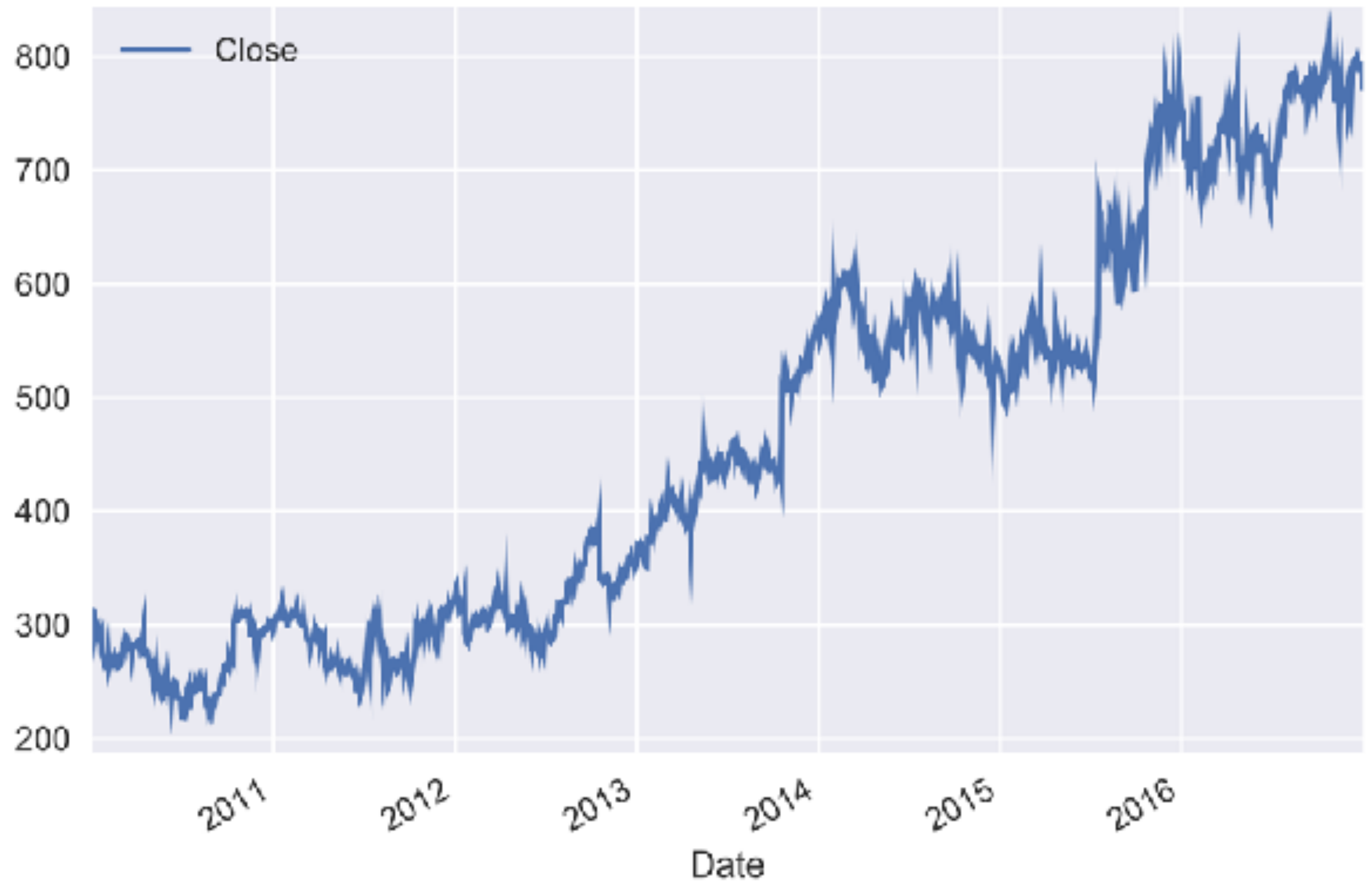
Rolling Window Functions with Pandas

Window Functions in pandas

- Windows identify sub periods of your time series
- Calculate metrics for sub periods inside the window
- Create a new time series of metrics
- Two types of windows:
    - Rolling: same size, sliding (this video)
    - Expanding: contain all prior values (next video)

Calculating a Rolling Average

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

DatetimeIndex: 1761 entries, 2010-01-04 to 2016-12-30
Data columns (total 1 columns):
price     1761 non-null float64
dtypes: float64(1)
```



Calculating a Rolling Average

```
# Integer-based window size
In [5]: data.rolling(window=30).mean() # fixed # observations

DatetimeIndex: 1761 entries, 2010-01-04 to 2017-05-24
Data columns (total 1 columns):
price     1732 non-null float64
dtypes: float64(1)
```

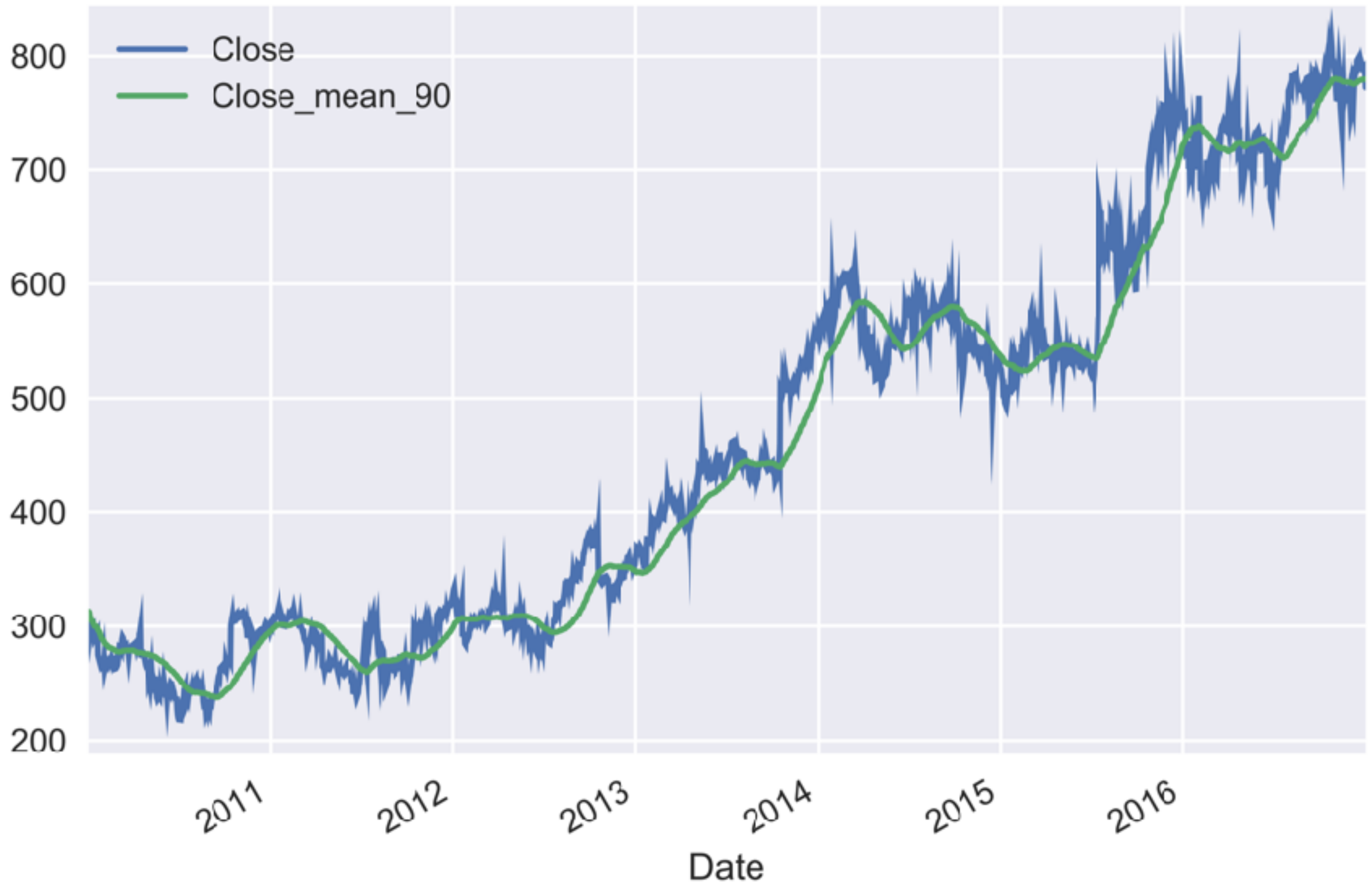
`window=30`: # business days
`min_periods`: choose value < 30 to get results for first days

```
# Offset-based window size
In [6]: data.rolling(window='30D').mean() # fixed period length

DatetimeIndex: 1761 entries, 2010-01-04 to 2017-05-24
Data columns (total 1 columns):
price     1761 non-null float64
dtypes: float64(1)
```

`30D`: # calendar days

90 Day Rolling Mean

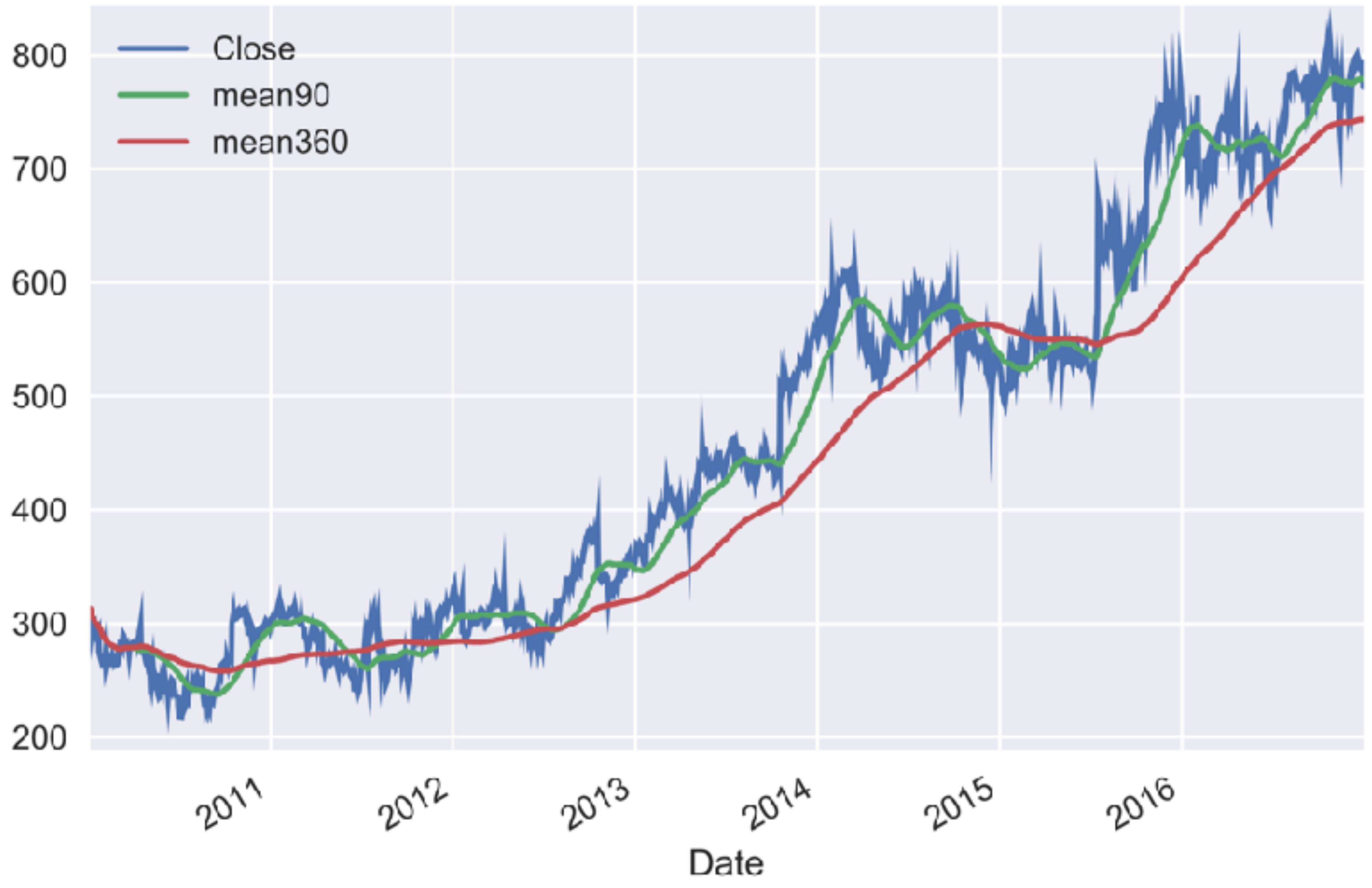
```
In [7]: r90 = data.rolling(window='90D').mean()

In [8]: google.join(r90.add_suffix('_mean_90')).plot()
```



.join: concatenate Series or DataFrame along axis=1

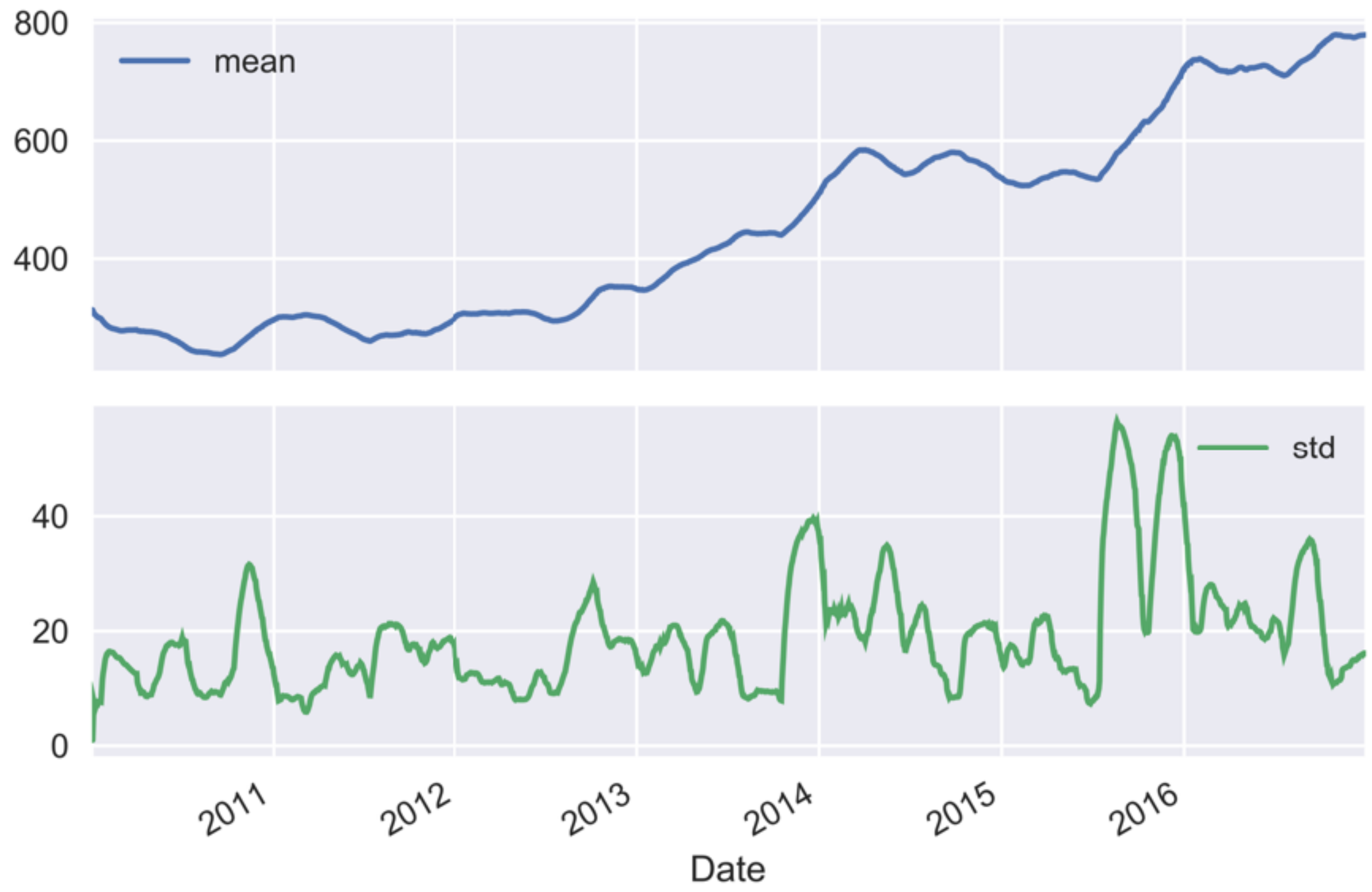
90 & 360 Day Rolling Means

```
In [8]: data['mean90'] = r90

In [9]: r360 = data['price'].rolling(window='360D').mean()

In [10]: data['mean360'] = r360; data.plot()
```



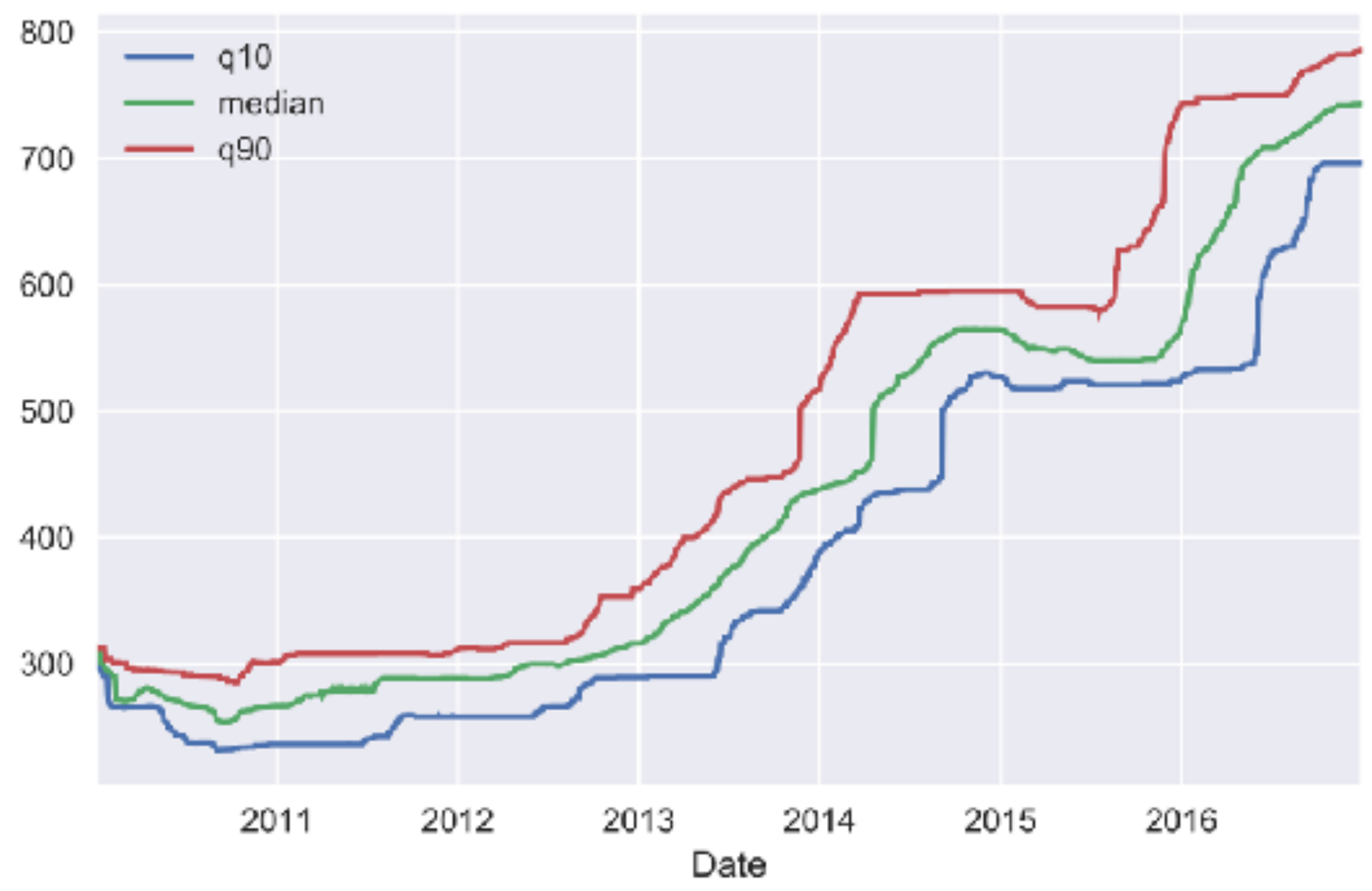
Multiple Rolling Metrics (1)

```
In [8]: r = data.price.rolling('90D').agg(['mean', 'std'])

In [9]: r.plot(subplots = True)
```



Multiple Rolling Metrics (2)

```
In [10]: rolling = data.google.rolling('360D')

In [11]: q10 = rolling.quantile(.1).to_frame('q10')

In [12]: median = rolling.median().to_frame('median')

In [13]: q90 = rolling.quantile(.9).to_frame('q90')

In [14]: pd.concat([q10, median, q90], axis=1).plot()
```



Let's practice!

Expanding Window Functions with Pandas

Expanding Windows in pandas

- From rolling to expanding windows
- Calculate metrics for periods up to current date
- New time series reflects all historical values
- Useful for running rate of return, running min/max
- Two options with pandas:
  - `.expanding()` - just like `.rolling()`
  - `.cumsum()`, `.cumprod()`, `cummin()/max()`

The Basic Idea

```
In [1]: df = pd.DataFrame({'data': range(5)})

In [2]: df['expanding sum'] = df.data.expanding().sum()

In [3]: df['cumulative sum'] = df.data.cumsum()

In [4]: df

   data  expanding sum  cumulative sum
0     0            0.0               0
1     1            1.0               1
2     2            3.0               3
3     3            6.0               6
4     4           10.0              10
```

Get data for the S&P 500

```
In [5]: data = pd.read_csv('sp500.csv', parse_dates=['date'],
                            index_col='date')

DatetimeIndex: 2519 entries, 2007-05-24 to 2017-05-24
Data columns (total 1 columns):
SP500    2519 non-null float64
```

How to calculate a Running Return

- Single period return r: current price over last price minus 1

$$r_t = \frac{P_t}{P_{t-1}} - 1$$

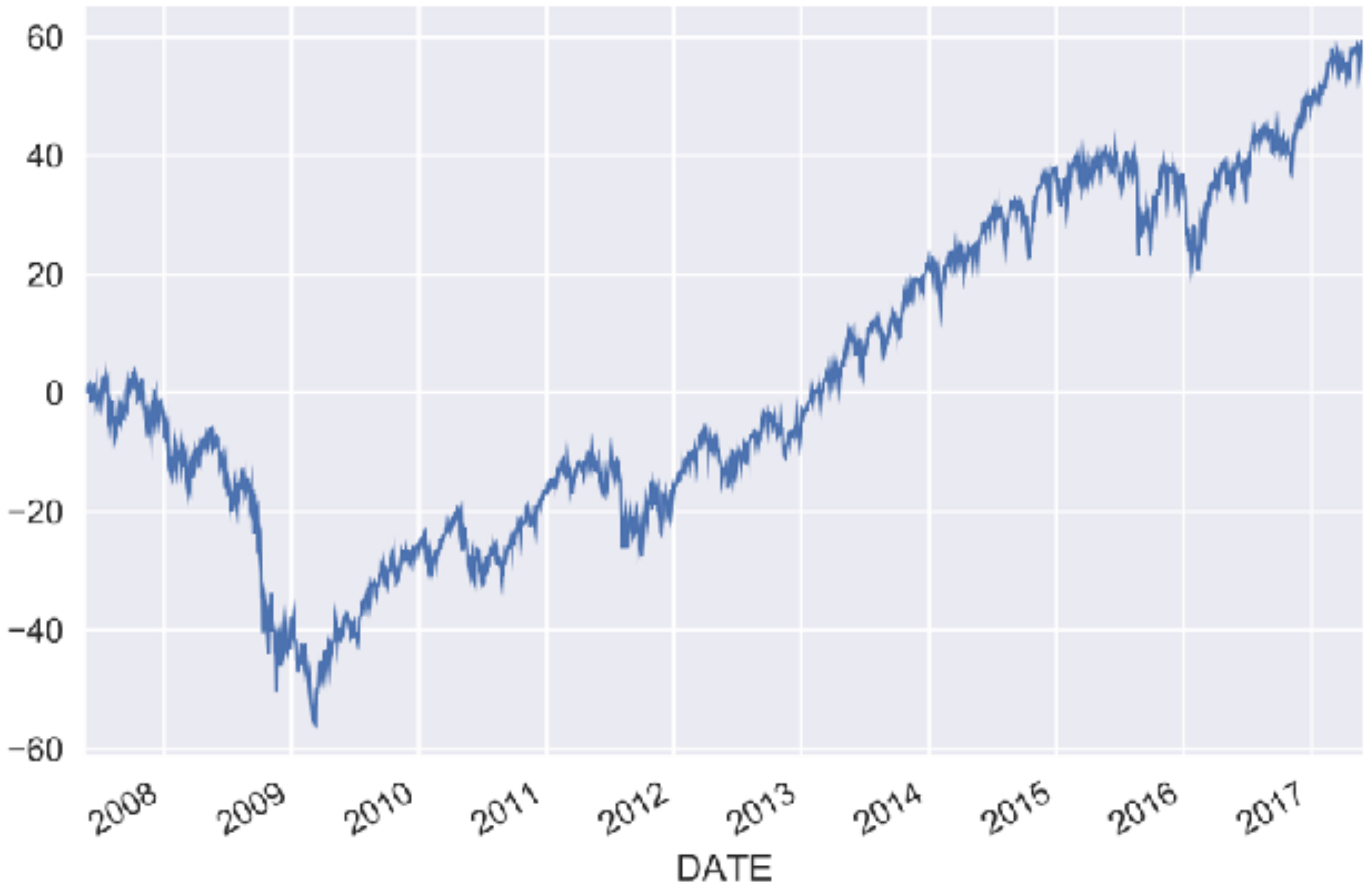
- Multi-period return: product of (1 + r) for all periods, minus 1:

$$R_T = (1 + r_1)(1 + r_2)...(1 + r_T) - 1$$

- For the period return: `.pct_change()`
- For basic math `.add()`, `.sub()`, `.mul()`, `.div()`
- For cumulative product: `.cumprod()`

Running Rate of Return in Practice

```
In [6]: pr = data.SP500.pct_change() # period return

In [7]: pr_plus_one = pr.add(1)

In [8]: cumulative_return = pr_plus_one.cumprod().sub(1)

In [9]: cumulative_return.mul(100).plot()
```



Getting the running min & max

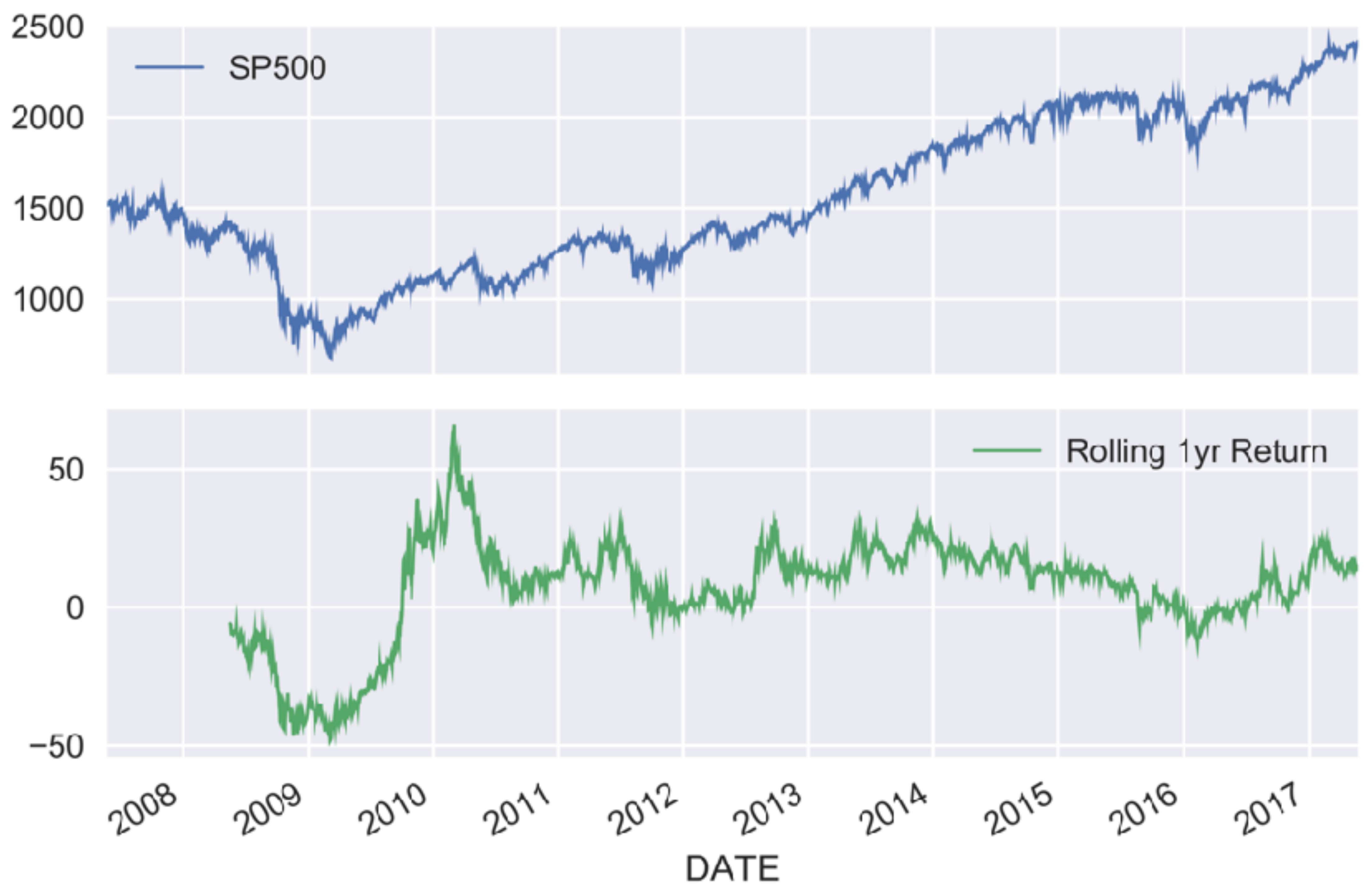
```
In [2]: data['running_min'] = data.SP500.expanding().min()

In [3]: data['running_max'] = data.SP500.expanding().max()

In [4]: data.plot()
```

Rolling Annual Rate of Return

```
In [10]: def multi_period_return(period_returns):
             return np.prod(period_returns + 1) - 1

In [11]: pr = data.SP500.pct_change() # period return

In [12]: r = pr.rolling('360D').apply(multi_period_return)

In [13]: data['Rolling 1yr Return'] = r.mul(100)

In [14]: data.plot(subplots=True)
```

Rolling Annual Rate of Return

```
In [13]: data['Rolling 1yr Return'] = r.mul(100)

In [14]: data.plot(subplots=True)
```



Let's practice!

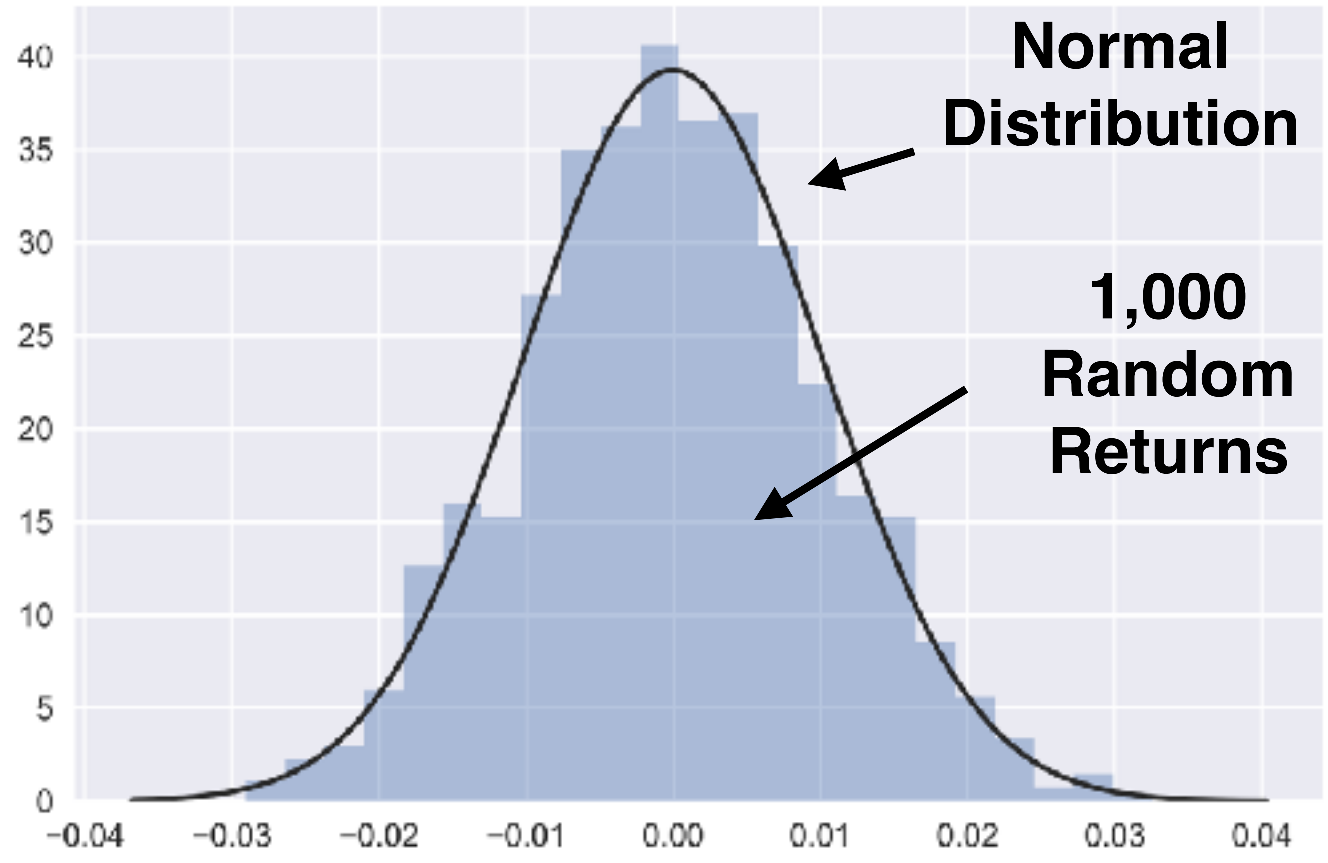
Case Study: S&P500 Price Simulation

Random Walks & Simulations

- Daily stock returns are hard to predict
- Models often assume they are random in nature
- Numpy allows you to generate random numbers
- From random returns to prices: use `.cumprod()`
- Two examples:
  - Generate random returns
  - Randomly selected actual SP500 returns

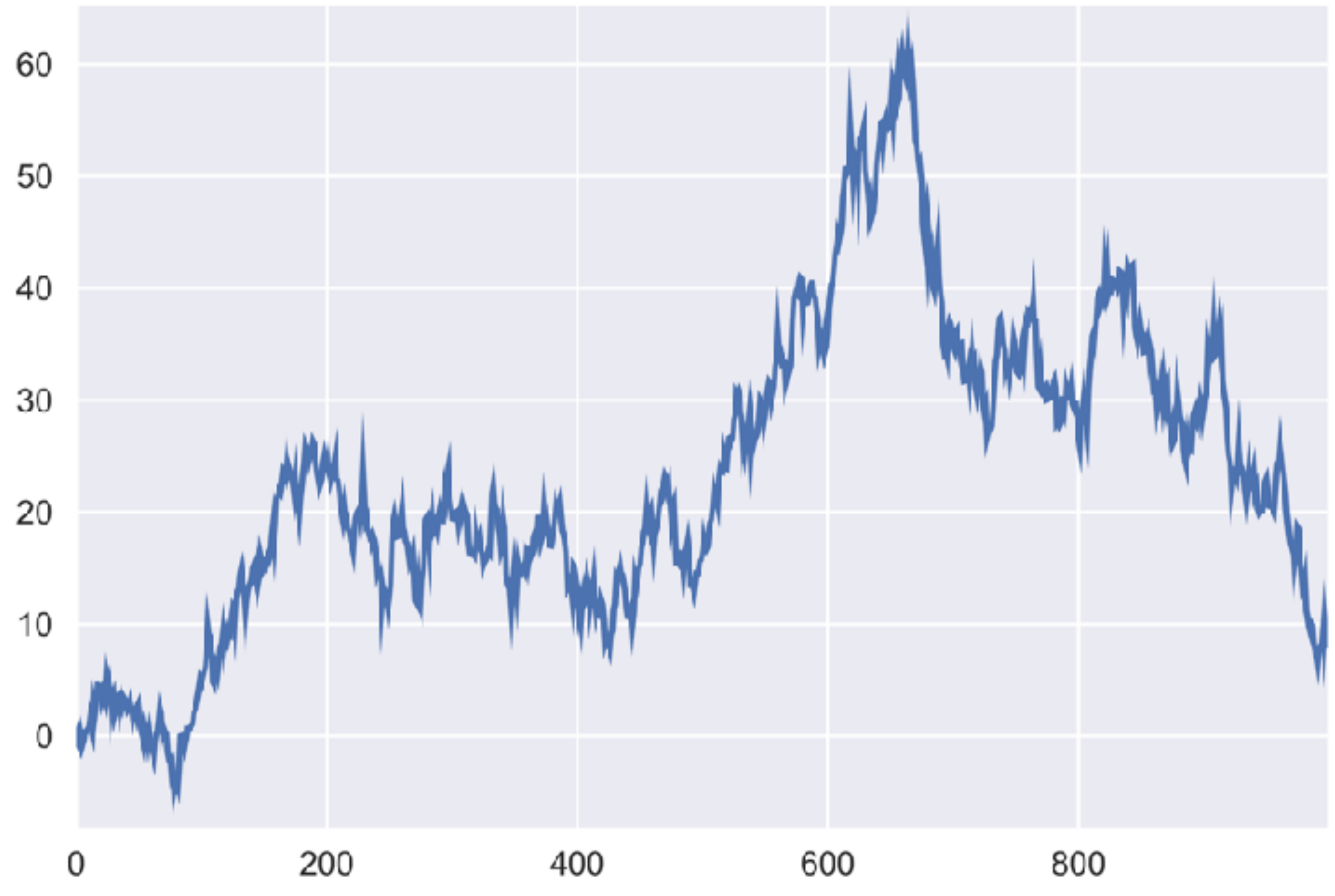
Generate Random Numbers

```
In [1]: from numpy.random import normal, seed

In [2]: from scipy.stats import norm

In [3]: seed(42)

In [3]: random_returns = normal(loc=0, scale=0.01, size=1000)

In [4]: sns.distplot(random_returns, fit=norm, kde=False)
```



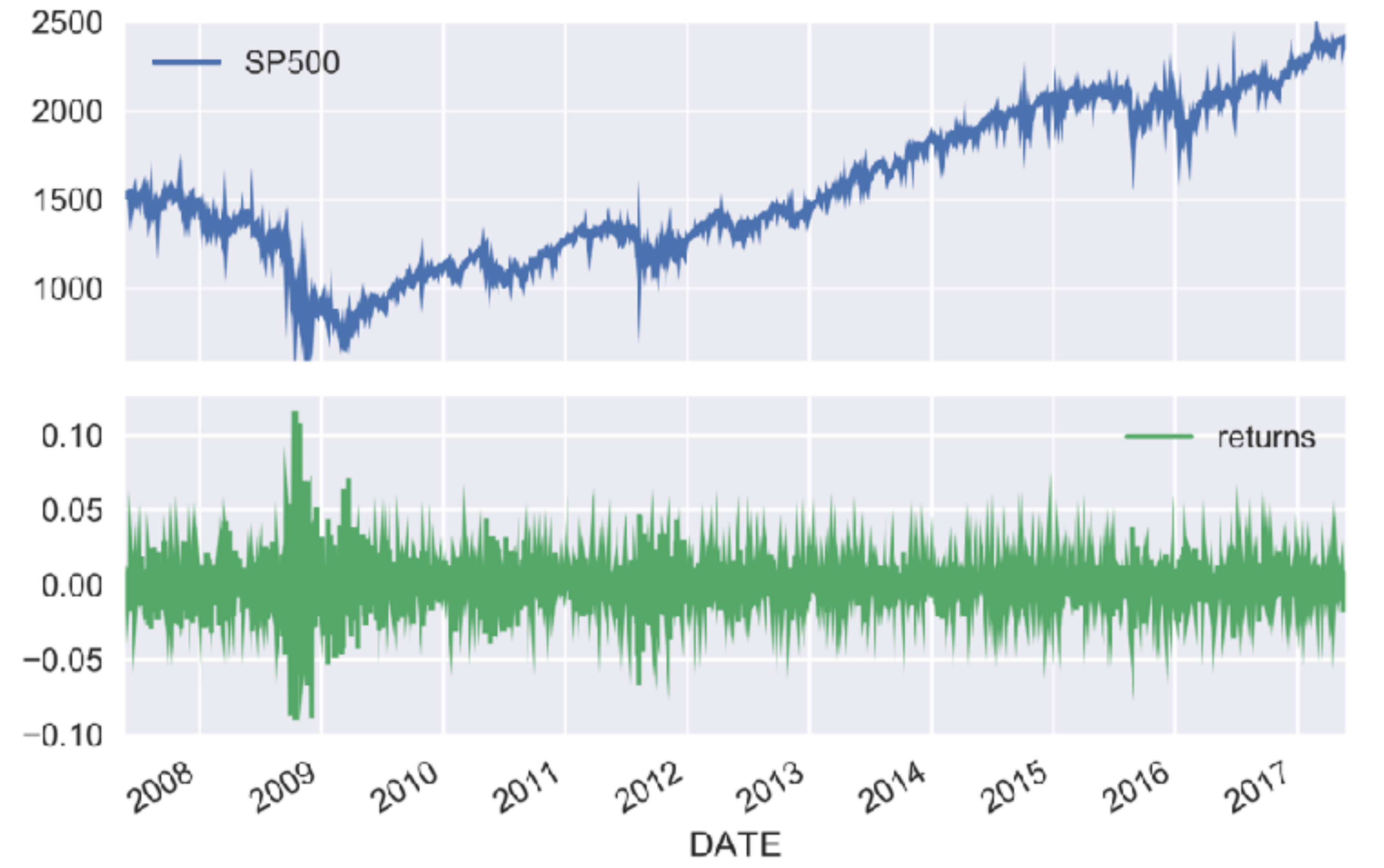
Create A Random Price Path

```
In [5]: return_series = pd.Series(random_returns)

In [6]: random_prices = return_series.add(1).cumprod().sub(1)

In [7]: random_prices.mul(100).plot()
```



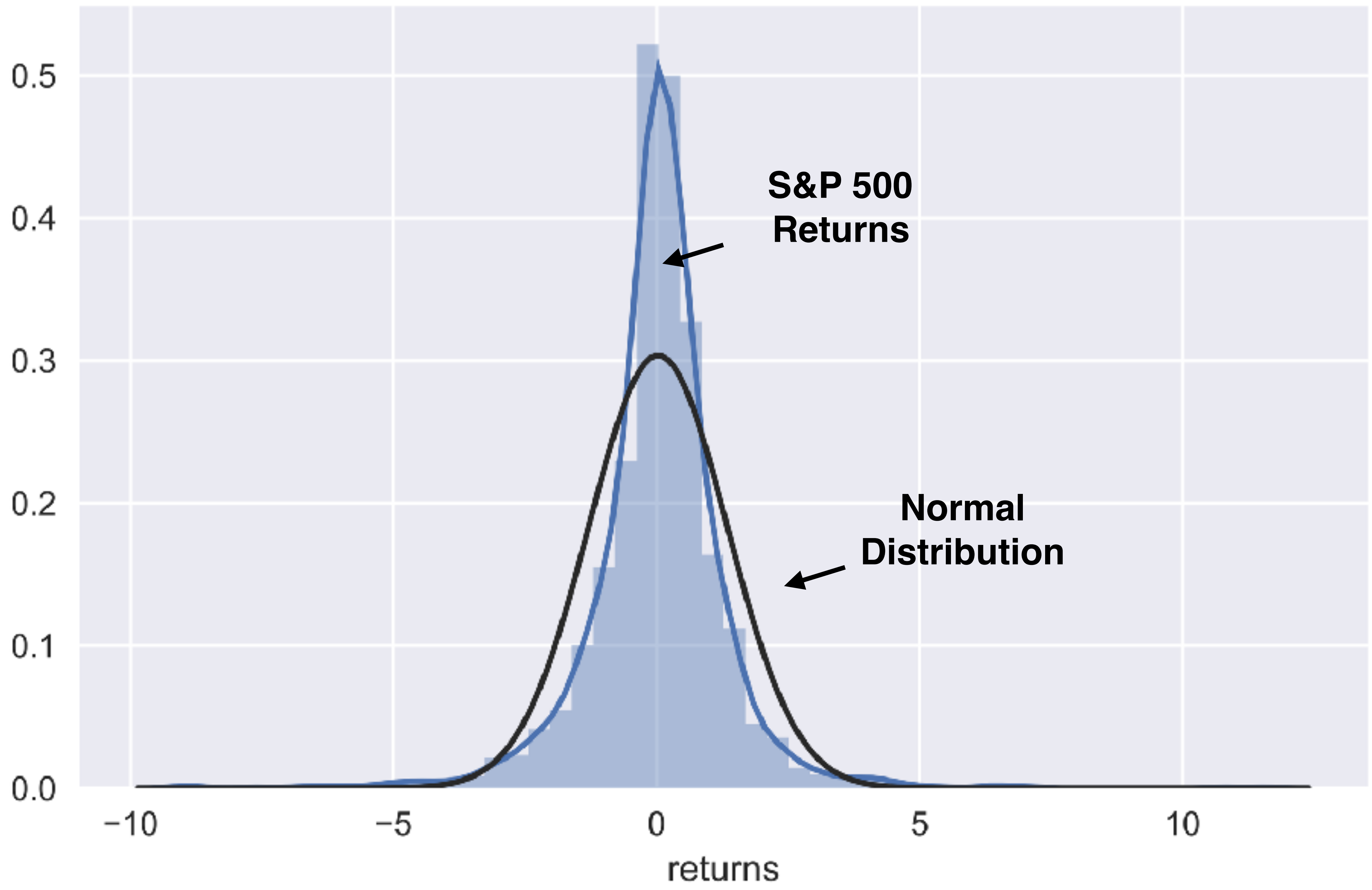
S&P 500 Prices & Returns

```
In [5]: data = pd.read_csv('sp500.csv', parse_dates=['date'],
                           index_col='date')

In [6]: data['returns'] = data.SP500.pct_change()

In [7]: data.plot(subplots=True)
```



S&P Return Distribution

```
In [8]: sns.distplot(data.returns.dropna().mul(100), fit=norm)
```



Generate Random S&P 500 Returns

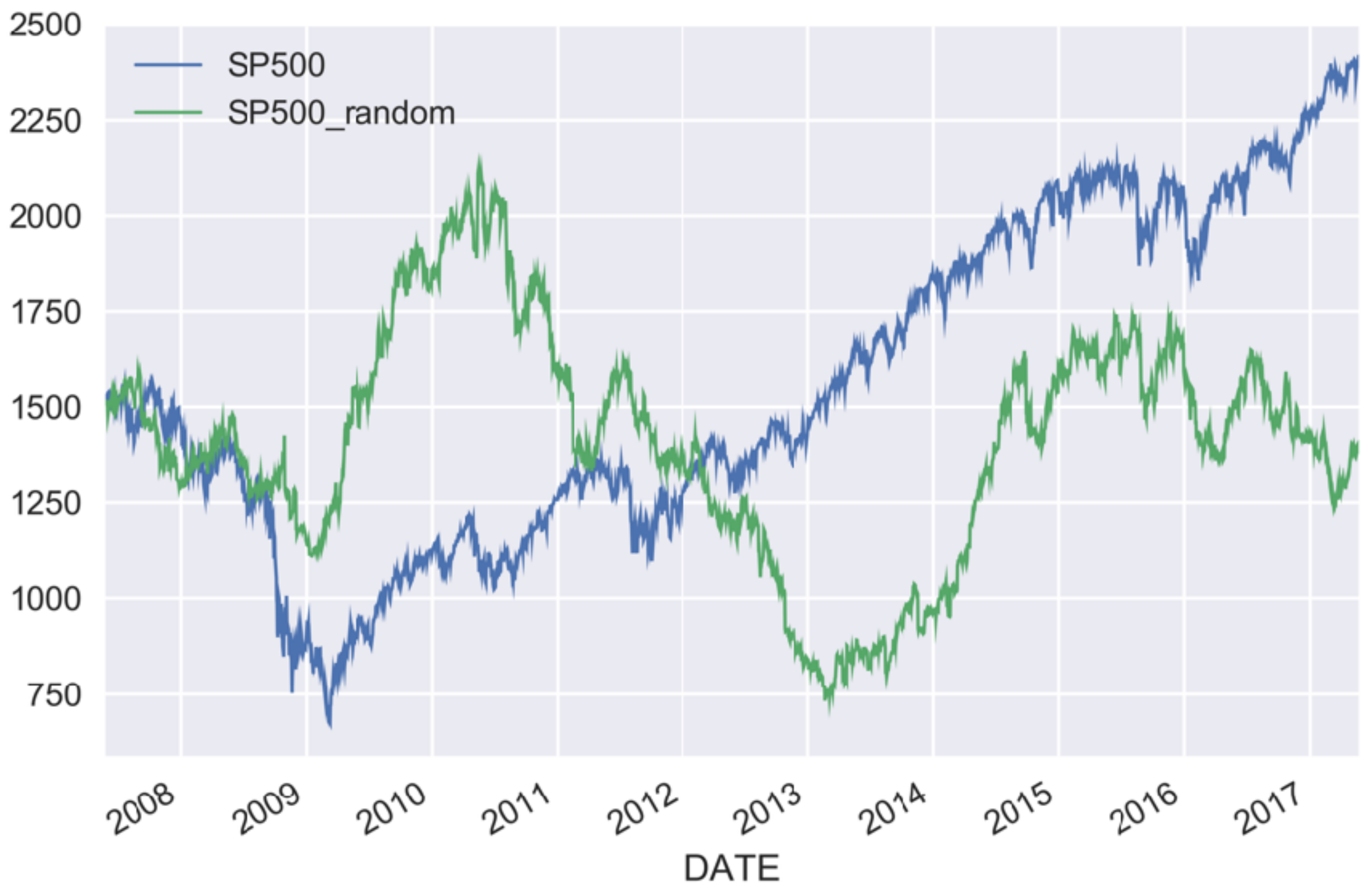
```
In [9]: from numpy.random import choice

In [10]: sample = data.returns.dropna()

In [11]: n_obs = data.returns.count()

In [12]: random_walk = choice(sample, size=n_obs)

In [14]: random_walk = pd.Series(random_walk, index=sample.index)

In [15]: random_walk.head()

DATE
2007-05-29   -0.008357
2007-05-30    0.003702
2007-05-31   -0.013990
2007-06-01    0.008096
2007-06-04    0.013120
```

Random S&P 500 Prices (1)

```
In [9]: start = data.SP500.first('D')

DATE
2007-05-25    1515.73
Name: SP500, dtype: float64

In [10]: sp500_random = start.append(random_walk.add(1))

In [11]: sp500_random.head())

DATE
2007-05-25    1515.730000
2007-05-29       0.998290
2007-05-30       0.995190
2007-05-31       0.997787
2007-06-01       0.983853
dtype: float64
```

Random S&P 500 Prices (2)

```
In [9]: data['SP500_random'] = sp500_random.cumprod()

In [10]: data[['SP500', 'SP500_random']].plot()
```



Let's practice!

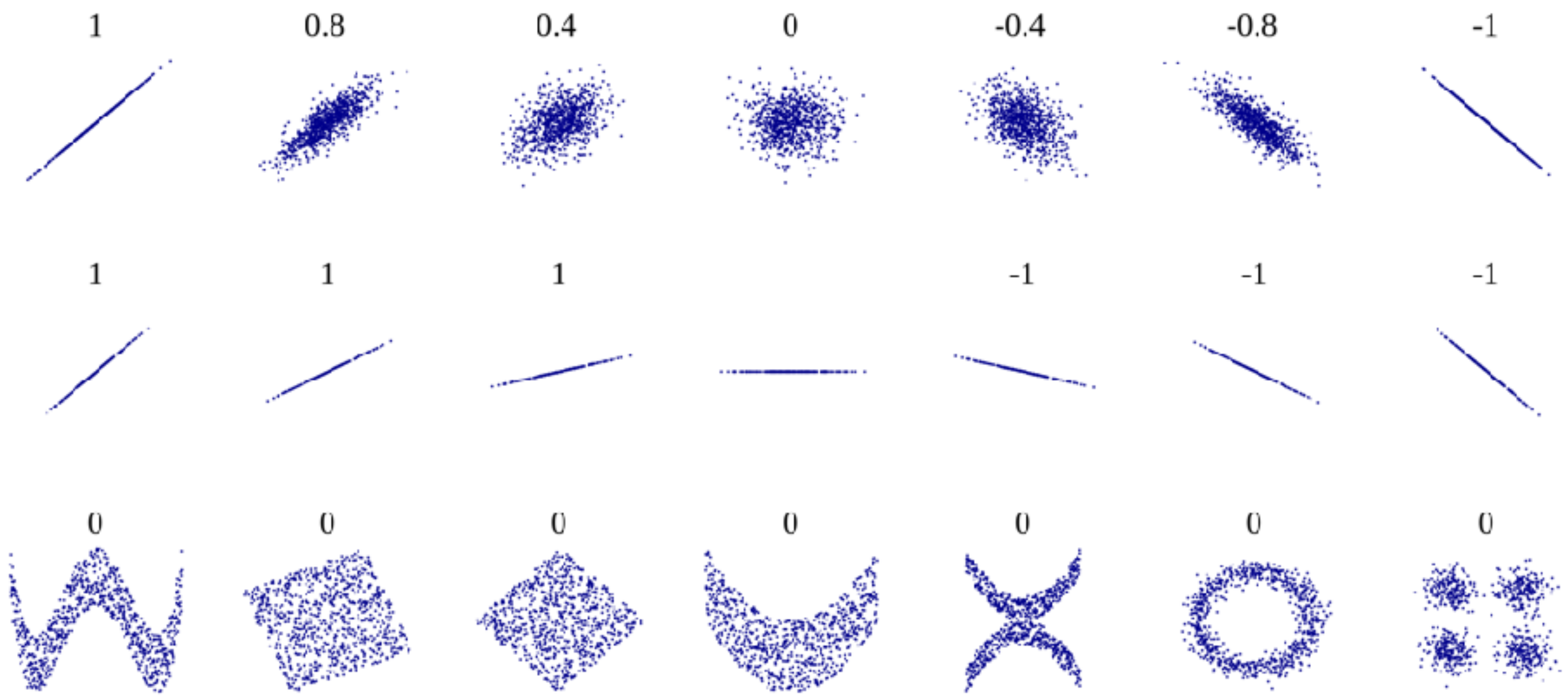
Relationships between Time Series: Correlation

Correlation & Relations between Series

- So far, focus on characteristics of individual variables
- Now: characteristic of relations between variables
- Correlation: measures linear relationships
- Financial markets: important for prediction and risk management
- Pandas & seaborns have tools to compute & visualize

Correlation & Linear Relationships

- Correlation coefficient: how similar is the pairwise movement of two variables around their averages?
- Varies between -1 and + 1 $r = \frac{\sum_{i=i}^{N}(x_i - \bar{x})(y_i - \bar{y})}{s_x s_y}$



Strength of linear relationship

Positive or negative

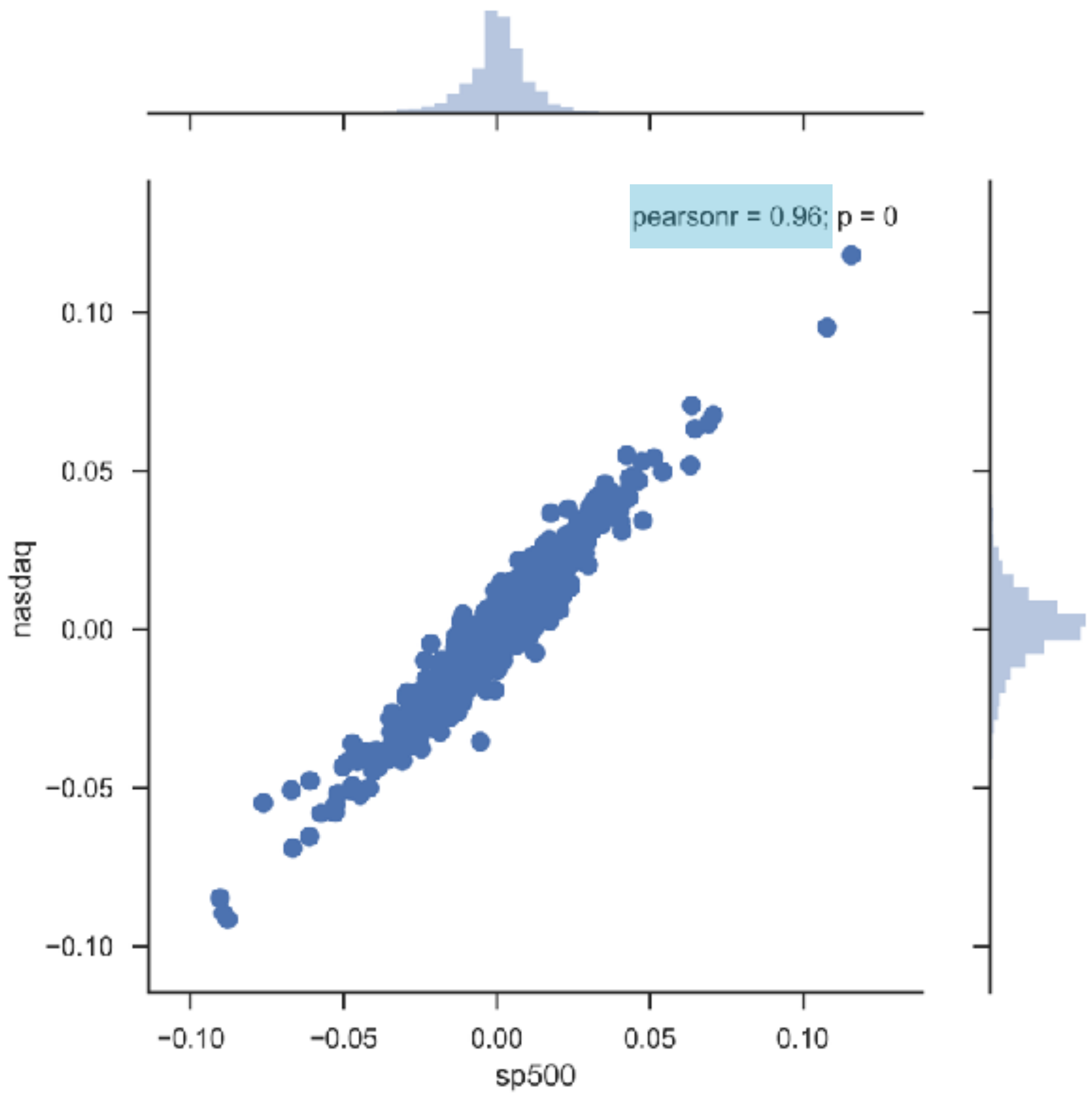
Not: non-linear relationships

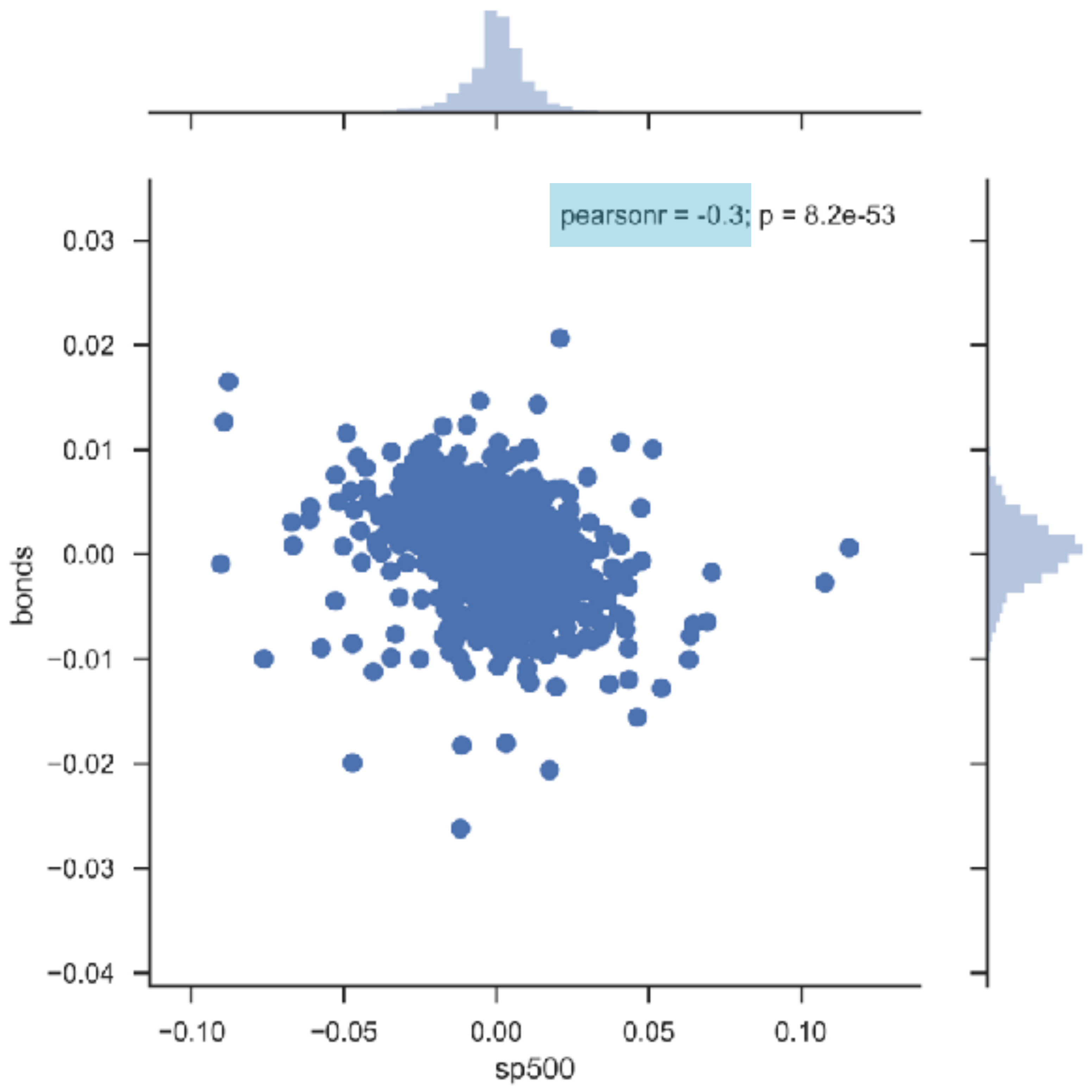
Importing Five Price Time Series

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

In [2]: data = data.dropna().info()

DatetimeIndex: 2469 entries, 2007-05-25 to 2017-05-22
Data columns (total 5 columns):
sp500     2469 non-null float64
nasdaq    2469 non-null float64
bonds     2469 non-null float64
gold      2469 non-null float64
oil       2469 non-null float64
```

Visualize pairwise linear relationships

```
In [4]: daily_returns = data.pct_change()

In [5]: sns.jointplot(x='sp500', y='nasdaq', data=data_returns);
```



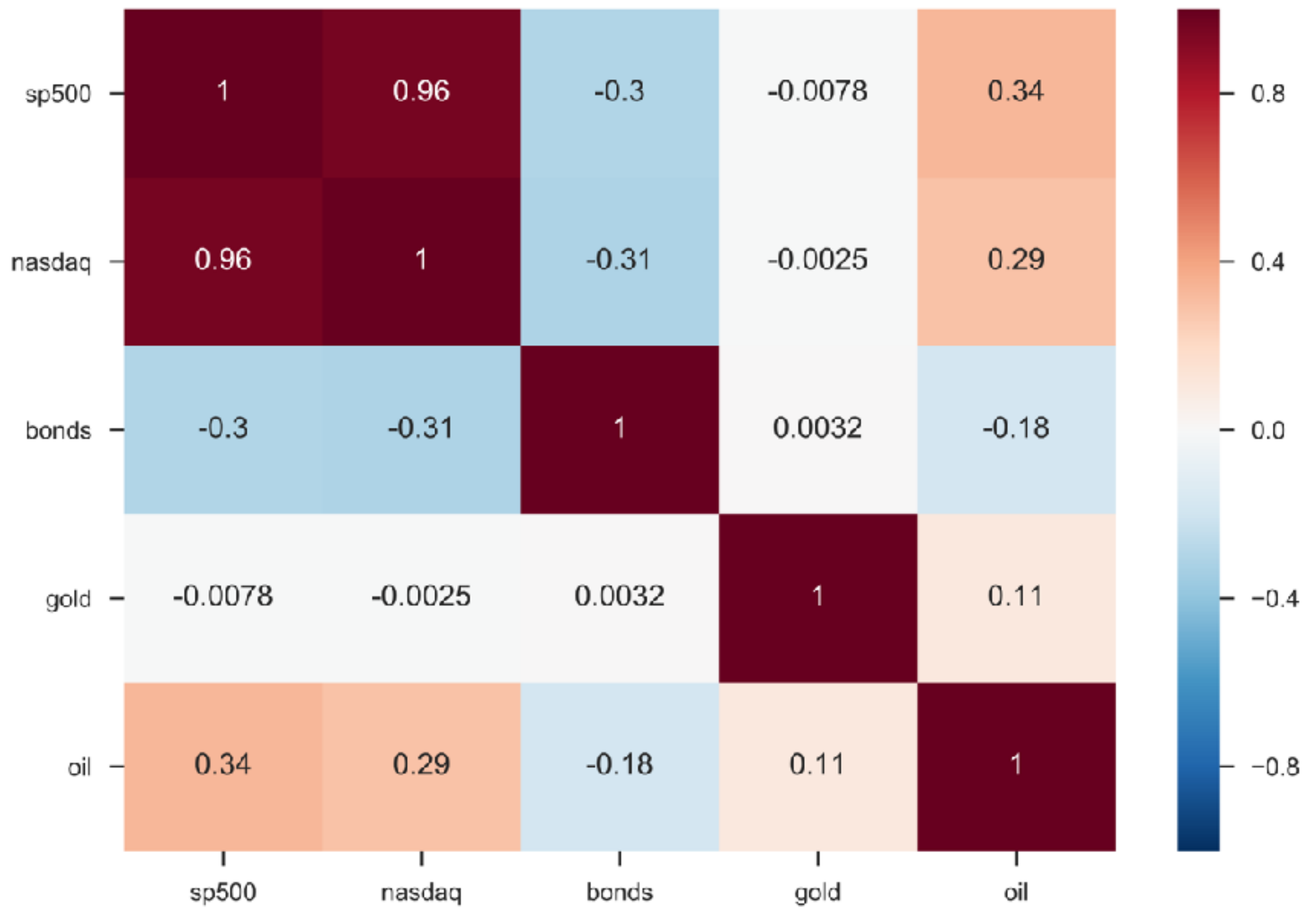


Calculate all Correlations

```
In [6]: correlations = returns.corr()

In [7]: correlations
Out[7]:
           bonds       oil      gold     sp500    nasdaq
bonds   1.000000 -0.183755  0.003167 -0.300877 -0.306437
oil    -0.183755  1.000000  0.105930  0.335578  0.289590
gold    0.003167  0.105930  1.000000 -0.007786 -0.002544
sp500  -0.300877  0.335578 -0.007786  1.000000  0.959990
nasdaq -0.306437  0.289590 -0.002544  0.959990  1.000000
```

Visualize all Correlations

```
In [8]: sns.heatmap(correlations, annot=True)
```



Let's practice!