00-Datetime-Index

October 19, 2022

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

1 Introduction to Time Series with Pandas

Most of our data will have a datatime index, so let's learn how to deal with this sort of data with pandas!

1.1 Python Datetime Review

In the course introduction section we discussed Python datetime objects.

[1]:
```
from datetime import datetime
```

[2]:
```
# To illustrate the order of arguments
my_year = 2017
my_month = 1
my_day = 2
my_hour = 13
my_minute = 30
my_second = 15
```

[3]:
```
# January 2nd, 2017
my_date = datetime(my_year,my_month,my_day)
```

[4]:
```
# Defaults to 0:00
my_date
```

[4]:
```
datetime.datetime(2017, 1, 2, 0, 0)
```

[5]:
```
# January 2nd, 2017 at 13:30:15
my_date_time = datetime(my_year,my_month,my_day,my_hour,my_minute,my_second)
```

[6]:
```
my_date_time
```

[6]:
```
datetime.datetime(2017, 1, 2, 13, 30, 15)
```

You can grab any part of the datetime object you want

[7]:
```
my_date.day
```

[7]:
```
2
```

[8]:
```
my_date_time.hour
```

[8]:
```
13
```

1.2 NumPy Datetime Arrays

We mentioned that NumPy handles dates more efficiently than Python's datetime format. The NumPy data type is called datetime64 to distinguish it from Python's datetime.

In this section we'll show how to set up datetime arrays in NumPy. These will become useful later on in the course. For more info on NumPy visit https://docs.scipy.org/doc/numpy-1.15.4/reference/arrays.datetime.html

[9]:
```
import numpy as np
```

[10]:
```
# CREATE AN ARRAY FROM THREE DATES
np.array(['2016-03-15', '2017-05-24', '2018-08-09'], dtype='datetime64')
```

[10]:
```
array(['2016-03-15', '2017-05-24', '2018-08-09'], dtype='datetime64[D]')
```

NOTE: We see the dtype listed as 'datetime64[D]'. This tells us that NumPy applied a day-level date precision. If we want we can pass in a different measurement, such as [h] for hour or [Y] for year.

[11]:
```
np.array(['2016-03-15', '2017-05-24', '2018-08-09'], dtype='datetime64[h]')
```

[11]:
```
array(['2016-03-15T00', '2017-05-24T00', '2018-08-09T00'],
      dtype='datetime64[h]')
```

[12]:
```
np.array(['2016-03-15', '2017-05-24', '2018-08-09'], dtype='datetime64[Y]')
```

[12]:
```
array(['2016', '2017', '2018'], dtype='datetime64[Y]')
```

1.3 NumPy Date Ranges

Just as np.arange(start,stop,step) can be used to produce an array of evenly-spaced integers, we can pass a dtype argument to obtain an array of dates. Remember that the stop date is exclusive.

[13]:
```
# AN ARRAY OF DATES FROM 6/1/18 TO 6/22/18 SPACED ONE WEEK APART
np.arange('2018-06-01', '2018-06-23', 7, dtype='datetime64[D]')
```

[13]:
```
array(['2018-06-01', '2018-06-08', '2018-06-15', '2018-06-22'],
      dtype='datetime64[D]')
```

By omitting the step value we can obtain every value based on the precision.

[14]:
```
# AN ARRAY OF DATES FOR EVERY YEAR FROM 1968 TO 1975
np.arange('1968', '1976', dtype='datetime64[Y]')
```

[14]:
```
array(['1968', '1969', '1970', '1971', '1972', '1973', '1974', '1975'],
      dtype='datetime64[Y]')
```

1.4 Pandas Datetime Index

We'll usually deal with time series as a datetime index when working with pandas dataframes. Fortunately pandas has a lot of functions and methods to work with time series! For more on the pandas DatetimeIndex visit https://pandas.pydata.org/pandas-docs/stable/timeseries.html

[15]:
```
import pandas as pd
```

The simplest way to build a DatetimeIndex is with the pd.date_range() method:

[16]:
```
# THE WEEK OF JULY 8TH, 2018
idx = pd.date_range('7/8/2018', periods=7, freq='D')
idx
```

[16]:
```
DatetimeIndex(['2018-07-08', '2018-07-09', '2018-07-10', '2018-07-11',
               '2018-07-12', '2018-07-13', '2018-07-14'],
              dtype='datetime64[ns]', freq='D')
```

DatetimeIndex Frequencies: When we used pd.date_range() above, we had to pass in a frequency parameter 'D'. This created a series of 7 dates spaced one day apart. We'll cover this topic in depth in upcoming lectures, but for now, a list of time series offset aliases like 'D' can be found here.

Another way is to convert incoming text with the pd.to_datetime() method:

[17]:
```
idx = pd.to_datetime(['Jan 01, 2018','1/2/18','03-Jan-2018',None])
idx
```

[17]:
```
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', 'NaT'],
dtype='datetime64[ns]', freq=None)
```

A third way is to pass a list or an array of datetime objects into the pd.DatetimeIndex() method:

[18]:
```
# Create a NumPy datetime array
```

```
some_dates = np.array(['2016-03-15', '2017-05-24', '2018-08-09'],␣
 ↪dtype='datetime64[D]')
some_dates
```

[18]:
```
array(['2016-03-15', '2017-05-24', '2018-08-09'], dtype='datetime64[D]')
```

[19]:
```
# Convert to an index
idx = pd.DatetimeIndex(some_dates)
idx
```

[19]:
```
DatetimeIndex(['2016-03-15', '2017-05-24', '2018-08-09'],
dtype='datetime64[ns]', freq=None)
```

Notice that even though the dates came into pandas with a day-level precision, pandas assigns a nanosecond-level precision with the expectation that we might want this later on.

To set an existing column as the index, use .set_index() >df.set_index('Date',inplace=True)

1.5 Pandas Datetime Analysis

[20]:
```
# Create some random data
data = np.random.randn(3,2)
cols = ['A','B']
print(data)
```

```
[[-1.64971705  1.07943894]
 [ 0.4587492  -0.04201784]
 [-1.2793774  -1.85383771]]
```

[21]:
```
# Create a DataFrame with our random data, our date index, and our columns
df = pd.DataFrame(data,idx,cols)
df
```

[21]:

	A	B
2016-03-15	-1.649717	1.079439
2017-05-24	0.458749	-0.042018
2018-08-09	-1.279377	-1.853838

Now we can perform a typical analysis of our DataFrame

[22]:
```
df.index
```

[22]:
```
DatetimeIndex(['2016-03-15', '2017-05-24', '2018-08-09'],
dtype='datetime64[ns]', freq=None)
```

[23]:
```
# Latest Date Value
df.index.max()
```

[23]:
```
Timestamp('2018-08-09 00:00:00')
```

[24]:
```
# Latest Date Index Location
df.index.argmax()
```

[24]:
```
2
```

[25]:
```
# Earliest Date Value
df.index.min()
```

[25]:
```
Timestamp('2016-03-15 00:00:00')
```

[26]:
```
# Earliest Date Index Location
df.index.argmin()
```

[26]:
```
0
```

NOTE: Normally we would find index locations by running .idxmin() or .idxmax() on df['column'] since .argmin() and .argmax() have been deprecated. However, we still use .argmin() and .argmax() on the index itself.

1.6 Great, let's move on!