2-Pandas

September 3, 2016

1 Getting to know your data with Pandas

1.1 Pandas

Pandas is the Python Data Analysis Library.

Pandas is an extremely versatile tool for manipulating datasets.

It also produces high quality plots with matplotlib, and integrates nicely with other libraries that expect NumPy arrays.

The most important tool provided by Pandas is the **data frame.**

A data frame is a table in which each row and column is given a label.

1.2 Getting started

1.3 Fetching, storing and retrieving your data

More information on what types of data you can fetch

http://pandas.pydata.org/pandas-docs/stable/remote_data.html

```
In [109]: stocks = 'YELP'
          data_source = 'yahoo'
          start = datetime(2015,1,1)
          end = datetime(2015,12,31)

          yahoo_stocks = web.DataReader(stocks, data_source, start, end)

          # yahoo_stocks.head()
          yahoo_stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2015-01-02 to 2015-12-31
Data columns (total 6 columns):
Open         252 non-null float64
High         252 non-null float64
Low          252 non-null float64
Close        252 non-null float64
Volume       252 non-null int64
Adj Close    252 non-null float64
dtypes: float64(5), int64(1)
```

```
memory usage: 13.8 KB
```

```
In [110]: yahoo_stocks
```

Out[110]:

Date	Open	High	Low	Close	Volume	Adj Clo
2015-01-02	55.459999	55.599998	54.240002	55.150002	1664500	55.1500
2015-01-05	54.540001	54.950001	52.330002	52.529999	2023000	52.5299
2015-01-06	52.549999	53.930000	50.750000	52.439999	3762800	52.4399
2015-01-07	53.320000	53.750000	51.759998	52.209999	1548200	52.2099
2015-01-08	52.590000	54.139999	51.759998	53.830002	2015300	53.8300
2015-01-09	55.959999	56.990002	54.720001	56.070000	6222600	56.0700
2015-01-12	56.000000	56.060001	53.430000	54.020000	2405100	54.0200
2015-01-13	54.470001	54.799999	52.520000	53.180000	1952100	53.1800
2015-01-14	52.799999	53.680000	51.459999	52.200001	1854600	52.2000
2015-01-15	53.000000	53.610001	50.029999	50.119999	2640400	50.1199
2015-01-16	50.180000	51.490002	50.029999	51.389999	2183300	51.3899
2015-01-20	51.650002	51.779999	50.689999	51.410000	1227600	51.4100
2015-01-21	51.200001	53.500000	51.200001	53.410000	3248100	53.4100
2015-01-22	53.869999	55.279999	53.119999	54.799999	2295400	54.7999
2015-01-23	54.660000	55.639999	54.299999	55.189999	1636400	55.1899
2015-01-26	55.119999	55.790001	54.830002	55.410000	1450300	55.4100
2015-01-27	56.060001	56.160000	54.570000	55.630001	2410400	55.6300
2015-01-28	56.150002	56.150002	52.919998	53.000000	2013100	53.0000
2015-01-29	52.849998	53.310001	51.410000	52.930000	1844100	52.9300
2015-01-30	52.590000	53.419998	52.049999	52.470001	1875400	52.4700
2015-02-02	52.939999	53.500000	51.209999	53.470001	2105500	53.4700
2015-02-03	53.830002	55.930000	53.410000	55.779999	2885400	55.7799
2015-02-04	55.529999	57.070000	55.250000	56.740002	2498600	56.7400
2015-02-05	57.599998	57.700001	56.080002	57.470001	4657300	57.4700
2015-02-06	47.700001	48.169998	44.860001	45.110001	25137400	45.1100
2015-02-09	44.910000	45.040001	42.099998	42.169998	13079300	42.1699
2015-02-10	43.830002	45.549999	43.310001	44.660000	11267700	44.6600
2015-02-11	45.389999	46.430000	44.810001	46.180000	6359400	46.1800
2015-02-12	46.450001	47.840000	45.950001	47.630001	4375000	47.6300
2015-02-13	48.509998	49.049999	47.220001	47.529999	4713100	47.5299
...	...	...	...	...	...	...
2015-11-18	27.540001	28.830000	27.309999	28.230000	3091600	28.2300
2015-11-19	28.190001	28.690001	27.910000	28.059999	1487500	28.0599
2015-11-20	28.100000	31.250000	28.049999	31.209999	6697500	31.2099
2015-11-23	30.580000	30.809999	29.150000	29.860001	4029900	29.8600
2015-11-24	29.459999	30.629999	29.450001	30.010000	2584500	30.0100
2015-11-25	29.790001	30.540001	29.709999	30.510000	1287100	30.5100
2015-11-27	30.500000	30.600000	29.610001	30.180000	1058900	30.1800
2015-11-30	30.110001	30.719999	29.770000	30.129999	2015600	30.1299
2015-12-01	30.110001	30.459999	29.799999	30.309999	1886000	30.3099
2015-12-02	30.299999	32.470001	30.290001	31.389999	4650300	31.3899

```
2015-12-03  31.389999  32.240002  30.480000  30.629999   2698900  30.6299
2015-12-04  30.530001  30.860001  29.320000  30.450001   2313800  30.4500
2015-12-07  30.379999  30.639999  29.629999  30.040001   1362300  30.0400
2015-12-08  29.809999  31.379999  29.500000  30.920000   1830200  30.9200
2015-12-09  30.980000  31.139999  29.260000  30.000000   2238500  30.0000
2015-12-10  30.110001  31.299999  29.990000  30.830000   1252900  30.8300
2015-12-11  30.690001  30.750000  29.600000  29.650000   1415000  29.6500
2015-12-14  29.600000  29.889999  28.850000  29.580000   2328600  29.5800
2015-12-15  29.680000  30.000000  26.459999  26.870001   5759200  26.8700
2015-12-16  26.889999  28.240000  26.260000  28.030001   2992100  28.0300
2015-12-17  28.139999  28.320000  27.190001  27.420000   1483900  27.4200
2015-12-18  27.309999  27.910000  26.900000  27.170000   1299800  27.1700
2015-12-21  27.170000  27.360001  26.030001  26.250000   1947600  26.2500
2015-12-22  26.250000  28.700001  26.150000  27.930000   2952700  27.9300
2015-12-23  27.950001  28.420000  27.440001  28.150000   1001000  28.1500
2015-12-24  28.270000  28.590000  27.900000  28.400000    587400  28.4000
2015-12-28  28.120001  28.379999  27.770000  27.879999   1004500  27.8799
2015-12-29  27.950001  28.540001  27.740000  28.480000   1103900  28.4800
2015-12-30  28.580000  28.780001  28.170000  28.250000   1068000  28.2500
2015-12-31  28.100000  28.969999  28.020000  28.799999   1301500  28.7999

[252 rows x 6 columns]
```

More on pandas data frames:
http://pandas.pydata.org/pandas-docs/dev/generated/pandas.DataFrame.html

Reading data from a .csv file

```
In [111]: yahoo_stocks.to_csv('yahoo_data.csv')
          #print(open('yahoo_data.csv').read())

In [112]: df = pd.read_csv('yahoo_data.csv')#, index_col='Date', infer_datetime_for
          #print(df.head())
          #print(df.info())
          #print(df.index)
          df
```

```
Out[112]:          Date       Open       High        Low      Close    Volume  \
0    2015-01-02  55.459999  55.599998  54.240002  55.150002   1664500
1    2015-01-05  54.540001  54.950001  52.330002  52.529999   2023000
2    2015-01-06  52.549999  53.930000  50.750000  52.439999   3762800
3    2015-01-07  53.320000  53.750000  51.759998  52.209999   1548200
4    2015-01-08  52.590000  54.139999  51.759998  53.830002   2015300
5    2015-01-09  55.959999  56.990002  54.720001  56.070000   6222600
6    2015-01-12  56.000000  56.060001  53.430000  54.020000   2405100
7    2015-01-13  54.470001  54.799999  52.520000  53.180000   1952100
8    2015-01-14  52.799999  53.680000  51.459999  52.200001   1854600
9    2015-01-15  53.000000  53.610001  50.029999  50.119999   2640400
```

10	2015-01-16	50.180000	51.490002	50.029999	51.389999	2183300
11	2015-01-20	51.650002	51.779999	50.689999	51.410000	1227600
12	2015-01-21	51.200001	53.500000	51.200001	53.410000	3248100
13	2015-01-22	53.869999	55.279999	53.119999	54.799999	2295400
14	2015-01-23	54.660000	55.639999	54.299999	55.189999	1636400
15	2015-01-26	55.119999	55.790001	54.830002	55.410000	1450300
16	2015-01-27	56.060001	56.160000	54.570000	55.630001	2410400
17	2015-01-28	56.150002	56.150002	52.919998	53.000000	2013100
18	2015-01-29	52.849998	53.310001	51.410000	52.930000	1844100
19	2015-01-30	52.590000	53.419998	52.049999	52.470001	1875400
20	2015-02-02	52.939999	53.500000	51.209999	53.470001	2105500
21	2015-02-03	53.830002	55.930000	53.410000	55.779999	2885400
22	2015-02-04	55.529999	57.070000	55.250000	56.740002	2498600
23	2015-02-05	57.599998	57.700001	56.080002	57.470001	4657300
24	2015-02-06	47.700001	48.169998	44.860001	45.110001	25137400
25	2015-02-09	44.910000	45.040001	42.099998	42.169998	13079300
26	2015-02-10	43.830002	45.549999	43.310001	44.660000	11267700
27	2015-02-11	45.389999	46.430000	44.810001	46.180000	6359400
28	2015-02-12	46.450001	47.840000	45.950001	47.630001	4375000
29	2015-02-13	48.509998	49.049999	47.220001	47.529999	4713100
..	...	...	...	...	...	...
222	2015-11-18	27.540001	28.830000	27.309999	28.230000	3091600
223	2015-11-19	28.190001	28.690001	27.910000	28.059999	1487500
224	2015-11-20	28.100000	31.250000	28.049999	31.209999	6697500
225	2015-11-23	30.580000	30.809999	29.150000	29.860001	4029900
226	2015-11-24	29.459999	30.629999	29.450001	30.010000	2584500
227	2015-11-25	29.790001	30.540001	29.709999	30.510000	1287100
228	2015-11-27	30.500000	30.600000	29.610001	30.180000	1058900
229	2015-11-30	30.110001	30.719999	29.770000	30.129999	2015600
230	2015-12-01	30.110001	30.459999	29.799999	30.309999	1886000
231	2015-12-02	30.299999	32.470001	30.290001	31.389999	4650300
232	2015-12-03	31.389999	32.240002	30.480000	30.629999	2698900
233	2015-12-04	30.530001	30.860001	29.320000	30.450001	2313800
234	2015-12-07	30.379999	30.639999	29.629999	30.040001	1362300
235	2015-12-08	29.809999	31.379999	29.500000	30.920000	1830200
236	2015-12-09	30.980000	31.139999	29.260000	30.000000	2238500
237	2015-12-10	30.110001	31.299999	29.990000	30.830000	1252900
238	2015-12-11	30.690001	30.750000	29.600000	29.650000	1415000
239	2015-12-14	29.600000	29.889999	28.850000	29.580000	2328600
240	2015-12-15	29.680000	30.000000	26.459999	26.870001	5759200
241	2015-12-16	26.889999	28.240000	26.260000	28.030001	2992100
242	2015-12-17	28.139999	28.320000	27.190001	27.420000	1483900
243	2015-12-18	27.309999	27.910000	26.900000	27.170000	1299800
244	2015-12-21	27.170000	27.360001	26.030001	26.250000	1947600
245	2015-12-22	26.250000	28.700001	26.150000	27.930000	2952700
246	2015-12-23	27.950001	28.420000	27.440001	28.150000	1001000
247	2015-12-24	28.270000	28.590000	27.900000	28.400000	587400
248	2015-12-28	28.120001	28.379999	27.770000	27.879999	1004500

```
249  2015-12-29  27.950001  28.540001  27.740000  28.480000   1103900
250  2015-12-30  28.580000  28.780001  28.170000  28.250000   1068000
251  2015-12-31  28.100000  28.969999  28.020000  28.799999   1301500

     Adj Close
0    55.150002
1    52.529999
2    52.439999
3    52.209999
4    53.830002
5    56.070000
6    54.020000
7    53.180000
8    52.200001
9    50.119999
10   51.389999
11   51.410000
12   53.410000
13   54.799999
14   55.189999
15   55.410000
16   55.630001
17   53.000000
18   52.930000
19   52.470001
20   53.470001
21   55.779999
22   56.740002
23   57.470001
24   45.110001
25   42.169998
26   44.660000
27   46.180000
28   47.630001
29   47.529999
..         ...
222  28.230000
223  28.059999
224  31.209999
225  29.860001
226  30.010000
227  30.510000
228  30.180000
229  30.129999
230  30.309999
231  31.389999
232  30.629999
233  30.450001
```

```
234  30.040001
235  30.920000
236  30.000000
237  30.830000
238  29.650000
239  29.580000
240  26.870001
241  28.030001
242  27.420000
243  27.170000
244  26.250000
245  27.930000
246  28.150000
247  28.400000
248  27.879999
249  28.480000
250  28.250000
251  28.799999

[252 rows x 7 columns]
```

The number of tuples in your data

```
In [113]: len(df)

Out[113]: 252
```

1.4 Working with data columns

The columns or "features" in your data

```
In [114]: df.columns

Out[114]: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'], dt
```

Selecting a single column from your data

```
In [115]: df['Open']

Out[115]: 0      55.459999
          1      54.540001
          2      52.549999
          3      53.320000
          4      52.590000
          5      55.959999
          6      56.000000
          7      54.470001
          8      52.799999
          9      53.000000
```

```
10     50.180000
11     51.650002
12     51.200001
13     53.869999
14     54.660000
15     55.119999
16     56.060001
17     56.150002
18     52.849998
19     52.590000
20     52.939999
21     53.830002
22     55.529999
23     57.599998
24     47.700001
25     44.910000
26     43.830002
27     45.389999
28     46.450001
29     48.509998
         ...
222    27.540001
223    28.190001
224    28.100000
225    30.580000
226    29.459999
227    29.790001
228    30.500000
229    30.110001
230    30.110001
231    30.299999
232    31.389999
233    30.530001
234    30.379999
235    29.809999
236    30.980000
237    30.110001
238    30.690001
239    29.600000
240    29.680000
241    26.889999
242    28.139999
243    27.309999
244    27.170000
245    26.250000
246    27.950001
247    28.270000
248    28.120001
```

```
249    27.950001
250    28.580000
251    28.100000
Name: Open, dtype: float64
```

Another way of selecting a single column from your data

```
In [116]: df.Open

Out[116]: 0      55.459999
          1      54.540001
          2      52.549999
          3      53.320000
          4      52.590000
          5      55.959999
          6      56.000000
          7      54.470001
          8      52.799999
          9      53.000000
          10     50.180000
          11     51.650002
          12     51.200001
          13     53.869999
          14     54.660000
          15     55.119999
          16     56.060001
          17     56.150002
          18     52.849998
          19     52.590000
          20     52.939999
          21     53.830002
          22     55.529999
          23     57.599998
          24     47.700001
          25     44.910000
          26     43.830002
          27     45.389999
          28     46.450001
          29     48.509998
                   ...
          222    27.540001
          223    28.190001
          224    28.100000
          225    30.580000
          226    29.459999
          227    29.790001
          228    30.500000
          229    30.110001
```

```
230    30.110001
231    30.299999
232    31.389999
233    30.530001
234    30.379999
235    29.809999
236    30.980000
237    30.110001
238    30.690001
239    29.600000
240    29.680000
241    26.889999
242    28.139999
243    27.309999
244    27.170000
245    26.250000
246    27.950001
247    28.270000
248    28.120001
249    27.950001
250    28.580000
251    28.100000
Name: Open, dtype: float64
```

```
In [117]: df[['Open','Close']]
```

Out[117]:

	Open	Close
0	55.459999	55.150002
1	54.540001	52.529999
2	52.549999	52.439999
3	53.320000	52.209999
4	52.590000	53.830002
5	55.959999	56.070000
6	56.000000	54.020000
7	54.470001	53.180000
8	52.799999	52.200001
9	53.000000	50.119999
10	50.180000	51.389999
11	51.650002	51.410000
12	51.200001	53.410000
13	53.869999	54.799999
14	54.660000	55.189999
15	55.119999	55.410000
16	56.060001	55.630001
17	56.150002	53.000000
18	52.849998	52.930000
19	52.590000	52.470001
20	52.939999	53.470001

```
21   53.830002  55.779999
22   55.529999  56.740002
23   57.599998  57.470001
24   47.700001  45.110001
25   44.910000  42.169998
26   43.830002  44.660000
27   45.389999  46.180000
28   46.450001  47.630001
29   48.509998  47.529999
..         ...        ...
222  27.540001  28.230000
223  28.190001  28.059999
224  28.100000  31.209999
225  30.580000  29.860001
226  29.459999  30.010000
227  29.790001  30.510000
228  30.500000  30.180000
229  30.110001  30.129999
230  30.110001  30.309999
231  30.299999  31.389999
232  31.389999  30.629999
233  30.530001  30.450001
234  30.379999  30.040001
235  29.809999  30.920000
236  30.980000  30.000000
237  30.110001  30.830000
238  30.690001  29.650000
239  29.600000  29.580000
240  29.680000  26.870001
241  26.889999  28.030001
242  28.139999  27.420000
243  27.309999  27.170000
244  27.170000  26.250000
245  26.250000  27.930000
246  27.950001  28.150000
247  28.270000  28.400000
248  28.120001  27.879999
249  27.950001  28.480000
250  28.580000  28.250000
251  28.100000  28.799999

[252 rows x 2 columns]
```

```
In [118]: df.Date.head(10)
```

```
Out[118]: 0    2015-01-02
          1    2015-01-05
          2    2015-01-06
```

```
3    2015-01-07
4    2015-01-08
5    2015-01-09
6    2015-01-12
7    2015-01-13
8    2015-01-14
9    2015-01-15
Name: Date, dtype: object
```

```
In [119]: df.Date.tail(10)
```

```
Out[119]: 242    2015-12-17
          243    2015-12-18
          244    2015-12-21
          245    2015-12-22
          246    2015-12-23
          247    2015-12-24
          248    2015-12-28
          249    2015-12-29
          250    2015-12-30
          251    2015-12-31
          Name: Date, dtype: object
```

Changing the column names

```
In [120]: new_column_names = [x.lower().replace(' ','_') for x in df.columns]
          df.columns = new_column_names
          df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 7 columns):
date         252 non-null object
open         252 non-null float64
high         252 non-null float64
low          252 non-null float64
close        252 non-null float64
volume       252 non-null int64
adj_close    252 non-null float64
dtypes: float64(5), int64(1), object(1)
memory usage: 13.9+ KB
```

Now all columns can be accessed using the dot notation

```
In [121]: df.adj_close
```

```
Out[121]: 0      55.150002
          1      52.529999
```

```
2      52.439999
3      52.209999
4      53.830002
5      56.070000
6      54.020000
7      53.180000
8      52.200001
9      50.119999
10     51.389999
11     51.410000
12     53.410000
13     54.799999
14     55.189999
15     55.410000
16     55.630001
17     53.000000
18     52.930000
19     52.470001
20     53.470001
21     55.779999
22     56.740002
23     57.470001
24     45.110001
25     42.169998
26     44.660000
27     46.180000
28     47.630001
29     47.529999
         ...
222    28.230000
223    28.059999
224    31.209999
225    29.860001
226    30.010000
227    30.510000
228    30.180000
229    30.129999
230    30.309999
231    31.389999
232    30.629999
233    30.450001
234    30.040001
235    30.920000
236    30.000000
237    30.830000
238    29.650000
239    29.580000
240    26.870001
```

```
241    28.030001
242    27.420000
243    27.170000
244    26.250000
245    27.930000
246    28.150000
247    28.400000
248    27.879999
249    28.480000
250    28.250000
251    28.799999
Name: adj_close, dtype: float64
```

```
In [122]: df.adj_close.head()
```

```
Out[122]: 0    55.150002
          1    52.529999
          2    52.439999
          3    52.209999
          4    53.830002
          Name: adj_close, dtype: float64
```

1.5 Data Frame methods

A DataFrame object has many useful methods.

```
In [123]: df.mean()
```

```
Out[123]: open         3.728766e+01
          high         3.805464e+01
          low          3.656373e+01
          close        3.729917e+01
          volume       3.492134e+06
          adj_close    3.729917e+01
          dtype: float64
```

```
In [124]: df.std()
```

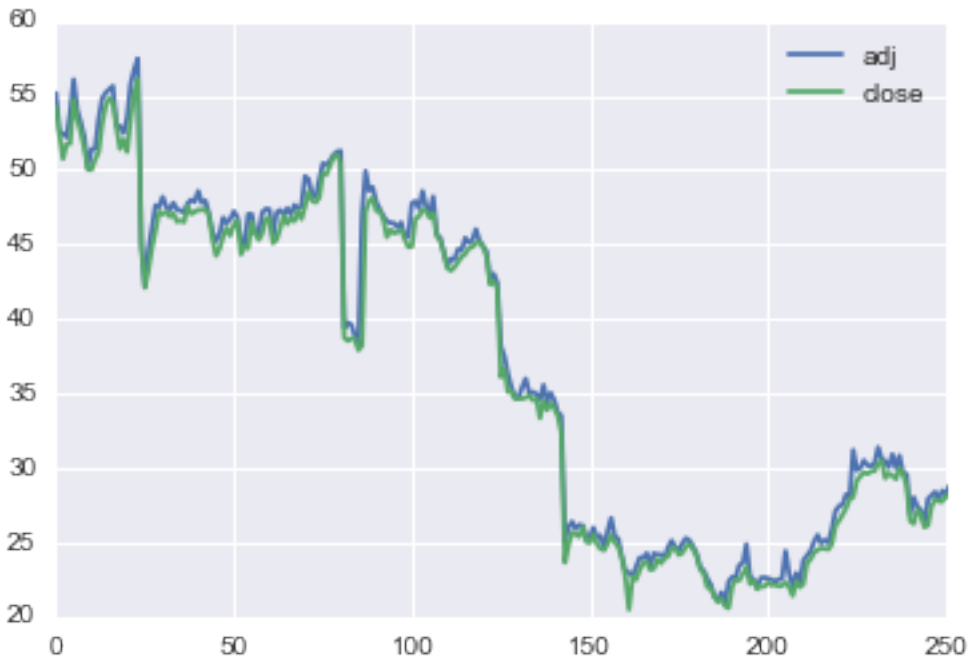
```
Out[124]: open         1.128093e+01
          high         1.138111e+01
          low          1.113097e+01
          close        1.125233e+01
          volume       4.145502e+06
          adj_close    1.125233e+01
          dtype: float64
```

```
In [125]: df.median()
```

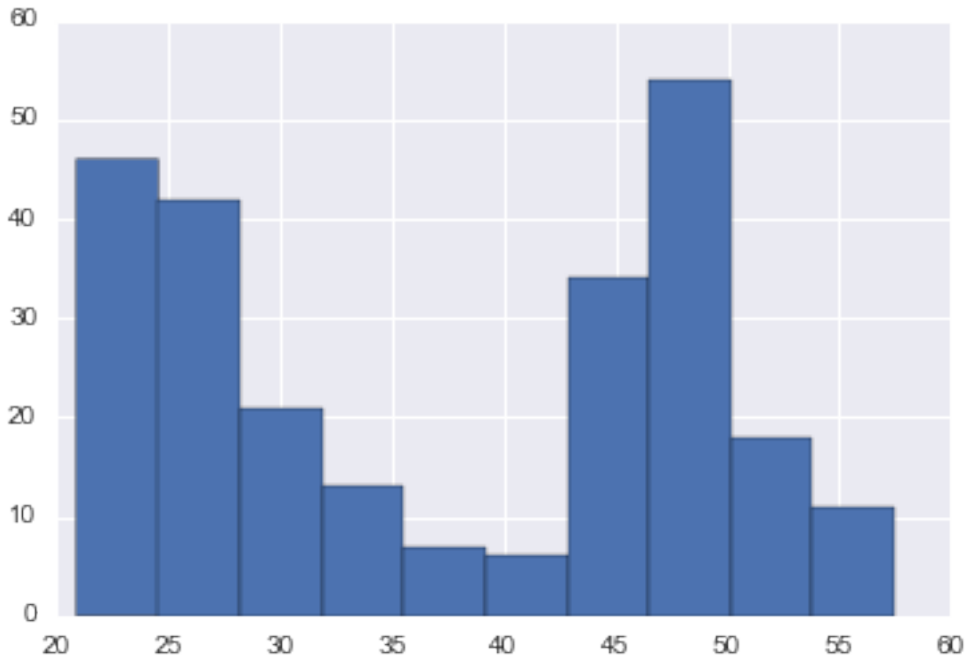
```
Out[125]: open          3.796500e+01
          high          3.871500e+01
          low           3.637500e+01
          close         3.783500e+01
          volume        2.354050e+06
          adj_close     3.783500e+01
          dtype: float64

In [126]: df.open.mean()

Out[126]: 37.287658698412692

In [127]: df.high.mean()

Out[127]: 38.054642952380952
```

1.5.1 Plotting methods

```
In [128]: df.adj_close.plot(label='adj')
          df.low.plot(label='close')
          plt.legend(loc='best')

Out[128]: <matplotlib.legend.Legend at 0x11cc669e8>
```




```
In [129]: df.adj_close.hist()

Out[129]: <matplotlib.axes._subplots.AxesSubplot at 0x11cc704a8>
```



1.5.2 Bulk Operations

Methods like **sum( )** and **std( )** work on entire columns.

We can run our own functions across all values in a column (or row) using **apply( )**.

```
In [130]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 7 columns):
date         252 non-null object
open         252 non-null float64
high         252 non-null float64
low          252 non-null float64
close        252 non-null float64
volume       252 non-null int64
adj_close    252 non-null float64
dtypes: float64(5), int64(1), object(1)
memory usage: 13.9+ KB
```

```
In [131]: df.date.head()
```

```
Out[131]: 0    2015-01-02
          1    2015-01-05
          2    2015-01-06
```

```
3    2015-01-07
4    2015-01-08
Name: date, dtype: object
```

The **values** property of the column returns a list of values for the column. Inspecting the first value reveals that these are strings with a particular format.

```
In [132]: first_date = df.date.values[0]
          first_date

Out[132]: '2015-01-02'

In [133]: datetime.strptime(first_date, "%Y-%m-%d")

Out[133]: datetime.datetime(2015, 1, 2, 0, 0)

In [134]: df.date = df.date.apply(lambda d: datetime.strptime(d, "%Y-%m-%d"))
          df.date.head()

Out[134]: 0   2015-01-02
          1   2015-01-05
          2   2015-01-06
          3   2015-01-07
          4   2015-01-08
          Name: date, dtype: datetime64[ns]
```

Each row in a DataFrame is associated with an index, which is a label that uniquely identifies a row.

The row indices so far have been auto-generated by pandas, and are simply integers starting from 0.

From now on we will use dates instead of integers for indices – the benefits of this will show later.

Overwriting the index is as easy as assigning to the index property of the DataFrame.

```
In [135]: df.index = df.date
          df.head()

Out[135]:                  date       open       high        low      close   volum
          date
          2015-01-02 2015-01-02  55.459999  55.599998  54.240002  55.150002  166450
          2015-01-05 2015-01-05  54.540001  54.950001  52.330002  52.529999  202300
          2015-01-06 2015-01-06  52.549999  53.930000  50.750000  52.439999  376280
          2015-01-07 2015-01-07  53.320000  53.750000  51.759998  52.209999  154820
          2015-01-08 2015-01-08  52.590000  54.139999  51.759998  53.830002  201530

                      adj_close
          date
          2015-01-02  55.150002
          2015-01-05  52.529999
          2015-01-06  52.439999
          2015-01-07  52.209999
          2015-01-08  53.830002
```

Now that we have made an index based on date, we can drop the original `date` column.

```
In [136]: df = df.drop(['date'],axis=1)
          df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2015-01-02 to 2015-12-31
Data columns (total 6 columns):
open         252 non-null float64
high         252 non-null float64
low          252 non-null float64
close        252 non-null float64
volume       252 non-null int64
adj_close    252 non-null float64
dtypes: float64(5), int64(1)
memory usage: 13.8 KB
```

1.5.3 Accessing rows of the DataFrame

So far we've seen how to access a column of the DataFrame. To access a row we use a different notation.

To access a row by its index value, use the `.ix()` method.

```
In [137]: df.ix[datetime(2015,1,23,0,0)]

Out[137]: open         5.466000e+01
          high         5.564000e+01
          low          5.430000e+01
          close        5.519000e+01
          volume       1.636400e+06
          adj_close    5.519000e+01
          Name: 2015-01-23 00:00:00, dtype: float64
```

To access a row by its sequence number (ie, like an array index), use `.iloc()` ('Integer Location')

```
In [138]: df.iloc[0,:]

Out[138]: open         5.546000e+01
          high         5.560000e+01
          low          5.424000e+01
          close        5.515000e+01
          volume       1.664500e+06
          adj_close    5.515000e+01
          Name: 2015-01-02 00:00:00, dtype: float64
```

To iterate over the rows, use `.iterrows()`

```
In [139]: num_positive_days = 0
          for idx, row in df.iterrows():
              if row.close > row.open:
                  num_positive_days += 1

          print("The total number of positive-gain days is {}.".format(num_positive

The total number of positive-gain days is 126.
```

1.6 Filtering

It is very easy to select interesting rows from the data.

All these operations below return a new DataFrame, which itself can be treated the same way as all DataFrames we have seen so far.

```
In [140]: positive_days = df[df.close > df.open]
          positive_days.head()

Out[140]:                  open       high        low      close   volume  adj_clos
          date
          2015-01-08  52.590000  54.139999  51.759998  53.830002  2015300  53.8300
          2015-01-09  55.959999  56.990002  54.720001  56.070000  6222600  56.0700
          2015-01-16  50.180000  51.490002  50.029999  51.389999  2183300  51.3899
          2015-01-21  51.200001  53.500000  51.200001  53.410000  3248100  53.4100
          2015-01-22  53.869999  55.279999  53.119999  54.799999  2295400  54.7999

In [141]: very_positive_days = df[df.close-df.open > 4]
          very_positive_days.head()

Out[141]:                  open   high        low      close    volume  adj_close
          date
          2015-05-07  38.220001  48.73  38.220001  47.009998  33831600  47.009998

In [142]: tmp_high = df.high > 60
          tmp_high.head()

Out[142]: date
          2015-01-02    False
          2015-01-05    False
          2015-01-06    False
          2015-01-07    False
          2015-01-08    False
          Name: high, dtype: bool

In [143]: len(tmp_high)

Out[143]: 252
```

Select only the rows of **df** that correspond to **tmp_high**

```
In [148]: df[tmp_high]
```

```
Out[148]: Empty DataFrame
          Columns: [open, high, low, close, volume, adj_close]
          Index: []
```

1.7 Creating new columns

```
In [149]: df['profit'] = (df.open < df.close)
          df.head()
```

```
Out[149]:                 open       high        low      close   volume  adj_clos
          date
          2015-01-02  55.459999  55.599998  54.240002  55.150002  1664500  55.15000
          2015-01-05  54.540001  54.950001  52.330002  52.529999  2023000  52.52999
          2015-01-06  52.549999  53.930000  50.750000  52.439999  3762800  52.43999
          2015-01-07  53.320000  53.750000  51.759998  52.209999  1548200  52.20999
          2015-01-08  52.590000  54.139999  51.759998  53.830002  2015300  53.83000

                      profit
          date
          2015-01-02  False
          2015-01-05  False
          2015-01-06  False
          2015-01-07  False
          2015-01-08   True
```

```
In [150]: for idx, row in df.iterrows():
              if row.close > row.open:
                  df.ix[idx,'gain']='negative'
              elif (row.open - row.close) < 1:
                  df.ix[idx,'gain']='small_gain'
              elif (row.open - row.close) < 6:
                  df.ix[idx,'gain']='medium_gain'
              else:
                  df.ix[idx,'gain']='large_gain'
          df.head()
```

```
Out[150]:                 open       high        low      close   volume  adj_clos
          date
          2015-01-02  55.459999  55.599998  54.240002  55.150002  1664500  55.15000
          2015-01-05  54.540001  54.950001  52.330002  52.529999  2023000  52.52999
          2015-01-06  52.549999  53.930000  50.750000  52.439999  3762800  52.43999
          2015-01-07  53.320000  53.750000  51.759998  52.209999  1548200  52.20999
          2015-01-08  52.590000  54.139999  51.759998  53.830002  2015300  53.83000

                      profit         gain
          date
          2015-01-02  False   small_gain
```

```
2015-01-05  False  medium_gain
2015-01-06  False   small_gain
2015-01-07  False  medium_gain
2015-01-08   True     negative
```

Alternatively

```
In [ ]: def namerow(row):
            if row.close > row.open:
                return 'negative'
            elif (row.open - row.close) < 1:
                return 'small_gain'
            elif (row.open - row.close) < 6:
                return 'medium_gain'
            else:
                return 'large_gain'

        df['test_column'] = df.apply(namerow, axis = 1)

In [ ]: print df.head()

In [ ]: df.drop('test_column', axis = 1)
```

1.8 Grouping

Besides **apply()**, another great DataFrame function is **groupby()**. It will group a DataFrame by one or more columns, and let you iterate through each group.

Here we will show the average gain among the three categories of negative, small, medium and large gains we defined above and stored in column “gain"

```
In [ ]: gains = {}
        for gain, gain_data in df.groupby("gain"):
            gains[gain] = gain_data.close.mean()
        gains
```

1.9 Comparing multiple stocks

As a last task, we will use the experience we obtained so far – and learn some new things – in order to compare the performance of different stocks we obtained from Yahoo finance.

```
In [ ]: stocks = ['ORCL', 'TSLA', 'IBM','YELP', 'MSFT']
        attr = 'Close'
        df = pd.io.data.get_data_yahoo(stocks,
                                       start=datetime(2014, 1, 1),
                                       end=datetime(2014, 12, 31))[attr]
        print df.head()

In [ ]: df.ORCL.plot(label = 'oracle')
        df.TSLA.plot(label = 'tesla')
```

```
df.IBM.plot(label = 'ibm')
df.MSFT.plot(label = 'msft')
df.YELP.plot(label = 'yelp')
plt.legend(loc=5)
```

Calculating returns over a period of length T

$$r(t) = \frac{f(t) - f(t-T)}{f(t)}$$

The returns can be computed in python with a simple function **pct_returns( )**

```
In [ ]: rets = df.pct_change(30)
        #print rets.head(20)
```

Plotting again the timeseries of the returns of the different stocks

```
In [ ]: rets.ORCL.plot(label = 'oracle')
        rets.TSLA.plot(label = 'tesla')
        rets.IBM.plot(label = 'ibm')
        rets.MSFT.plot(label = 'msft')
        rets.YELP.plot(label = 'yelp')
        plt.legend()

In [ ]: plt.scatter(rets.TSLA, rets.ORCL)
        plt.xlabel('Returns TESLA')
        plt.ylabel('Returns ORCL')
```

Correlations of columns The correlation coefficient between variables X and Y is defined as follows:

$$\text{Corr}(X, Y) = \frac{E\left[(X - \mu_X)(Y - \mu_Y)\right]}{\sigma_X \sigma_Y}$$

In python we can compute the correlation coefficient of all pairs of columns with **corr( )**

```
In [ ]: corr = rets.corr()
        print corr
```

Visualizing the correlation coefficient of all columns We will learn more about visualization later, but for now this is a simple example

```
In [ ]: sns.heatmap(corr, annot=True)
```

Returns vs risk (standard deviation) In many applications, we want to know both the returns as well as the standard deviation of the returns of a stock (i.e., its risk). Below we visualize the result of such an analysis

In [ ]:
```
plt.scatter(rets.mean(), rets.std())
plt.xlabel('Expected returns')
plt.ylabel('Standard Deviation (Risk)')
for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(
        label,
        xy = (x, y), xytext = (20, -20),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        bbox = dict(boxstyle = 'round,pad=0.5', fc = 'yellow', alpha = 0.5)
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'
```

In [ ]:
```
# Code for setting the style of the notebook
from IPython.core.display import HTML
def css_styling():
    styles = open("../theme/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

In [ ]: