01-Datetime-Basics

October 19, 2022

For more information, visit us at www.pieriandata.com

For most of this course we will be loading datasets into `pandas`, and we'll seldom worry about the format that dates take. This is because the `pandas` native data type (brought over from NumPy) is more compact and runs far more efficiently than Python's built-in datetime object.Still, it can't hurt to understand `datetime` objects.

1 The `datetime` module

Python has built-in date, time and datetime objects available through the `datetime` module For more info on datetime visit https://docs.python.org/3/library/datetime.html

[1]:
```
# Import the entire module:
import datetime
```

1.0.1 datetime `time` objects

Values can be passed in as keyword arguments…

[2]:
```
tm = datetime.time(hour=5,minute=25,second=1)
tm
```

[2]:
```
datetime.time(5, 25, 1)
```

…or as positional arguments.

[3]:
```
tm = datetime.time(5,25,1)
tm
```

[3]:
```
datetime.time(5, 25, 1)
```

[4]:
```
print(tm)
```

```
05:25:01
```

[5]:
```python
type(tm)
```

[5]:
```
datetime.time
```

1.0.2 datetime date objects

[6]:
```python
dt = datetime.date(2019,1,2)
dt
```

[6]:
```
datetime.date(2019, 1, 2)
```

[7]:
```python
print(dt)
```

```
2019-01-02
```

[8]:
```python
type(dt)
```

[8]:
```
datetime.date
```

1.0.3 datetime datetime objects

[9]:
```python
d = datetime.datetime(2019, 1, 2, 5, 25, 1)
d
```

[9]:
```
datetime.datetime(2019, 1, 2, 5, 25, 1)
```

[10]:
```python
print(d)
```

```
2019-01-02 05:25:01
```

[11]:
```python
type(d)
```

[11]:
```
datetime.datetime
```

When no time data is provided, minimum values are used:

[12]:
```python
d = datetime.datetime(2019, 2, 2)
print(d)
```

```
2019-02-02 00:00:00
```

1.0.4 Selective import

For efficiency, we can import just those object classes we plan to use.

[13]:

```
from datetime import datetime, date, time

d = datetime(2019, 3, 1, 15, 10)     # this is easier to type
print(d)
```

```
2019-03-01 15:10:00
```

1.1 date, time, and datetime components

We can access specific elements of the date and time within each object.

[14]:

```
print(tm)
print(tm.minute)
```

```
05:25:01
25
```

[15]:

```
print(dt)
print(dt.day)
```

```
2019-01-02
2
```

[16]:

```
print(d)
print(d.second)
```

```
2019-03-01 15:10:00
0
```

Of course, time objects don't contain date information, and date objects don't store time.

[17]:

```
print(tm.day)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-17-e82d43f80cff> in <module>()
----> 1 print(tm.day)

AttributeError: 'datetime.time' object has no attribute 'day'
```

[18]:

```
print(dt.second)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
```

```
<ipython-input-18-97b14771deb1> in <module>()
----> 1 print(dt.second)

AttributeError: 'datetime.date' object has no attribute 'second'
```

1.2 Today's date

Both `date` and `datetime` objects offer a `.today()` method that returns the current date as determined by the computer system clock.

[19]:
```
x = date.today()
print(x)
```

```
2019-01-03
```

[20]:
```
y = datetime.today()
print(y)
```

```
2019-01-03 12:15:05.526582
```

Note that assignments take a snapshot of the current date and store it. This value doesn't move forward with time.

[21]:
```
print(y)
```

```
2019-01-03 12:15:05.526582
```

1.3 Useful methods

[22]:
```
d = datetime(1969,7,20,20,17)
```

`d.weekday()` returns the day of the week as an integer, where Monday is 0 and Sunday is 6

[23]:
```
d.weekday()
```

[23]:
```
6
```

`d.isoweekday()` returns the day of the week as an integer, where Monday is 1 and Sunday is 7

[24]:
```
d.isoweekday()
```

[24]:
```
7
```

`d.replace()` returns a modified copy of the original, permitting substitutions for any date/time attribute

[25]:
```
d.replace(year=1975,month=3)
```

```
[25]: datetime.datetime(1975, 3, 20, 20, 17)
```

Note that d.replace() does not change the original.

```
[26]: print(d)
```

```
1969-07-20 20:17:00
```

1.4 Time tuples

`datetime.timetuple()` returns a named tuple of values. Note that `date.timetuple()` returns 0 values for time elements.

```
[32]: r = date(2004,10,27)
      s = datetime(2004,10,27,20,25,55)
```

```
[33]: r.timetuple()
```

```
[33]: time.struct_time(tm_year=2004, tm_mon=10, tm_mday=27, tm_hour=0, tm_min=0,
      tm_sec=0, tm_wday=2, tm_yday=301, tm_isdst=-1)
```

```
[34]: s.timetuple()
```

```
[34]: time.struct_time(tm_year=2004, tm_mon=10, tm_mday=27, tm_hour=20, tm_min=25,
      tm_sec=55, tm_wday=2, tm_yday=301, tm_isdst=-1)
```

TIME TUPLE VALUES

NAME

EQUIVALENT

EXAMPLES

tm_year

d.year

2004

tm_mon

d.month

10

tm_mday

d.day

27

tm_hour

d.hour

20

tm_min

d.minute

25

tm_sec

d.second

55

tm_wday

d.weekday()

2

tm_yday

see below

301

tm_yday is the number of days within the current year starting with 1 for January 1st, as given by the formula yday = d.toordinal() - date(d.year, 1, 1).toordinal() + 1 **tm_isdst** relates to timezone settings which we'll cover in an upcoming section.

1.4.1 This just scratches the surface

There's a lot we can do with Python datetime objects as far as formatting their appearance, parsing incoming text with the 3rd party dateutil module, and more. For now, we'll leave this alone and focus on NumPy. NumPy's `datetime64` dtype encodes dates as 64-bit integers, so that arrays of dates are stored very compactly.