015-07-03 06:00:00', '2015-07-03 07:00:00'],
              dtype='datetime64[ns]', freq='H')
```

Создадим массив временных меток с шагом в один час, который охватывает интервал в пять дней:
>>> dt_h = pd.date_range(start='2017-02-01', freq='H', periods=120)
>>> dt_h[:30]

Для создания регулярных последовательностей значений периодов или временных дельт можно воспользоваться функциями pd.period_range() и pd.timedelta_range() , напоминающими функцию date_range() . Вот несколько периодов времени длительностью в месяц:

```
>>> pd.period_range('2015-07', periods=8, freq='M')
PeriodIndex(['2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12',
             '2016-01', '2016-02'],
            dtype='period[M]', freq='M')
```

Вот последовательность продолжительностей, увеличивающихся на час:

```
>>> pd.timedelta_range(0, periods=10, freq='H')
TimedeltaIndex(['0 days 00:00:00', '0 days 01:00:00', '0 days 02:00:00',
                '0 days 03:00:00', '0 days 04:00:00', '0 days 05:00:00',
                '0 days 06:00:00', '0 days 07:00:00', '0 days 08:00:00',
                '0 days 09:00:00'],
               dtype='timedelta64[ns]', freq='H')
```

Периодичность и смещения дат

Список кодов периодичности библиотеки Pandas

Код	Описание	Код	Описание
D	Календарный день	B	Рабочий день
W	Неделя		
M	Конец месяца	BM	Конец отчетного месяца
Q	Конец квартала	BQ	Конец отчетного квартала
A	Конец года	BA	Конец финансового года
H	Час	BH	Рабочие часы
T	Минута		
S	Секунда		
L	Миллисекунда		
U	Микросекунда		
N	Наносекунда		

Периодичность в месяц, квартал и год определяется на конец соответствующего периода. Добавление к любому из кодов суффикса S приводит к определению начала периода

Список стартовых кодов периодичности

Код	Описание
MS	Начало месяца
BMS	Начало отчетного месяца
QS	Начало квартала

Код	Описание
BQS	Начало отчетного квартала
AS	Начало года
BAS	Начало финансового года

Кроме этого, можно изменить используемый для определения квартала или года месяц с помощью добавления в конец кода месяца, состоящего из трех букв:

- Q-JAN , BQ-FEB , QS-MAR , BQS-APR и т. д.
- A-JAN , BA-FEB , AS-MAR , BAS-APR и т. д.

Аналогичным образом можно изменить точку разбиения для недельной периодичности, добавив состоящий из трех букв код дня недели:

W-SUN , W-MON , W-TUE , W-WED и т. д.

Для указания иной периодичности можно сочетать коды с числами. Например, для периодичности 2 часа 30 минут можно скомбинировать коды для часа (H) и минуты (T):

```
>>> pd.timedelta_range(0, periods=9, freq="2H30T")
TimedeltaIndex(['0 days 00:00:00', '0 days 02:30:00', '0 days 05:00:00',
                '0 days 07:30:00', '0 days 10:00:00', '0 days 12:30:00',
                '0 days 15:00:00', '0 days 17:30:00', '0 days 20:00:00'],
               dtype='timedelta64[ns]', freq='150T')
```

Все эти короткие коды ссылаются на соответствующие экземпляры смещений даты/времени временных рядов библиотеки Pandas, которые можно найти в модуле **pd.tseries.offsets** . Например, можно непосредственно создать смещение в один рабочий день следующим образом:

```
>>> from pandas.tseries.offsets import BDay
>>> pd.date_range('2015-07-01', periods=5, freq=BDay())
DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-06',
               '2015-07-07'],
              dtype='datetime64[ns]', freq='B')
```

Передискретизация, временные сдвиги и окна

Pandas, будучи разработанной в значительной степени для работы с финансовыми данными, имеет для этой цели несколько весьма специфических инструментов. Например, сопутствующий Pandas пакет pandas-datareader умеет импортировать финансовые данные из множества источников.

Московская биржа (MOEX) предоставляет исторические данные:

```
>>> import pandas_datareader.data as web
>>> f = web.DataReader('USD000UTSTOM', 'moex', start='2017-07-01', end='2017-07-31')
>>> f.head()
```

TRADEDATE	BOARDID	CLOSE	HIGH	LOW	NUMTRADES	OPEN	SECID	SHORTNAME	VOLRUR	WAPRICE
2017-07-03	CETS	59.2650	59.4825	58.790	29108	58.95	USD000UTSTOM	USDRUB_TOM	1.472424e+11	59.1903
2017-07-03	CNGD	59.3600	59.4250	58.840	24	58.98	USD000UTSTOM	USDRUB_TOM	1.864785e+09	NaN
2017-07-04	CETS	59.4125	59.4575	59.135	21053	59.30	USD000UTSTOM	USDRUB_TOM	1.090265e+11	59.2700
2017-07-04	CNGD	59.3575	59.3600	58.930	37	59.36	USD000UTSTOM	USDRUB_TOM	1.046416e+09	NaN
2017-07-05	CETS	59.9825	60.2600	59.300	50108	59.30	USD000UTSTOM	USDRUB_TOM	2.874226e+11	59.9234

Для простоты будем использовать только окончательную цену:

```
>>> f = f.CLOSE
>>> f
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()
f.plot();
```