How to use Dates & Times with pandas

Date & Time Series Functionality

- At the root: data types for date & time information
  - Objects for points in time and periods
  - Attributes & methods reflect time-related details
- Sequences of dates & periods:
  - Series or DataFrame columns
  - Index: convert object into Time Series
- Many Series/DataFrame methods rely on time information in the index to provide time-series functionality

Basic Building Block: pd.Timestamp

```
In [1]: import pandas as pd  # assumed imported going forward

In [2]: from datetime import datetime  # To manually create dates

In [3]: time_stamp = pd.Timestamp(datetime(2017, 1, 1))

In [4]: In [13]: pd.Timestamp('2017-01-01') == time_stamp
Out[4]: True # Understands dates as strings

In [5]: time_stamp # type: pandas.tslib.Timestamp
Out[5]: Timestamp('2017-01-01 00:00:00’)

In [6]: time_stamp.year
Out[6]: 2017

In [7]: time_stamp.weekday_name
Out[7]: 'Sunday'
```

Timestamp object has many attributes to store time-specific information

More building blocks: `pd.Period` & `freq`

```
In [8]: period = pd.Period('2017-01')

In [9]: period # default: month-end
Out[9]: Period('2017-01', 'M')

In [10]: period.asfreq('D') # convert to daily
Out[10]: Period('2017-01-31', 'D')

In [11]: period.to_timestamp().to_period('M')
Out[11]: Period('2017-01', 'M')

In [12]: period + 2
Out[12]: Period('2017-03', 'M')

In [13]: pd.Timestamp('2017-01-31', 'M') + 1
Out[13]: Timestamp('2017-02-28 00:00:00', freq='M')
```

`Period` object has `freq` attribute to store frequency info

Convert `pd.Period()` to `pd.Timestamp()` and back

Frequency info enables basic date arithmetic

Sequences of Dates & Times

```
In [14]: index = pd.date_range(start='2017-1-1', periods=12, freq='M')
```

pd.date_range: start, end, periods, freq

```
In [15]: index
DatetimeIndex(['2017-01-31', '2017-02-28', '2017-03-31', …,
               '2017-09-30', '2017-10-31', '2017-11-30', '2017-12-31'],
              dtype='datetime64[ns]', freq='M')

In [16]: index[0]
Timestamp('2017-01-31 00:00:00', freq='M')
```

pd.DateTimeIndex: sequence of Timestamp objects with frequency info

```
In [17]: index.to_period()
PeriodIndex(['2017-01', '2017-02', '2017-03', '2017-04', …,
             '2017-11', '2017-12'], dtype='period[M]', freq='M')
```

Create a Time Series: pd.DateTimeIndex

```
In [14]: pd.DataFrame({'data': index}).info()

RangeIndex: 12 entries, 0 to 11
Data columns (total 1 columns):
data     12 non-null datetime64[ns]
dtypes: datetime64[ns](1)

In [15]: data = np.random.random((size=12,2))

In [16]: pd.DataFrame(data=data, index=index).info()

DatetimeIndex: 12 entries, 2017-01-31 to 2017-12-31
Freq: M
Data columns (total 2 columns):
0     12 non-null float64
1     12 non-null float64
dtypes: float64(2)
```

np.random.random:
Random numbers [0,1]
12 rows, 2 columns

Frequency Aliases & Time Info

There are many frequency aliases besides 'M' and 'D':

Period	Alias
Hour	H
Day	D
Week	W
Month	M
Quarter	Q
Year	A

These may be further differentiated by beginning/end of period, or business-specific definition

You can also access these `pd.Timestamp()` attributes:

attribute
`.second, .minute, .hour,`
`.day, .month, .quarter, .year`
`.weekday`
`dayofweek`
`.weekofyear`
`.dayofyear`

Let’s practice!

Indexing & Resampling Time Series

Time Series Transformation

Basic Time Series transformations include:

- Parsing string dates and convert to `datetime64`
- Selecting & slicing for specific subperiods
- Setting & changing `DateTimeIndex` frequency
    - Upsampling vs Downsampling

Getting GOOG stock prices

```
In [1]: google = pd.read_csv('google.csv')  # import pandas as pd

In [2]: google.info()
<class ‘pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 2 columns):
date     504 non-null object
price    504 non-null float64
dtypes: float64(1), object(1)

In [3]: google.head()

         date   price
0  2015-01-02  524.81
1  2015-01-05  513.87
2  2015-01-06  501.96
3  2015-01-07  501.10
4  2015-01-08  502.68
```

Converting string dates to datetime64

```
In [4]: google.date = pd.to_datetime(google.date)

In [5]: google.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 2 columns):
date      504 non-null datetime64[ns]
price     504 non-null float64
dtypes: datetime64[ns](1), float64(1)

In [6]: google.set_index('date', inplace=True)

In [7]: google.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 504 entries, 2015-01-02 to 2016-12-30
Data columns (total 1 columns):
price     504 non-null float64
dtypes: float64(1)
```

`pd.to_datetime()`:

- Parse date string
- Convert to `datetime64`

`.set_index()`:

- Date into `index`

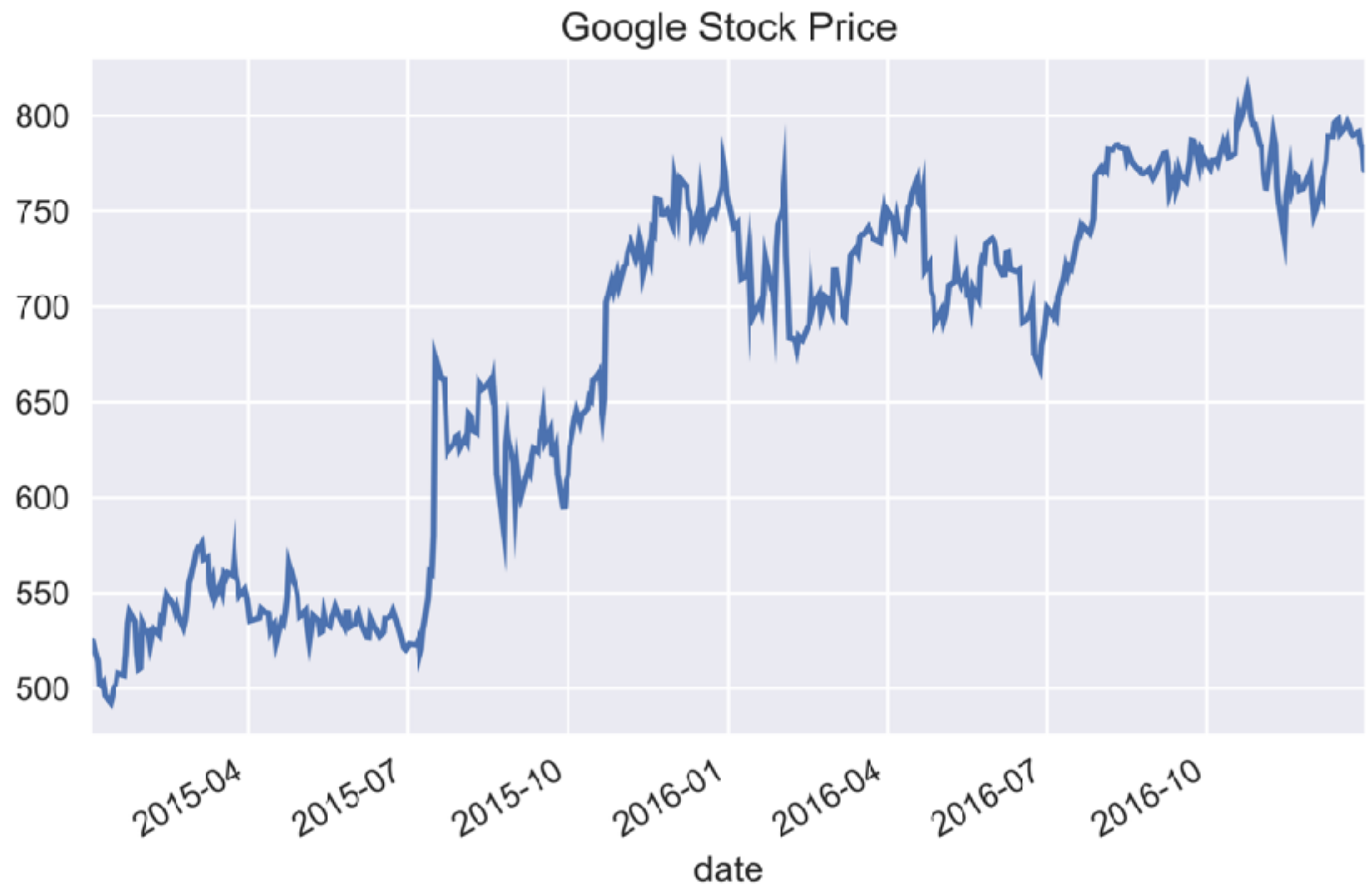
`inplace`:

- don't create copy

Plotting the Google Stock Time Series

```
In [8]: google.price.plot(title='Google Stock Price')

In [9]: plt.tight_layout(); plt.show()
```



Partial String Indexing

```
In [10]: google['2015'].info() # Pass string for part of date

DatetimeIndex: 252 entries, 2015-01-02 to 2015-12-31
Data columns (total 1 columns):
price     252 non-null float64
dtypes: float64(1)
```

Selecting/indexing using `strings` that parse to dates

```
In [11]: google['2015-3': '2016-2'].info() # Slice includes last month

DatetimeIndex: 252 entries, 2015-03-02 to 2016-02-29
Data columns (total 1 columns):
price     252 non-null float64
dtypes: float64(1)
memory usage: 3.9 KB

In [12]: google.loc['2016-6-1', 'price'] # Use full date with .loc[]
Out[12]: 734.15
```

.asfreq(): Set Frequency

```
In [13]: google.asfreq('D').info() # set calendar day frequency

DatetimeIndex: 729 entries, 2015-01-02 to 2016-12-30
Freq: D
Data columns (total 1 columns):
price     504 non-null float64
dtypes: float64(1)

In [14]: google.asfreq('D').head()
Out[14]:
             price
date
2015-01-02  524.81
2015-01-03     NaN
2015-01-04     NaN
2015-01-05  513.87
2015-01-06  501.96
```

.asfreq('D'):
Convert DateTimeIndex to calendar day frequency

Upsampling:
Higher frequency implies new dates => missing data

.asfreq(): Reset Frequency

```
In [18]: google = google.asfreq('B') # Change to calendar day frequency

In [19]: google.info()

DatetimeIndex: 521 entries, 2015-01-02 to 2016-12-30
Freq: B
Data columns (total 1 columns):
price     504 non-null float64
dtypes: float64(1)

In [20]: google[google.price.isnull()] # Select missing 'price' values
Out[20]:
             price
date
2015-01-19     NaN
2015-02-16     NaN
..
2016-11-24     NaN
2016-12-26     NaN
```

.asfreq('B'): Convert DateTimeIndex to business day frequency

Business days that were not trading days

Let’s practice!

Lags, changes, and returns for Stock Price Series

Basic Time Series Calculations

- Typical Time Series manipulations include:
  - Shift or lag values back or forward back in time
  - Get the difference in value for a given time period
  - Compute the percent change over any number of periods
- `pandas` built-in methods rely on `pd.DateTimeIndex`

Getting GOOG stock prices

```
In [1]: google = pd.read_csv('google.csv',
                             parse_dates=['date'],
                             index_col='date')

In [2]: google.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 504 entries, 2015-01-02 to 2016-12-30
Data columns (total 1 columns):
price    504 non-null float64
dtypes: float64(1)

In [3]: google.head()
             price
date
2015-01-02  524.81
2015-01-05  513.87
2015-01-06  501.96
2015-01-07  501.10
2015-01-08  502.68
```

Let `pd.read_csv()` do the parsing for you!

.shift(): Moving data between past & future

```
In [4]: google['shifted'] = google.price.shift() # default: periods=1

In [5]: google.head(3)
Out[5]:
             price  shifted
date
2015-01-02  542.81      NaN
2015-01-05  513.87   542.81
2015-01-06  501.96   513.87

In [6]: google['lagged'] = google.price.shift(periods=-1)

In [7]: google[['price', 'lagged', 'shifted']].tail(3)
Out[7]:
             price  lagged  shifted
date
2016-12-28  785.05  782.79   791.55
2016-12-29  782.79  771.82   785.05
2016-12-30  771.82     NaN   782.79
```

.shift():
defaults to periods=1
1 period into future

.shift(periods=-1):
lagged data:
1 period back in time

Calculate one-period percent change

```
In [10]: google['change'] = google.price.div(google.shifted) # xt / xt-1

In [11]: google[['price', 'shifted', 'change']].head(3)
Out[11]:
             price  shifted    change
Date
2017-01-03  786.14      NaN       NaN
2017-01-04  786.90   786.14  1.000967
2017-01-05  794.02   786.90  1.009048

In [12]: google['return'] = google.change.sub(1).mul(100)

In [13]: google[['price', 'shifted', 'change', 'return']].head(3)
Out[13]:
             price  shifted  change  return
date
2015-01-02  524.81      NaN     NaN     NaN
2015-01-05  513.87   524.81    0.98   -2.08
2015-01-06  501.96   513.87    0.98   -2.32
```

.diff() & .pct_change(): built-in time-series change

```
In [14]: google['diff'] = google.price.diff() # xt - xt-1

In [15]: google[['price', 'diff']].head(3)
Out[15]:
             price      diff
date
2015-01-02  524.81       NaN
2015-01-05  513.87    -10.94
2015-01-06  501.96    -11.91
```

Difference in value for two adjacent periods

```
In [16]: google['pct_change'] = google.price.pct_change().mul(100)

In [17]: google[['price', 'return', 'pct_change']].head(3)
Out[17]:
             price   return  pct_change
date
2015-01-02  524.81      NaN         NaN
2015-01-05  513.87    -2.08       -2.08
2015-01-06  501.96    -2.32       -2.32
```

Percent change for two adjacent periods

Looking ahead: Get Multi-period returns

```
In [25]: google['return_3d'] = google.price.pct_change(3).mul(100)

In [34]: google[['price', 'return_3d']].head()
Out[34]:
             price  return_3d
date
2015-01-02  524.81        NaN
2015-01-05  513.87        NaN
2015-01-06  501.96        NaN
2015-01-07  501.10  -4.517825
2015-01-08  502.68  -2.177594
```

Percent change for two periods, 3 trading days apart

Let’s practice!