PANDAS FOUNDATIONS

pandas Foundations

What is pandas?

- Python library for data analysis
- High-performance containers for data analysis
- Data structures with a lot of functionality
  - Meaningful labels
  - Time series functionality
  - Handling missing data
  - Relational operations

What you will learn

- How to work with pandas
  - Data import & export in various formats
- Exploratory Data Analysis using pandas
  - Statistical & graphical methods
- Using pandas to model *time series*
  - Time indexes, resampling

PANDAS FOUNDATIONS

See you in the course!

PANDAS FOUNDATIONS

Review of pandas DataFrames

Pandas is a library for data analysis.

the powertool of Pd is the dataframe, a tabular data structure with labelled rows and columns.

pandas DataFrames

Indexes in Pd are tailored lists of labels that permit fast look-up and some powerful relational operations.

- Example: DataFrame of Apple Stock data

Date	Open	High	Low	Close	Volume	Adj Close
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
...	...	...	...	...	...	...

Indexes and columns

```
In [1]: import pandas as pd

In [2]: type(AAPL)
Out[2]: pandas.core.frame.DataFrame

In [3]: AAPL.shape
Out[3]: (8514, 6)

In [4]: AAPL.columns
Out[4]:
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'],
dtype='object')

In [5]: type(AAPL.columns)
Out[5]: pandas.indexes.base.Index
```

Indexes and columns

```
In [6]: AAPL.index
Out[6]:
DatetimeIndex(['2014-09-16', '2014-09-15', '2014-09-12',
               '2014-09-11', '2014-09-10', '2014-09-09',
               '2014-09-08', '2014-09-05', '2014-09-04',
               '2014-09-03',
               ...
               '1980-12-26', ‘1980-12-24', '1980-12-23',
               '1980-12-22', '1980-12-19', '1980-12-18',
               '1980-12-17', '1980-12-16', '1980-12-15',
               '1980-12-12'],
              dtype='datetime64[ns]', name='Date', length=8514,
              freq=None)

In [7]: type(AAPL.index)
Out[7]: pandas.tseries.index.DatetimeIndex
```

Slicing

```
In [8]: AAPL.iloc[:5,:]
Out[8]:
              Open    High     Low   Close     Volume  Adj Close
Date
2014-09-16   99.80  101.26   98.89  100.86   66818200     100.86
2014-09-15  102.81  103.05  101.44  101.63   61216500     101.63
2014-09-12  101.21  102.19  101.08  101.66   62626100     101.66
2014-09-11  100.41  101.44   99.62  101.43   62353100     101.43
2014-09-10   98.01  101.11   97.76  101.00  100741900     101.00

In [9]: AAPL.iloc[-5:,:]
Out[9]:
             Open   High    Low  Close     Volume  Adj Close
Date
1980-12-18  26.63  26.75  26.63  26.63   18362400       0.41
1980-12-17  25.87  26.00  25.87  25.87   21610400       0.40
1980-12-16  25.37  25.37  25.25  25.25   26432000       0.39
1980-12-15  27.38  27.38  27.25  27.25   43971200       0.42
1980-12-12  28.75  28.87  28.75  28.75  117258400       0.45
```

head()

```
In [10]: AAPL.head(5)
Out[10]:
               Open    High      Low   Close      Volume  Adj Close
Date
2014-09-16    99.80  101.26    98.89  100.86    66818200      100.86
2014-09-15   102.81  103.05   101.44  101.63    61216500      101.63
2014-09-12   101.21  102.19   101.08  101.66    62626100      101.66
2014-09-11   100.41  101.44    99.62  101.43    62353100      101.43
2014-09-10    98.01  101.11    97.76  101.00   100741900      101.00

In [11]: AAPL.head(2)
Out[11]:
               Open    High      Low   Close     Volume  Adj Close
Date
2014-09-16    99.80  101.26    98.89  100.86   66818200      100.86
2014-09-15   102.81  103.05   101.44  101.63   61216500      101.63
```

tail()

```
In [12]: AAPL.tail()
Out[12]:
              Open   High    Low  Close     Volume  Adj Close
Date
1980-12-18  26.63  26.75  26.63  26.63   18362400       0.41
1980-12-17  25.87  26.00  25.87  25.87   21610400       0.40
1980-12-16  25.37  25.37  25.25  25.25   26432000       0.39
1980-12-15  27.38  27.38  27.25  27.25   43971200       0.42
1980-12-12  28.75  28.87  28.75  28.75  117258400       0.45

In [13]: AAPL.tail(3)
Out[13]:
              Open   High    Low  Close     Volume  Adj Close
Date
1980-12-16  25.37  25.37  25.25  25.25   26432000       0.39
1980-12-15  27.38  27.38  27.25  27.25   43971200       0.42
1980-12-12  28.75  28.87  28.75  28.75  117258400       0.45
```

info()

```
In [14]: AAPL.info()
Out[14]:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8514 entries, 2014-09-16 to 1980-12-12
Data columns (total 6 columns):
Open         8514 non-null float64
High         8514 non-null float64
Low          8514 non-null float64
Close        8514 non-null float64
Volume       8514 non-null int64
Adj Close    8514 non-null float64
dtypes: float64(5), int64(1)
memory usage: 465.6 KB
```

Broadcasting

```
In [15]: import numpy as np

In [16]: AAPL.iloc[::3, -1] = np.nan

In [17]: AAPL.head(6)
Out[17]:
              Open    High     Low   Close     Volume  Adj Close
Date
2014-09-16   99.80  101.26   98.89  100.86   66818200        NaN
2014-09-15  102.81  103.05  101.44  101.63   61216500     101.63
2014-09-12  101.21  102.19  101.08  101.66   62626100     101.66
2014-09-11  100.41  101.44   99.62  101.43   62353100        NaN
2014-09-10   98.01  101.11   97.76  101.00  100741900     101.00
2014-09-09   99.08  103.08   96.14   97.99  189560600      97.99
2014-09-08   99.30   99.31   98.05   98.36   46277800        NaN
```

Assigning scalar value to column slice *broadcasts* value to each row.

Broadcasting

```
In [18]: AAPL.info()
Out[18]:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8514 entries, 2014-09-16 to 1980-12-12
Data columns (total 6 columns):
Open          8514 non-null float64
High          8514 non-null float64
Low           8514 non-null float64
Close         8514 non-null float64
Volume        8514 non-null int64
Adj Close     5676 non-null float64
dtypes: float64(5), int64(1)
memory usage: 465.6 KB
```

Series

```
In [19]: low = AAPL['Low']

In [20]: type(low)
Out[20]: pandas.core.series.Series

In [21]: low.head()
Out[21]:
Date
2014-09-16     98.89
2014-09-15    101.44
2014-09-12    101.08
2014-09-11     99.62
2014-09-10     97.76
Name: Low, dtype: float64

In [22]: lows = low.values

In [23]: type(lows)
Out[23]: numpy.ndarray
```

the data in the Series actually form a numpy array which is what the valus attribute yeilds.

a Pd series is a 1D labelled numpy array and a dataframe is a 2D labelled array whose columns are series.

PANDAS FOUNDATIONS

Let’s practice!

PANDAS FOUNDATIONS

Building DataFrames from scratch

DataFrames from CSV files

```
In [1]: import pandas as pd

In [2]: users = pd.read_csv('datasets/users.csv', index_col=0)

In [3]: print(users)
Out[3]:
  weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

DataFrames from dict (1)

```
In [1]: import pandas as pd

In [2]: data = {'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],
   ...:         'city': ['Austin', 'Dallas', 'Austin', 'Dallas',
   ...:         'visitors': [139, 237, 326, 456],
   ...:         'signups': [7, 12, 3, 5]}

In [3]: users = pd.DataFrame(data)

In [4]: print(users)
Out[4]:
  weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

with no index specified, the row labels are integers zero to three by default.

DataFrames from dict (2)

```
In [1]: import pandas as pd

In [2]: cities = ['Austin', 'Dallas', 'Austin', 'Dallas']

In [3]: signups = [7, 12, 3, 5]

In [4]: visitors = [139, 237, 326, 456]

In [5]: weekdays = ['Sun', 'Sun', 'Mon', 'Mon']

In [6]: list_labels = ['city', 'signups', 'visitors', 'weekday']

In [7]: list_cols = [cities, signups, visitors, weekdays]

In [8]: zipped = list(zip(list_labels, list_cols))
```

DataFrames from dict (3)

```
In [9]: print(zipped)
Out[9]:
[('city', ['Austin', 'Dallas', 'Austin', 'Dallas']), ('signups',
[7, 12, 3, 5]), ('visitors', [139, 237, 326, 456]), ('weekday',
['Sun', 'Sun', 'Mon', 'Mon'])]

In [10]: data = dict(zipped)

In [11]: users = pd.DataFrame(data)

In [12]: print(users)
Out[12]:
  weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

Broadcasting

add a new column

```
In [13]: users['fees'] = 0 # Broadcasts to entire column

In [14]: print(users)
Out[14]:
     city  signups  visitors weekday  fees
0  Austin        7       139     Sun     0
1  Dallas       12       237     Sun     0
2  Austin        3       326     Mon     0
3  Dallas        5       456     Mon     0
```

Broadcasting with a dict

```
In [1]: import pandas as pd

In [2]: heights = [ 59.0, 65.2, 62.9, 65.4, 63.7, 65.7, 64.1 ]

In [3]: data = {'height': heights, 'sex': 'M'}
```

value M for the whole column

```
In [4]: results = pd.DataFrame(data)

In [5]: print(results)
Out[5]:
   height sex
0    59.0   M
1    65.2   M
2    62.9   M
3    65.4   M
4    63.7   M
5    65.7   M
6    64.1   M
```

Index and columns

```
In [6]: results.columns = ['height (in)', 'sex']

In [7]: results.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G']

In [8]: print(results)
Out[8]:
   height (in) sex
A          59.0   M
B          65.2   M
C          62.9   M
D          65.4   M
E          63.7   M
F          65.7   M
G          64.1   M
```

PANDAS FOUNDATIONS

Let’s practice!

PANDAS FOUNDATIONS

Importing & exporting data

Original CSV file

Sunspot Index & Long-term Solar Observations

- Dataset: Sunspot observations collected from SILSO

```
1818,01,01,1818.004, -1,1
1818,01,02,1818.007, -1,1
1818,01,03,1818.010, -1,1
1818,01,04,1818.012, -1,1
1818,01,05,1818.015, -1,1
1818,01,06,1818.018, -1,1
                ...
```

Source: SILSO, Daily total sunspot number (http://www.sidc.be/silso/infossntotdaily)

Datasets from CSV files

```
In [1]: import pandas as pd

In [2]: filepath = 'ISSN_D_tot.csv'

In [3]: sunspots = pd.read_csv(filepath)

In [4]: sunspots.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71921 entries, 0 to 71920
Data columns (total 6 columns):
1818        71921 non-null int64
01          71921 non-null int64
01.1        71921 non-null int64
1818.004    71921 non-null float64
 -1         71921 non-null int64
1           71921 non-null int64
dtypes: float64(1), int64(5)
memory usage: 3.3 MB
```

row labels are of type RangeIndex (just integers)

Datasets from CSV files

```
In [5]: sunspots.iloc[10:20, :]
Out[5]:
    1818  01  01.1  1818.004   -1  1
10  1818   1    12  1818.034   -1  1
11  1818   1    13  1818.037   22  1
12  1818   1    14  1818.040   -1  1
13  1818   1    15  1818.042   -1  1
14  1818   1    16  1818.045   -1  1
15  1818   1    17  1818.048   46  1
16  1818   1    18  1818.051   59  1
17  1818   1    19  1818.053   63  1
18  1818   1    20  1818.056   -1  1
19  1818   1    21  1818.059   -1  1
```

Problems

- CSV file has no column headers
  - Columns 0-2: Gregorian date (year, month, day)
  - Column 3: Date as fraction as year
  - Column 4: Daily total sunspot number
  - Column 5: Definitive/provisional indicator (1 or 0)
- Missing values in column 4: indicated by -1
- Dates representation inconvenient

Using header keyword

```
In [6]: sunspots = pd.read_csv(filepath, header=None)

In [7]: sunspots.iloc[10:20, :]
Out[7]:
      0  1   2          3   4  5
10  1818  1  11  1818.031  -1  1
11  1818  1  12  1818.034  -1  1
12  1818  1  13  1818.037  22  1
13  1818  1  14  1818.040  -1  1
14  1818  1  15  1818.042  -1  1
15  1818  1  16  1818.045  -1  1
16  1818  1  17  1818.048  46  1
17  1818  1  18  1818.051  59  1
18  1818  1  19  1818.053  63  1
19  1818  1  20  1818.056  -1  1
```

Using names keyword

```
In [8]: col_names = ['year', 'month', 'day', 'dec_date',
   ...:              'sunspots', 'definite']

In [9]: sunspots = pd.read_csv(filepath, header=None,
   ...:                        names=col_names)

In [10]:  sunspots.iloc[10:20, :]
Out[10]:
    year  month  day  dec_date  sunspots  definite
10  1818      1   11  1818.031        -1         1
11  1818      1   12  1818.034        -1         1
12  1818      1   13  1818.037        22         1
13  1818      1   14  1818.040        -1         1
14  1818      1   15  1818.042        -1         1
15  1818      1   16  1818.045        -1         1
16  1818      1   17  1818.048        46         1
17  1818      1   18  1818.051        59         1
18  1818      1   19  1818.053        63         1
19  1818      1   20  1818.056        -1         1
```

Using na_values keyword (1)

```
In [11]: sunspots = pd.read_csv(filepath, header=None,
    ...:                        names=col_names, na_values='-1')

In [12]: sunspots.iloc[10:20, :]
Out[12]:
    year  month  day  dec_date  sunspots  definite
10  1818      1   11  1818.031        -1         1
11  1818      1   12  1818.034        -1         1
12  1818      1   13  1818.037        22         1
13  1818      1   14  1818.040        -1         1
14  1818      1   15  1818.042        -1         1
15  1818      1   16  1818.045        -1         1
16  1818      1   17  1818.048        46         1
17  1818      1   18  1818.051        59         1
18  1818      1   19  1818.053        63         1
19  1818      1   20  1818.056        -1         1
```

there are space characters preceding minus ones throughout column 4

Using na_values keyword (2)

```
In [13]: sunspots = pd.read_csv(filepath, header=None,
    ...:                        names=col_names, na_values=' -1')

In [14]: sunspots.iloc[10:20, :]
Out[14]:
    year  month  day  dec_date  sunspots  definite
10  1818      1   11  1818.031       NaN         1
11  1818      1   12  1818.034       NaN         1
12  1818      1   13  1818.037      22.0         1
13  1818      1   14  1818.040       NaN         1
14  1818      1   15  1818.042       NaN         1
15  1818      1   16  1818.045       NaN         1
16  1818      1   17  1818.048      46.0         1
17  1818      1   18  1818.051      59.0         1
18  1818      1   19  1818.053      63.0         1
19  1818      1   20  1818.056       NaN         1
```

Using na_values keyword (3)

```
In [15]: sunspots = pd.read_csv(filepath, header=None,
    ...: names=col_names, na_values={'sunspots':[' -1']})

In [16]: sunspots.iloc[10:20, :]
Out[16]:
    year  month  day  dec_date  sunspots  definite
10  1818      1   11  1818.031       NaN         1
11  1818      1   12  1818.034       NaN         1
12  1818      1   13  1818.037      22.0         1
13  1818      1   14  1818.040       NaN         1
14  1818      1   15  1818.042       NaN         1
15  1818      1   16  1818.045       NaN         1
16  1818      1   17  1818.048      46.0         1
17  1818      1   18  1818.051      59.0         1
18  1818      1   19  1818.053      63.0         1
19  1818      1   20  1818.056       NaN         1
```

Using parse_dates keyword

```
In [17]: sunspots = pd.read_csv(filepath, header=None,
    ...: names=col_names, na_values={'sunspots':[' -1']},
    ...: parse_dates=[[0, 1, 2]])

In [18]: sunspots.iloc[10:20, :]
Out[18]:
   year_month_day  dec_date  sunspots  definite
10     1818-01-11  1818.031       NaN         1
11     1818-01-12  1818.034       NaN         1
12     1818-01-13  1818.037      22.0         1
13     1818-01-14  1818.040       NaN         1
14     1818-01-15  1818.042       NaN         1
15     1818-01-16  1818.045       NaN         1
16     1818-01-17  1818.048      46.0         1
17     1818-01-18  1818.051      59.0         1
18     1818-01-19  1818.053      63.0         1
19     1818-01-20  1818.056       NaN         1
```

Inspecting DataFrame

```
In [19]: sunspots.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71922 entries, 0 to 71921
Data columns (total 4 columns):
year_month_day    71922 non-null datetime64[ns]
dec_date          71922 non-null float64
sunspots          68675 non-null float64
definite          71922 non-null int64
dtypes: datetime64[ns](1), float64(2), int64(1)
memory usage: 2.2 MB
```

Using dates as index

```
In [20]: sunspots.index = sunspots['year_month_day']

In [21]: sunspots.index.name = 'date'

In [22]: sunspots.info()
Out[22]:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 71922 entries, 1818-01-01 to 2014-11-30
Data columns (total 4 columns):
year_month_day    71922 non-null datetime64[ns]
dec_date          71922 non-null float64
sunspots          68675 non-null float64
definite          71922 non-null int64
dtypes: datetime64[ns](1), float64(2), int64(1)
memory usage: 2.7 MB
```

Trimming redundant columns

```
In [23]: cols = ['sunspots', 'definite']

In [24]: sunspots = sunspots[cols]

In [25]: sunspots.iloc[10:20, :]
Out[25]:
            sunspots  definite
date
1818-01-11       NaN         1
1818-01-12       NaN         1
1818-01-13      22.0         1
1818-01-14       NaN         1
1818-01-15       NaN         1
1818-01-16       NaN         1
1818-01-17      46.0         1
1818-01-18      59.0         1
1818-01-19      63.0         1
1818-01-20       NaN         1
```

Writing files

```
In [26]: out_csv = 'sunspots.csv'

In [27]: sunspots.to_csv(out_csv)

In [28]: out_tsv = 'sunspots.tsv'

In [29]: sunspots.to_csv(out_tsv, sep='\t')

In [30]: out_xlsx = 'sunspots.xlsx'

In [31]: sunspots.to_excel(out_xlsx)
```

PANDAS FOUNDATIONS

Let’s practice!

PANDAS FOUNDATIONS

Plotting with pandas

AAPL stock data

```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt

In [3]: aapl = pd.read_csv('aapl.csv', index_col='date',
   ...:                    parse_dates=True)
```

to force datetime64 index

```
In [4]: aapl.head(6)
Out[4]:
            adj_close   close    high     low    open     volume
date
2000-03-01      31.68  130.31  132.06  118.50  118.56  38478000
2000-03-02      29.66  122.00  127.94  120.69  127.00  11136800
2000-03-03      31.12  128.00  128.23  120.00  124.87  11565200
2000-03-06      30.56  125.69  129.13  125.00  126.00   7520000
2000-03-07      29.87  122.87  127.44  121.12  126.44   9767600
2000-03-08      29.66  122.00  123.94  118.56  122.87   9690800
```

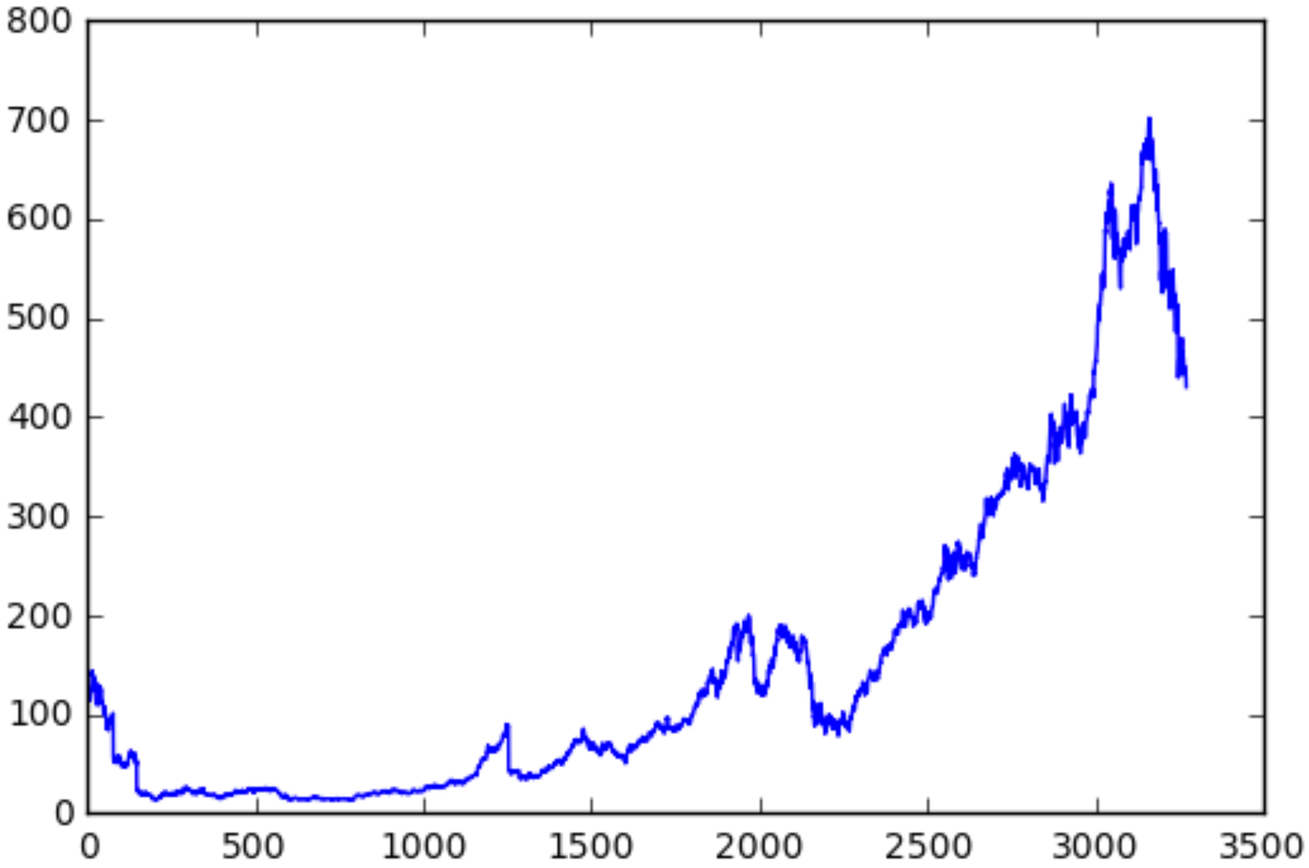
Plotting arrays (matplotlib)

```
In [5]: close_arr = aapl['close'].values
```

(to array)

```
In [6]: type(close_arr)
Out[6]: numpy.ndarray

In [7]: plt.plot(close_arr)
Out[7]: [<matplotlib.lines.Line2D at 0x115550358>]

In [8]: plt.show()
```

Plotting arrays (Matplotlib)

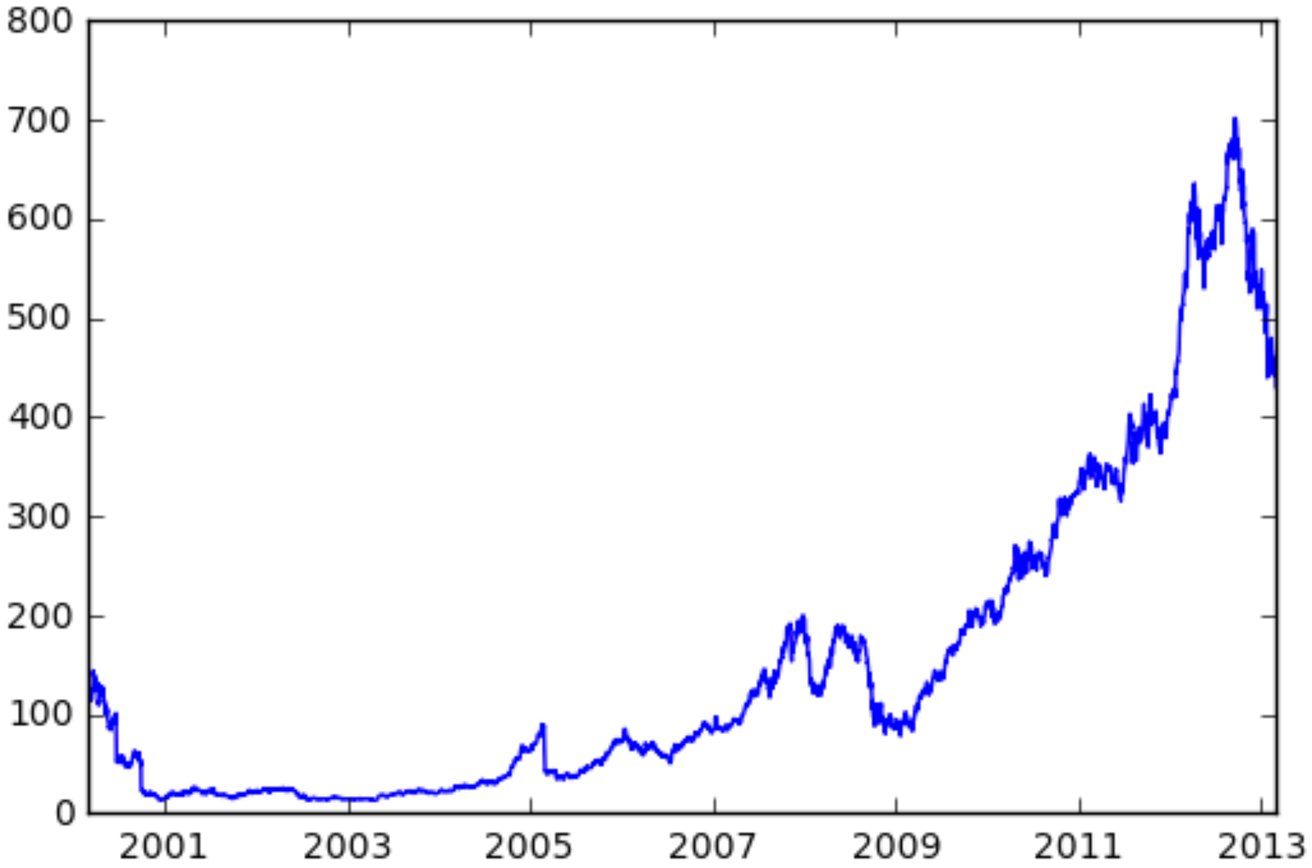


x-axis corresponds to date indices of the array

Plotting Series (matplotlib)

```
In [9]: close_series = aapl['close']

In [10]: type(close_series)
Out[10]: pandas.core.series.Series

In [11]: plt.plot(close_series)
Out[11]: [<matplotlib.lines.Line2D at 0x11801cd30>]

In [12]: plt.show()
```

Plotting Series (matplotlib)



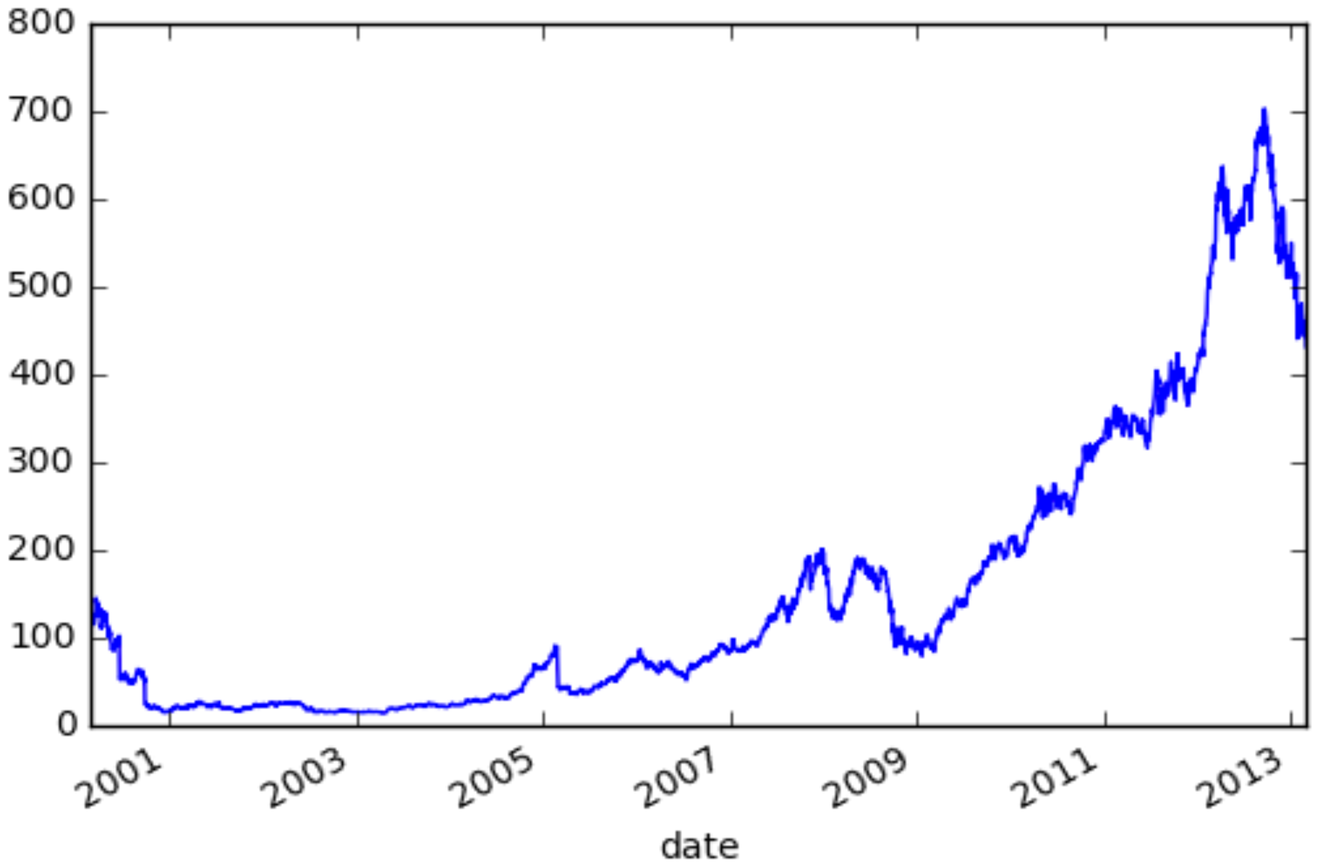
plot function automatically uses the Series's datetime index labels along the horizontal axis

Plotting Series (pandas)

```
In [13]: close_series.plot()  # plots Series directly

In [14]: plt.show()
```

nicer alternative is to use pandas series plot method

Plotting Series (pandas)

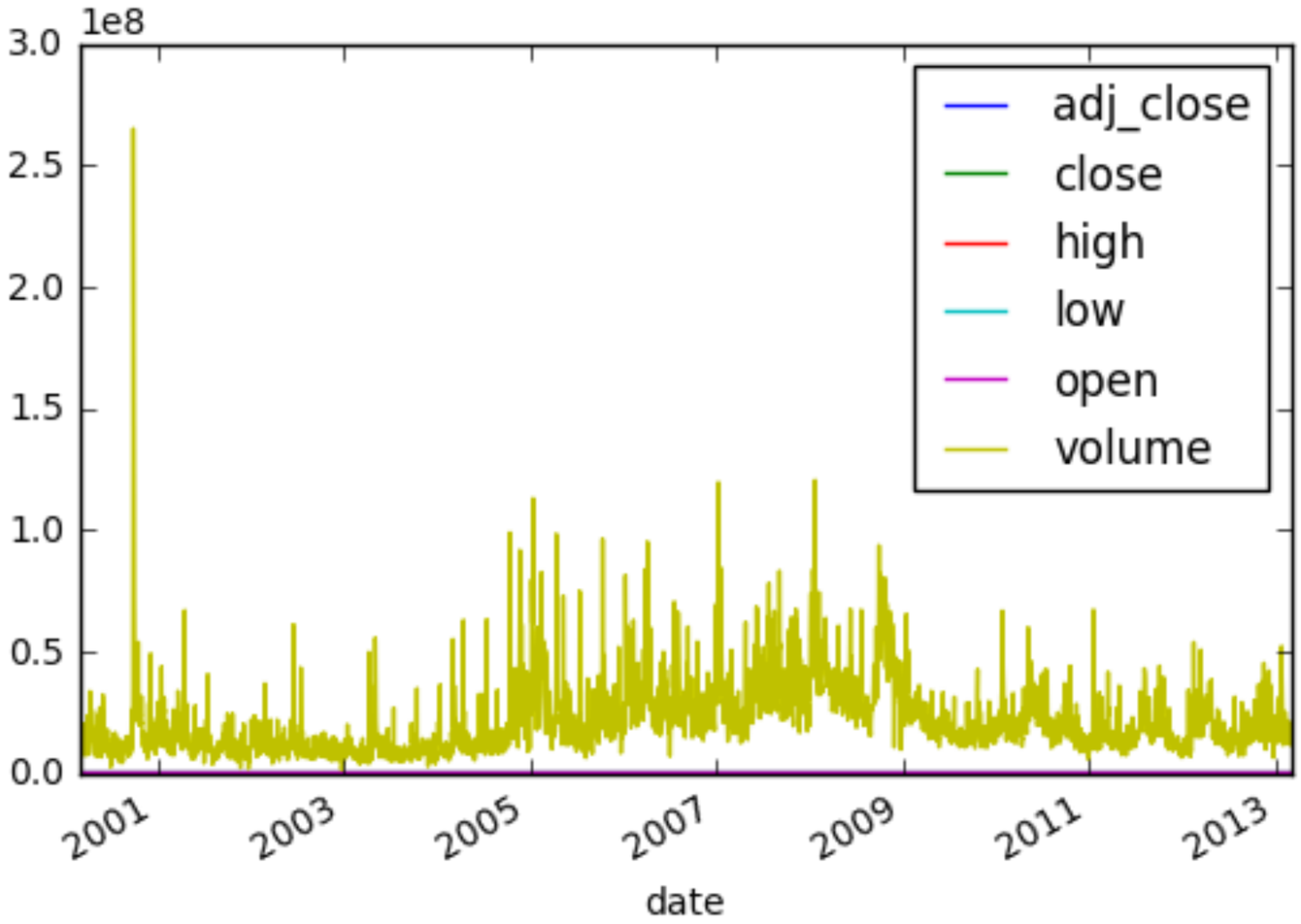


Plotting DataFrames (pandas)

```
In [15]: aapl.plot()  # plots all Series at once
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x118039b38>

In [16]: plt.show()
```

in fact, pandas DataFrames have a plot method just like pandas series

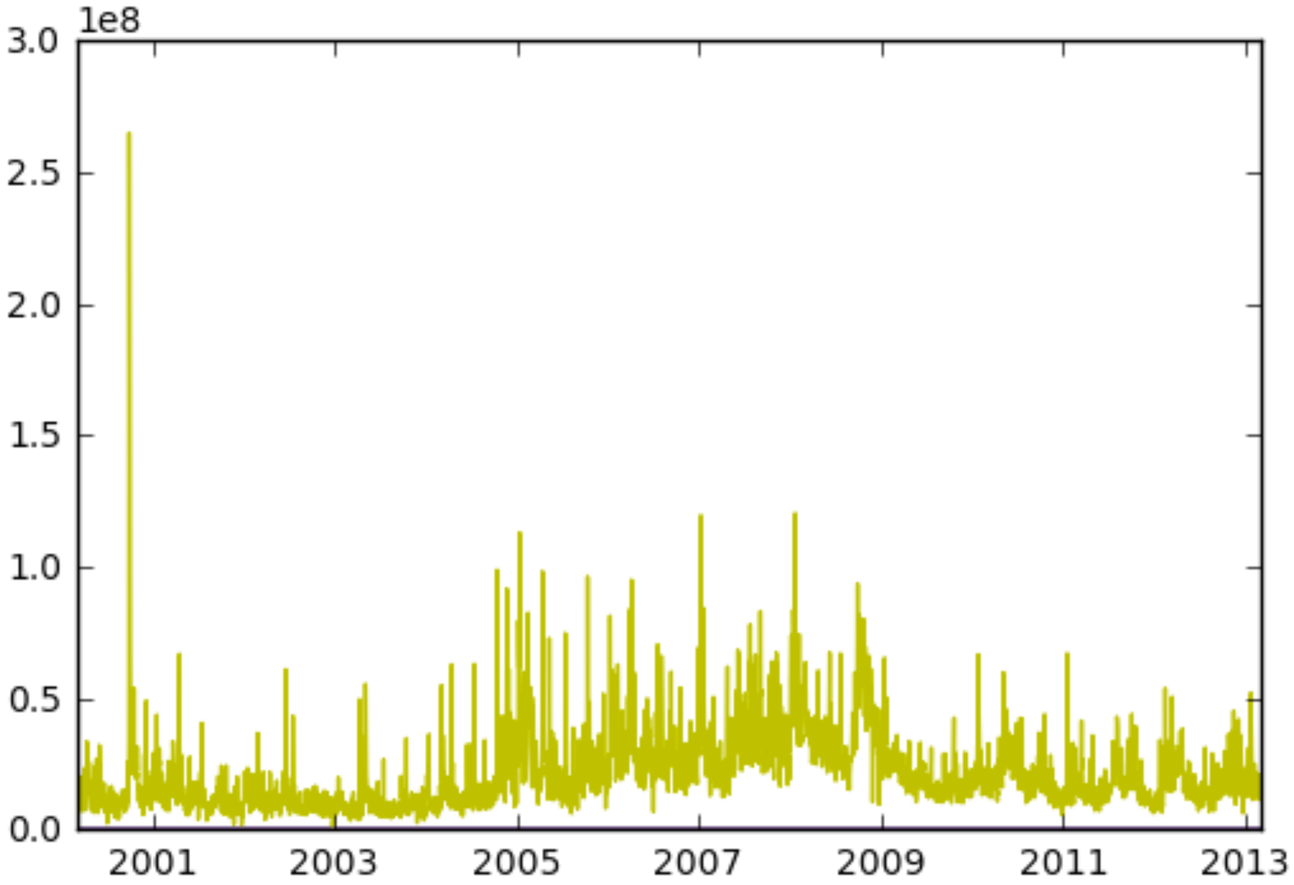
Plotting DataFrames (pandas)



Plotting DataFrames (matplotlib)

```
In [17]: plt.plot(aapl) # plots all columns at once
Out[17]:
 <matplotlib.lines.Line2D at 0x1156290f0>,
 <matplotlib.lines.Line2D at 0x1156525f8>,
 <matplotlib.lines.Line2D at 0x1156527f0>,
 <matplotlib.lines.Line2D at 0x1156529e8>,
 <matplotlib.lines.Line2D at 0x115652be0>,
 <matplotlib.lines.Line2D at 0x115652dd8>

In [18]: plt.show()
```

Plotting DataFrames (matplotlib)

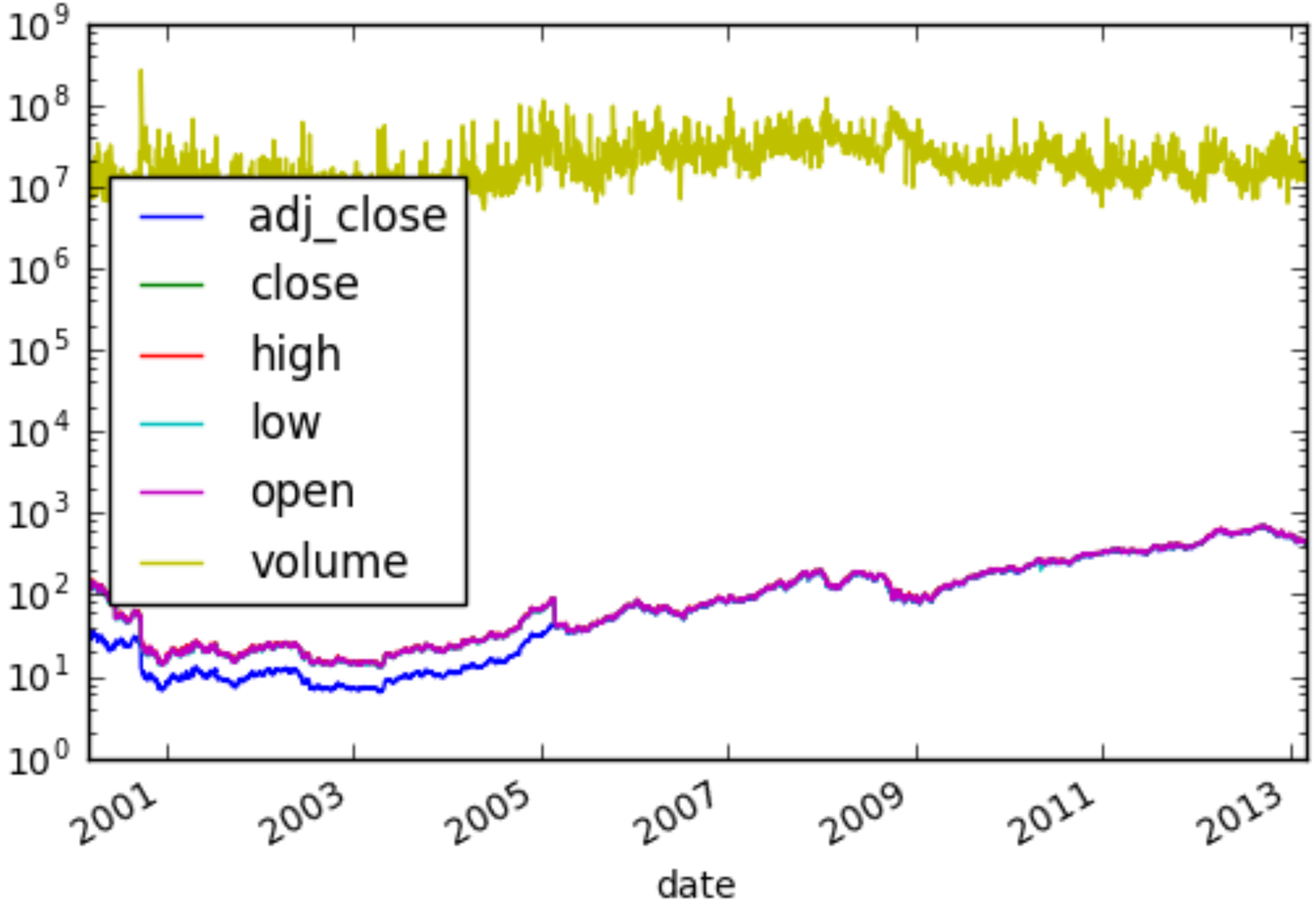


no legend, no title

Fixing scales

```
In [19]: aapl.plot()
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x118afe048>

In [20]: plt.yscale('log') # logarithmic scale on vertical axis

In [21]: plt.show()
```

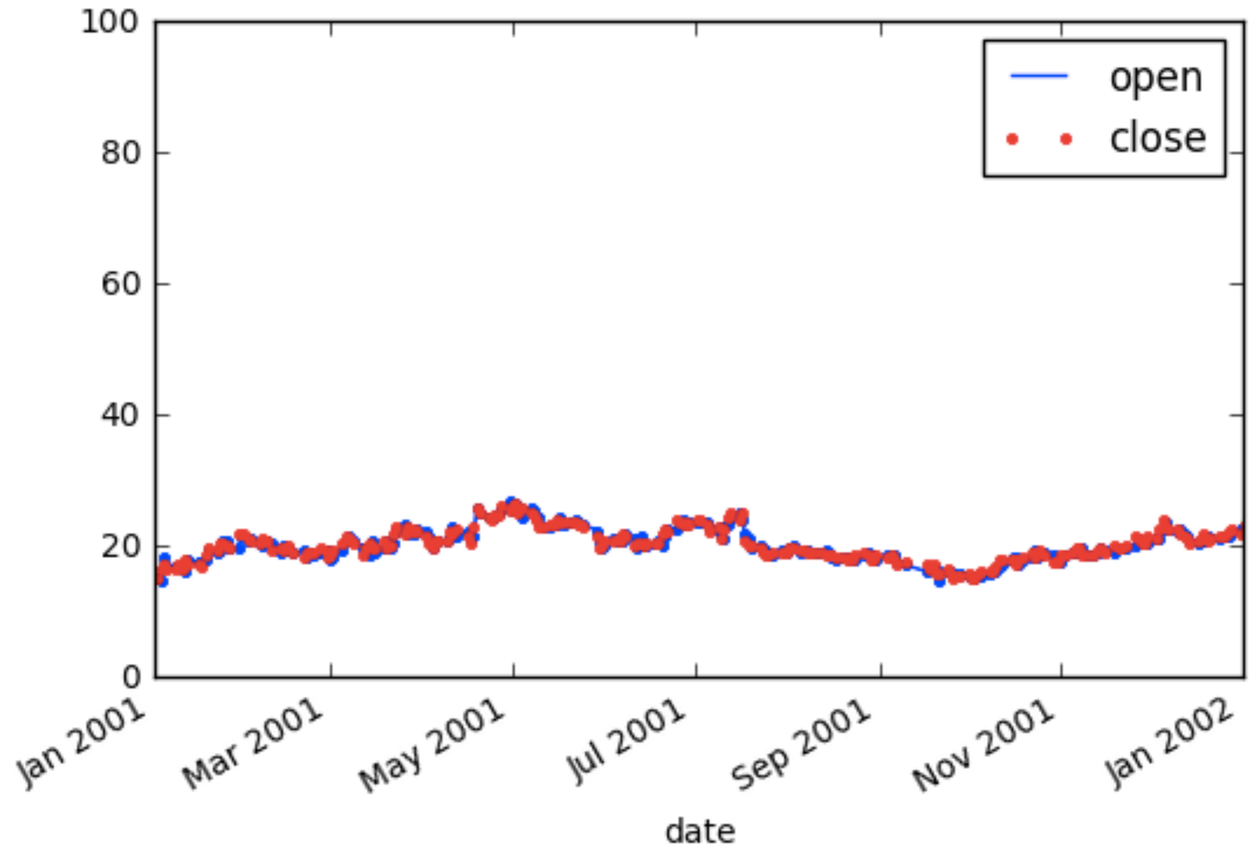
Fixing scales



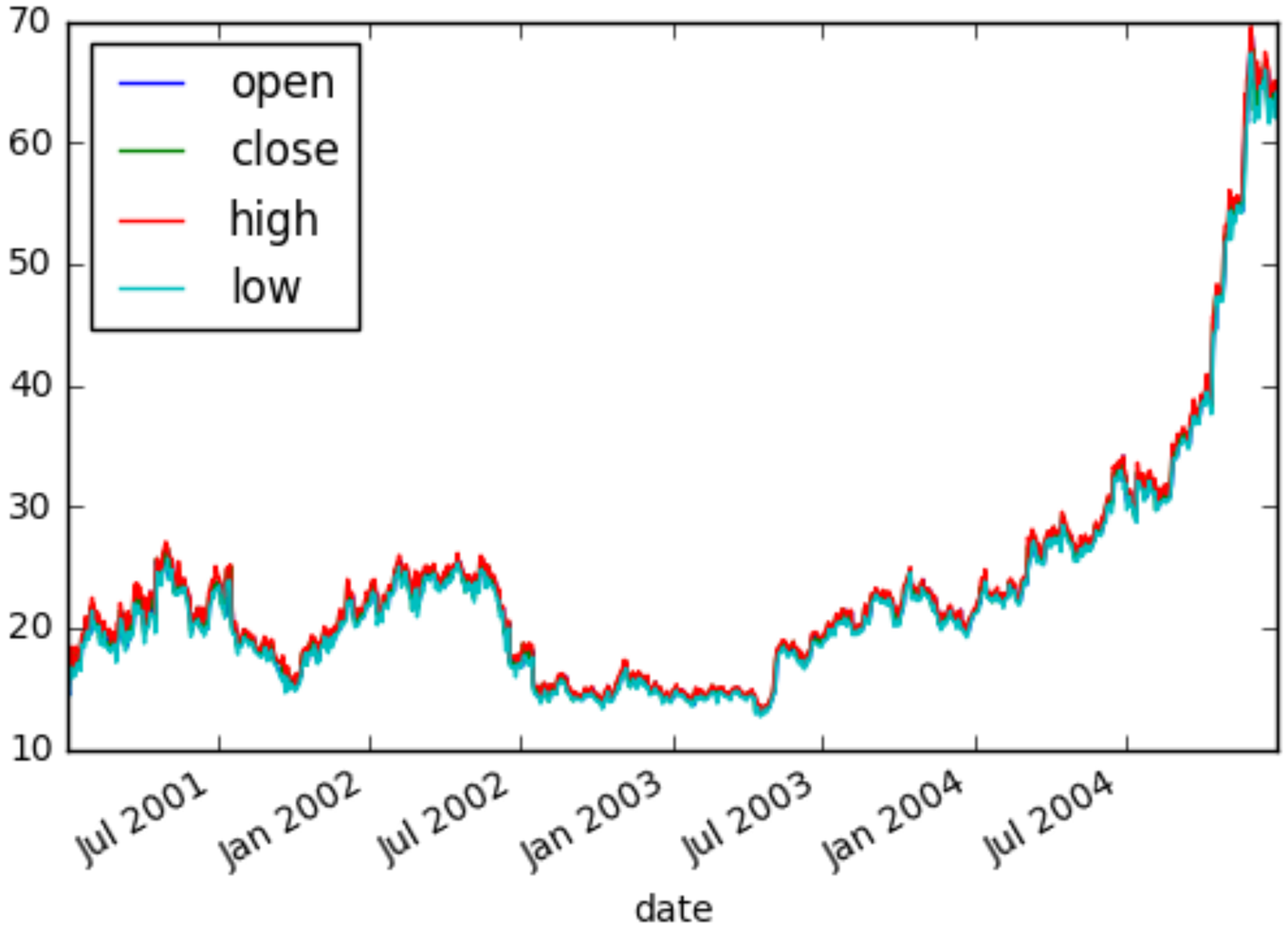
Customizing plots

```
In [22]: aapl['open'].plot(color='b', style='.-', legend=True)
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>

In [23]: aapl['close'].plot(color='r', style='.', legend=True)
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>

In [24]: plt.axis(('2001', '2002', 0, 100))
Out[24]: ('2001', '2002', 0, 100)

In [25]: plt.show()
```

Customizing plots



Saving plots



Saving plots

```
In [26]: aapl.loc['2001':'2004',['open', 'close', 'high',
    ...:               'low']].plot()
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x11ab42978>

In [27]: plt.savefig('aapl.png')

In [28]: plt.savefig('aapl.jpg')

In [29]: plt.savefig('aapl.pdf')

In [30]: plt.show()
```

PANDAS FOUNDATIONS

Let’s practice!