Time Series

Prof. Gheith Abandah

Reference

- **Chapter 11: Time Series**

- Wes McKinney, **Python for Data Analysis**: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media, 2nd Edition, 2018.
  - Material: https://github.com/wesm/pypop-book

Time Series

- Time series data is an **important** form of structured data in **many** different **fields**, such as finance, economics, ecology, neuroscience, and physics.
- Can be of **fixed frequency** or **irregular**.
- **Types**:
  - **Timestamps**, specific instants in time
  - **Fixed periods**, such as the month January 2007 or the full year 2010
  - **Intervals of time**, indicated by a start and end timestamp.
  - Experiment or **elapsed time**

Outline

11.1 Date and Time Data Types and Tools

- Converting Between String and Datetime

11.2 Time Series Basics

- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting

- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

11.1 Date and Time Data Types and Tools

- The Python **datetime**, **time**, and **calendar** modules support date and time data, as well as calendar-related functionality.
- **datetime** stores both the date and time down to the microsecond.
- **timedelta** represents the temporal difference between two datetime objects.

```
from datetime import datetime
now = datetime.now()
now
datetime.datetime(2020, 12, 16, 14,
                  5, 52, 72973)
now.year, now.month, now.day
(2020, 12, 16)
delta = datetime(2020, 12, 31, 1) -
            datetime(2020, 1, 1)
delta
datetime.timedelta(365, 3600)
delta.days, delta.seconds
(365, 3600)
```

11.1 Date and Time Data Types and Tools

- You can **add** (or **subtract**) **timedeltas** to a datetime object to yield a new shifted object.

```
from datetime import timedelta
start = datetime(2020, 12, 16)
start + timedelta(10)
datetime.datetime(2020, 12, 26,
                  0, 0)
start - 2 * timedelta(10)
datetime.datetime(2020, 11, 26,
                  0, 0)
```

Table 11-1. Types in datetime module

Type	Description
`date`	Store calendar date (year, month, day) using the Gregorian calendar
`time`	Store time of day as hours, minutes, seconds, and microseconds
`datetime`	Stores both date and time
`timedelta`	Represents the difference between two `datetime` values (as days, seconds, and microseconds)
`tzinfo`	Base type for storing time zone information

Converting Between String and Datetime

- You can format **datetime** objects and pandas **Timestamp** objects as strings using **str** or the **strftime** method.

- You can use the same format codes to convert strings to dates using **datetime.strptime**.

```
stamp = datetime(2020, 1, 3)
str(stamp)
'2020-01-03 00:00:00'
stamp.strftime('%Y-%m-%d')
'2020-01-03'
```

```
value = '2020-01-03'
datetime.strptime(value,
                  '%Y-%m-%d')
datetime.datetime(2011, 1, 3, 0, 0)
```

Datetime format specification (ISO C89 compatible)

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month [01, 12]
%d	Two-digit day [01, 31]
%H	Hour (24-hour clock) [00, 23]
%I	Hour (12-hour clock) [01, 12]
%M	Two-digit minute [00, 59]
%S	Second [00, 61] (seconds 60, 61 account for leap seconds)
%w	Weekday as integer [0 (Sunday), 6]

Datetime format specification (ISO C89 compatible) – cont.

Type	Description
%U	Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are "week 0"
%W	Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are "week 0"
%z	UTC time zone offset as +HHMM or -HHMM; empty if time zone naive
%F	Shortcut for %Y-%m-%d (e.g., 2012-4-18)
%D	Shortcut for %m/%d/%y (e.g., 04/18/12)

Converting Between String and Datetime

- The **parser.parse** method can parse dates of **common** date formats.

- For our date format, you can pass **dayfirst=True**.

```
from dateutil.parser import parse
parse('2020-01-03')
datetime.datetime(2020, 1, 3, 0, 0)
parse('Jan 31, 1997 10:45 PM')
datetime.datetime(1997, 1, 31,
                  22, 45)
parse('6/12/2020', dayfirst=True)
datetime.datetime(2020, 12, 6,
                  0, 0)
```

Converting Between String and Datetime

- The pandas **to_datetime** method parses many date representations including standard date formats quickly.
- It also handles **missing values**.
- pandas stores timestamps using NumPy's **datetime64** data type at the nanosecond resolution.

```
datestrs = ['2011-07-06 12:00:00',
            '2011-08-06 00:00:00']
idx = pd.to_datetime(datestrs +
                     [None])
idx
```

```
DatetimeIndex(['2011-07-06
12:00:00', '2011-08-06 00:00:00',
'NaT'], dtype='datetime64[ns]',
freq=None)
```

Outline

11.1 Date and Time Data Types and Tools

- Converting Between String and Datetime

11.2 Time Series Basics

- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting

- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

11.2 Time Series Basics

- pandas **Series** can be **indexed** by timestamps.
- The datetime objects is put in a **DatetimeIndex**.

- pandas stores scalar values of datetime objects as **Timestamp** objects.

```
ts.index[0]
```

```
Timestamp('2011-01-02 00:00:00')
```

```
dates = [datetime(2011, 1, 2),
         datetime(2011, 1, 5),
         datetime(2011, 1, 7)]
ts = pd.Series(np.random.randn(3),
               index=dates)
ts
```

```
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
```

```
ts.index
```

```
DatetimeIndex(['2011-01-02', '2011-
01-05', '2011-01-07'],
dtype='datetime64[ns]', freq=None)
```

Indexing, Selection, Subsetting

- Time series behaves like any other pandas.Series when you are **indexing** and **selecting** data based on **label**.

- You can also pass a **string** that is interpretable as a date.

- Slicing:

Equivalent to:

```
ts['1/5/2011':'1/7/2011']
```

```
stamp = ts.index[2]
ts[stamp]
-0.5194387150567381

ts['1/7/2011']
-0.5194387150567381
ts['20110107']
-0.5194387150567381
ts[datetime(2011, 1, 5):]
2011-01-05   0.478943
2011-01-07  -0.519439
```

Indexing, Selection, Subsetting

- For long time series, **select slices** of data by passing a **year** or only a **year and month**.

- The **truncate** method slices a Series between two dates.

```
len(longer_ts.truncate(
                    after='12/31/2020'))
```

```
366
```

```
longer_ts = pd.Series(
     np.random.randn(1000),
     index=pd.date_range(
          '1/1/2020', periods=1000))
len(longer_ts)
```

```
1000
```

```
len(longer_ts['2021'])
```

```
365
```

```
len(longer_ts['2021-5'])
```

```
31
```

Indexing, Selection, Subsetting

- **DataFrames** can be indexed by time series as well.

```
dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')
long_df = pd.DataFrame(np.random.randn(100, 4), index=dates,
                       columns=['Colorado', 'Texas', 'New York', 'Ohio'])
long_df.loc['5-2001']
```

	Colorado	Texas	New York	Ohio
2001-05-02	0.981586	0.056159	-1.787511	0.348340
2001-05-09	0.758415	-1.591688	1.103367	0.945877
2001-05-16	-1.797193	-1.660692	-0.415911	0.521095
2001-05-23	0.675365	-0.169244	0.174330	-0.448221
2001-05-30	-0.699178	0.892690	1.535503	0.350494

Time Series with Duplicate Indices

- Duplicates in time series index are **allowed**.

```
dates = pd.DatetimeIndex(
    ['1/1/2000', '1/2/2000',
     '1/2/2000', '1/2/2000',
     '1/3/2000'])
dup_ts = pd.Series(np.arange(5),
                   index=dates)
dup_ts
2000-01-01  0
2000-01-02  1
2000-01-02  2
2000-01-02  3
2000-01-03  4
```

```
dup_ts.index.is_unique
False
dup_ts['1/3/2000'] # not duplicated
4
dup_ts['1/2/2000'] # duplicated
2000-01-02  1
2000-01-02  2
2000-01-02  3

grouped = dup_ts.groupby(level=0)
grouped.count()
2000-01-01  1
2000-01-02  3
2000-01-03  1
```

Outline

11.1 Date and Time Data Types and Tools
- Converting Between String and Datetime

11.2 Time Series Basics
- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting
- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

11.3 Date Ranges, Frequencies, and Shifting

- pandas has a full suite of standard time series frequencies and tools for:
  - **generating** fixed-frequency date **ranges**.
  - **inferring frequencies**,
  - **resampling**.

Generating Date Ranges

- pandas **date_range** can generate time series index by specifying:
  - Start and end
  - Star or end and periods
- The **freq** option allows you to specify wide range of frequencies:
  - H, min, S
  - D, B, M, BM, MS, BMS
  - W-SUN, W-MON, …
  - AS-JAN, AS-FEB, …

```
pd.date_range('2020-12-16',
              '2020-12-19')
DatetimeIndex(['2020-12-16',
 '2020-12-17', '2020-12-18'
 '2020-12-19'],
 dtype='datetime64[ns]', freq='D')
pd.date_range(start='2012-04-01',
              periods=20)
pd.date_range(end='2012-06-01',
              periods=20)
pd.date_range('2020-01-01',
              '2020-12-01', freq='MS')
```

Frequencies and Date Offsets

- Frequencies in pandas are composed of a **base frequency** and a **multiplier**.

```
from pandas.tseries.offsets import
                Hour, Minute
pd.date_range('2000-01-01',
  '2000-01-03 23:59', freq='4h')


Hour(2) + Minute(30)
<150 * Minutes>
pd.date_range('2000-01-01',
      periods=10, freq='1h30min')
```

Shifting (Leading and Lagging) Data

- **Shifting** refers to moving data backward and forward through time. Both Series and DataFrame have a shift method that leaves the **index unmodified**.

```
ts = pd.Series(np.random.randn(4),
   index=pd.date_range('1/1/2020',
   periods=4, freq='M'))
ts
2020-01-31    0.137923
2020-02-29    0.887474
2020-03-31   -0.090994
2020-04-30    0.412466
```

```
ts.shift(2)
2020-01-31         NaN
2020-02-29         NaN
2020-03-31    0.137923
2020-04-30    0.887474
```

```
ts.shift(-2)
2020-01-31   -0.090994
2020-02-29    0.412466
2020-03-31         NaN
2020-04-30         NaN
```

Shifting (Leading and Lagging) Data

- If the **frequency** is **known**, it can be passed to shift to **advance** the **timestamps** instead of advancing the data.

```
ts.shift(2, freq='M')
2000-03-31 -0.066748
2000-04-30  0.838639
2000-05-31 -0.117388
2000-06-30 -0.517795
```

```
ts.shift(3, freq='D')
```

```
ts.shift(1, freq='90T')
```

Outline

11.1 Date and Time Data Types and Tools

- Converting Between String and Datetime

11.2 Time Series Basics

- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting

- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data