stock_data

September 27, 2018

0.1 Stock Data

Now that you've had exposure to time series data, let's look at bringing stock prices into Pandas. ## Reading in Data Your dataset can come in a variety of different formats. The most common format is the CSV. We'll use the "prices.csv" file as an example csv file.

```
In [1]: with open('prices.csv', 'r') as file:
            prices = file.read()

        print(prices)
```

```
ABC,2017-09-05,163.09,164.24,160.21,162.63,29417590.0,162.49,29414672.0
ABC,2017-09-06,162.85,162.46,159.99,161.13,21131267.0,162.44,21169319.0
ABC,2017-09-07,162.11,162.7,160.65,161.26,21722502.0,161.46,21719856.0
ABC,2017-09-08,160.41,160.89,159.31,158.05,28311012.0,158.26,28305810.0
ABC,2017-09-11,161.09,162.14,159.54,161.29,31075573.0,160.97,31163734.0
ABC,2017-09-12,162.54,164.61,159.52,161.09,70921229.0,160.62,71097150.0
ABC,2017-09-13,160.01,160.51,158.22,159.29,44580353.0,159.07,44260255.0
EFG,2017-09-05,154.45,154.69,153.17,154.52,1270203.0,153.58,1270679.0
EFG,2017-09-06,155.03,155.14,153.89,154.45,1195987.0,154.06,1196107.0
EFG,2017-09-07,154.73,155.36,153.6,155.68,1420730.0,155.6,1409098.0
EFG,2017-09-08,156.01,155.91,154.17,155.86,1438929.0,156.08,1445338.0
EFG,2017-09-11,157.07,157.71,155.93,157.17,1608840.0,156.7,1610357.0
EFG,2017-09-12,155.98,156.72,154.28,156.71,1692197.0,156.86,1687819.0
EFG,2017-09-13,156.4,157.07,155.68,155.54,1211779.0,155.85,1210716.0
XYZ,2017-09-05,63.9,64.51,63.13,63.95,1738651.0,63.33,1733249.0
XYZ,2017-09-06,63.85,63.65,61.72,62.23,3730110.0,61.95,3725435.0
XYZ,2017-09-07,61.97,61.93,59.47,60.46,6166046.0,60.64,6191712.0
XYZ,2017-09-08,60.36,60.45,58.51,59.35,5173590.0,59.4,5174940.0
XYZ,2017-09-11,60.04,59.92,57.68,58.24,5003322.0,58.02,5001118.0
XYZ,2017-09-12,58.19,59.29,57.89,58.71,3633446.0,58.96,3635132.0
XYZ,2017-09-13,59.01,60.66,58.8,60.33,3571591.0,60.46,3583560.0
```

The data provider will provide you with information for each field in the CSV. This csv has the fields ticker, date, open, high, low, close, volume, adj_close, adj_volume in that order. That means, the first line in the CSV has the following data:

- ticker: ABC
- date: 2017-09-05
- open: 163.09
- high: 164.24
- low: 160.21
- close: 162.63
- volume: 29417590.0
- adj_close: 162.49
- adj_volume: 29414672.0

Let's move this data into a DataFrame. For this, we'll need to use the `pd.read_csv` function. This allows you generate a DataFrame from CSV data.

```
In [2]: import pandas as pd

        price_df = pd.read_csv('prices.csv')

        price_df
```

```
Out[2]:     ABC  2017-09-05  163.09  164.24  160.21  162.63  29417590.0  162.49  \
        0   ABC  2017-09-06  162.85  162.46  159.99  161.13  21131267.0  162.44
        1   ABC  2017-09-07  162.11  162.70  160.65  161.26  21722502.0  161.46
        2   ABC  2017-09-08  160.41  160.89  159.31  158.05  28311012.0  158.26
        3   ABC  2017-09-11  161.09  162.14  159.54  161.29  31075573.0  160.97
        4   ABC  2017-09-12  162.54  164.61  159.52  161.09  70921229.0  160.62
        5   ABC  2017-09-13  160.01  160.51  158.22  159.29  44580353.0  159.07
        6   EFG  2017-09-05  154.45  154.69  153.17  154.52   1270203.0  153.58
        7   EFG  2017-09-06  155.03  155.14  153.89  154.45   1195987.0  154.06
        8   EFG  2017-09-07  154.73  155.36  153.60  155.68   1420730.0  155.60
        9   EFG  2017-09-08  156.01  155.91  154.17  155.86   1438929.0  156.08
        10  EFG  2017-09-11  157.07  157.71  155.93  157.17   1608840.0  156.70
        11  EFG  2017-09-12  155.98  156.72  154.28  156.71   1692197.0  156.86
        12  EFG  2017-09-13  156.40  157.07  155.68  155.54   1211779.0  155.85
        13  XYZ  2017-09-05   63.90   64.51   63.13   63.95   1738651.0   63.33
        14  XYZ  2017-09-06   63.85   63.65   61.72   62.23   3730110.0   61.95
        15  XYZ  2017-09-07   61.97   61.93   59.47   60.46   6166046.0   60.64
        16  XYZ  2017-09-08   60.36   60.45   58.51   59.35   5173590.0   59.40
        17  XYZ  2017-09-11   60.04   59.92   57.68   58.24   5003322.0   58.02
        18  XYZ  2017-09-12   58.19   59.29   57.89   58.71   3633446.0   58.96
        19  XYZ  2017-09-13   59.01   60.66   58.80   60.33   3571591.0   60.46

            29414672.0
        0   21169319.0
        1   21719856.0
        2   28305810.0
        3   31163734.0
        4   71097150.0
        5   44260255.0
```

```
6    1270679.0
7    1196107.0
8    1409098.0
9    1445338.0
10   1610357.0
11   1687819.0
12   1210716.0
13   1733249.0
14   3725435.0
15   6191712.0
16   5174940.0
17   5001118.0
18   3635132.0
19   3583560.0
```

That generated a DataFrame using the CSV, but assumed the first row contains the field names. We'll have to supply the function's parameter `names` with a list of fiels names.

```
In [3]: price_df = pd.read_csv('prices.csv', names=['ticker', 'date', 'open', 'high', 'low',
                                                    'close', 'volume', 'adj_close', 'adj_volume
```

```
price_df
```

```
Out[3]:    ticker        date    open    high     low   close      volume  adj_close  \
0     ABC  2017-09-05  163.09  164.24  160.21  162.63  29417590.0     162.49
1     ABC  2017-09-06  162.85  162.46  159.99  161.13  21131267.0     162.44
2     ABC  2017-09-07  162.11  162.70  160.65  161.26  21722502.0     161.46
3     ABC  2017-09-08  160.41  160.89  159.31  158.05  28311012.0     158.26
4     ABC  2017-09-11  161.09  162.14  159.54  161.29  31075573.0     160.97
5     ABC  2017-09-12  162.54  164.61  159.52  161.09  70921229.0     160.62
6     ABC  2017-09-13  160.01  160.51  158.22  159.29  44580353.0     159.07
7     EFG  2017-09-05  154.45  154.69  153.17  154.52   1270203.0     153.58
8     EFG  2017-09-06  155.03  155.14  153.89  154.45   1195987.0     154.06
9     EFG  2017-09-07  154.73  155.36  153.60  155.68   1420730.0     155.60
10    EFG  2017-09-08  156.01  155.91  154.17  155.86   1438929.0     156.08
11    EFG  2017-09-11  157.07  157.71  155.93  157.17   1608840.0     156.70
12    EFG  2017-09-12  155.98  156.72  154.28  156.71   1692197.0     156.86
13    EFG  2017-09-13  156.40  157.07  155.68  155.54   1211779.0     155.85
14    XYZ  2017-09-05   63.90   64.51   63.13   63.95   1738651.0      63.33
15    XYZ  2017-09-06   63.85   63.65   61.72   62.23   3730110.0      61.95
16    XYZ  2017-09-07   61.97   61.93   59.47   60.46   6166046.0      60.64
17    XYZ  2017-09-08   60.36   60.45   58.51   59.35   5173590.0      59.40
18    XYZ  2017-09-11   60.04   59.92   57.68   58.24   5003322.0      58.02
19    XYZ  2017-09-12   58.19   59.29   57.89   58.71   3633446.0      58.96
20    XYZ  2017-09-13   59.01   60.66   58.80   60.33   3571591.0      60.46

    adj_volume
0   29414672.0
```

```
1   21169319.0
2   21719856.0
3   28305810.0
4   31163734.0
5   71097150.0
6   44260255.0
7    1270679.0
8    1196107.0
9    1409098.0
10   1445338.0
11   1610357.0
12   1687819.0
13   1210716.0
14   1733249.0
15   3725435.0
16   6191712.0
17   5174940.0
18   5001118.0
19   3635132.0
20   3583560.0
```

0.2 DataFrame Calculations

Now that we have the data in a DataFrame, we can start to do calculations on it. Let's find out the median value for each stock using the `DataFrame.median` function.

```
In [4]: price_df.median()

Out[4]: open              155.98
        high              155.91
        low               154.17
        close             155.68
        volume        3730110.00
        adj_close         155.85
        adj_volume    3725435.00
        dtype: float64
```

That's not right. Those are the median values for the whole stock universe. We'll use the `DataFrame.groupby` function to get mean for each stock.

```
In [5]: price_df.groupby('ticker').median()

Out[5]:           open    high     low   close      volume  adj_close  adj_volume
        ticker
        ABC     162.11  162.46  159.54  161.13  29417590.0     160.97  29414672.0
        EFG     155.98  155.91  154.17  155.68   1420730.0     155.85   1409098.0
        XYZ      60.36   60.66   58.80   60.33   3730110.0      60.46   3725435.0
```

That's what we're looking for! However, we don't want to run the `groupby` function each time we make an operation. We could save the GroupBy object by doing `price_df_ticker_groups = price_df.groupby('ticker')`. This limits us to the operations of GroupBy objects. There's the `GroupBy.apply`, but then we lose out on performance. The true problem is the way the data is represented.

```
In [6]: price_df.iloc[:16]
```

Out[6]:

	ticker	date	open	high	low	close	volume	adj_close	adj_volume
0	ABC	2017-09-05	163.09	164.24	160.21	162.63	29417590.0	162.49	29414672.0
1	ABC	2017-09-06	162.85	162.46	159.99	161.13	21131267.0	162.44	21169319.0
2	ABC	2017-09-07	162.11	162.70	160.65	161.26	21722502.0	161.46	21719856.0
3	ABC	2017-09-08	160.41	160.89	159.31	158.05	28311012.0	158.26	28305810.0
4	ABC	2017-09-11	161.09	162.14	159.54	161.29	31075573.0	160.97	31163734.0
5	ABC	2017-09-12	162.54	164.61	159.52	161.09	70921229.0	160.62	71097150.0
6	ABC	2017-09-13	160.01	160.51	158.22	159.29	44580353.0	159.07	44260255.0
7	EFG	2017-09-05	154.45	154.69	153.17	154.52	1270203.0	153.58	1270679.0
8	EFG	2017-09-06	155.03	155.14	153.89	154.45	1195987.0	154.06	1196107.0
9	EFG	2017-09-07	154.73	155.36	153.60	155.68	1420730.0	155.60	1409098.0
10	EFG	2017-09-08	156.01	155.91	154.17	155.86	1438929.0	156.08	1445338.0
11	EFG	2017-09-11	157.07	157.71	155.93	157.17	1608840.0	156.70	1610357.0
12	EFG	2017-09-12	155.98	156.72	154.28	156.71	1692197.0	156.86	1687819.0
13	EFG	2017-09-13	156.40	157.07	155.68	155.54	1211779.0	155.85	1210716.0
14	XYZ	2017-09-05	63.90	64.51	63.13	63.95	1738651.0	63.33	1733249.0
15	XYZ	2017-09-06	63.85	63.65	61.72	62.23	3730110.0	61.95	3725435.0

Can you spot our problem? Take a moment to see if you can find it.
The problem is between lines [6, 7] and [13, 14]

```
In [7]: price_df.iloc[[6, 7, 13, 14]]
```

```
Out[7]:    ticker        date    open    high     low   close      volume  adj_close  \
        6     ABC  2017-09-13  160.01  160.51  158.22  159.29  44580353.0     159.07
        7     EFG  2017-09-05  154.45  154.69  153.17  154.52   1270203.0     153.58
        13    EFG  2017-09-13  156.40  157.07  155.68  155.54   1211779.0     155.85
        14    XYZ  2017-09-05   63.90   64.51   63.13   63.95   1738651.0      63.33

            adj_volume
        6   44260255.0
        7    1270679.0
        13   1210716.0
        14   1733249.0
```

Data for all the tickers are stacked. We're representing 3 dimensional data in 2 dimensions. This was solved using Panda's Panels, which is deprecated. The Pandas documentation recommends we use either MultiIndex or xarray. MultiIndex still doesn't solve our problem, since the data is still represented in 2 dimensions. xarray is able to store 3 dimensional data, but Finance uses Pandas, so we'll stick with this library. After you finish this program, I recommend you check out xarray.

So, how do we use our 3-dimensional data with Pandas? We can split each 3rd dimension into it's own 2 dimension DataFrame. Let's take this array as an example:

```
[
    [
        [ 0,  1],
        [ 2,  3],
        [ 4,  5]
    ],[
        [ 6,  7],
        [ 8,  9],
        [10, 11]
    ],[
        [12, 13],
        [14, 15],
        [16, 17]
    ],[
        [18, 19],
        [20, 21],
        [22, 23]
    ]
]
```

We want to split it into these two 2d arrays:

```
[
    [0, 2, 4],
    [6, 8, 10],
    [12, 14, 16],
    [18, 20, 22]
]
```

```
[
    [1, 3, 5],
    [7, 8, 11],
    [13, 15, 17],
    [19, 21, 23]
]
```

In our case, our third dimensions are "open", "high", "low", "close", "volume", "adj_close", and "adj_volume". We'll use the `DataFrame.pivot` function to generate these DataFrames.

```
In [8]: open_prices = price_df.pivot(index='date', columns='ticker', values='open')
        high_prices = price_df.pivot(index='date', columns='ticker', values='high')
        low_prices = price_df.pivot(index='date', columns='ticker', values='low')
        close_prices = price_df.pivot(index='date', columns='ticker', values='close')
        volume = price_df.pivot(index='date', columns='ticker', values='volume')
        adj_close_prices = price_df.pivot(index='date', columns='ticker', values='adj_close')
        adj_volume = price_df.pivot(index='date', columns='ticker', values='adj_volume')

        open_prices
```

Out[8]:

ticker	ABC	EFG	XYZ
date			
2017-09-05	163.09	154.45	63.90
2017-09-06	162.85	155.03	63.85
2017-09-07	162.11	154.73	61.97
2017-09-08	160.41	156.01	60.36
2017-09-11	161.09	157.07	60.04
2017-09-12	162.54	155.98	58.19
2017-09-13	160.01	156.40	59.01

That gives you DataFrames for all the open, high low, etc.. Now, what we have been waiting for.. The mean for each ticker.

```
In [9]: open_prices.mean()
```

```
Out[9]: ticker
        ABC    161.728571
        EFG    155.667143
        XYZ     61.045714
        dtype: float64
```

We can also get the mean for each date by doing a transpose.

```
In [10]: open_prices.T.mean()
```

```
Out[10]: date
         2017-09-05    127.146667
         2017-09-06    127.243333
         2017-09-07    126.270000
```

```
2017-09-08    125.593333
2017-09-11    126.066667
2017-09-12    125.570000
2017-09-13    125.140000
dtype: float64
```

It doesn't matter whether date is the index and tickers are the colums or the other way around. It's always a transpose away. Since we're going to do a lot of operations across dates, we will stick with date as the index and tickers as the colums throughtout this program. ## Quiz Let's see if you can apply what you learned. Implment the `csv_to_pivot` function to take in a filepath, `csv_filename`, and output the close 2d array. You can assume the CSV file used by `csv_to_close` has the same field names as "prices.csv" and in the same order.

To help with your implemention of quizzes, we provide you with unit tests to test your function implemention. For this quiz, we'll be using the function `test_csv_to_close` in the `quiz_tests` module to test `csv_to_close`.

```
In [11]: import quiz_tests
```

```
def csv_to_close(csv_filepath, field_names):
    """Reads in data from a csv file and produces a DataFrame with close data.

    Parameters
    ----------
    csv_filepath : str
        The name of the csv file to read
    field_names : list of str
        The field names of the field in the csv file

    Returns
    -------
    close : DataFrame
        Close prices for each ticker and date
    """

    # TODO: Implement Function

    return pd.read_csv(csv_filepath, names = field_names).pivot(index = 'date', columns


quiz_tests.test_csv_to_close(csv_to_close)
```

```
Tests Passed
```

0.3 Quiz Solution

If you're having trouble, you can check out the quiz solution here.