# Rolling window functions with pandas

MANIPULATING TIME SERIES DATA IN PYTHON

**Stefan Jansen**

Founder & Lead Data Scientist at Applied Artificial Intelligence

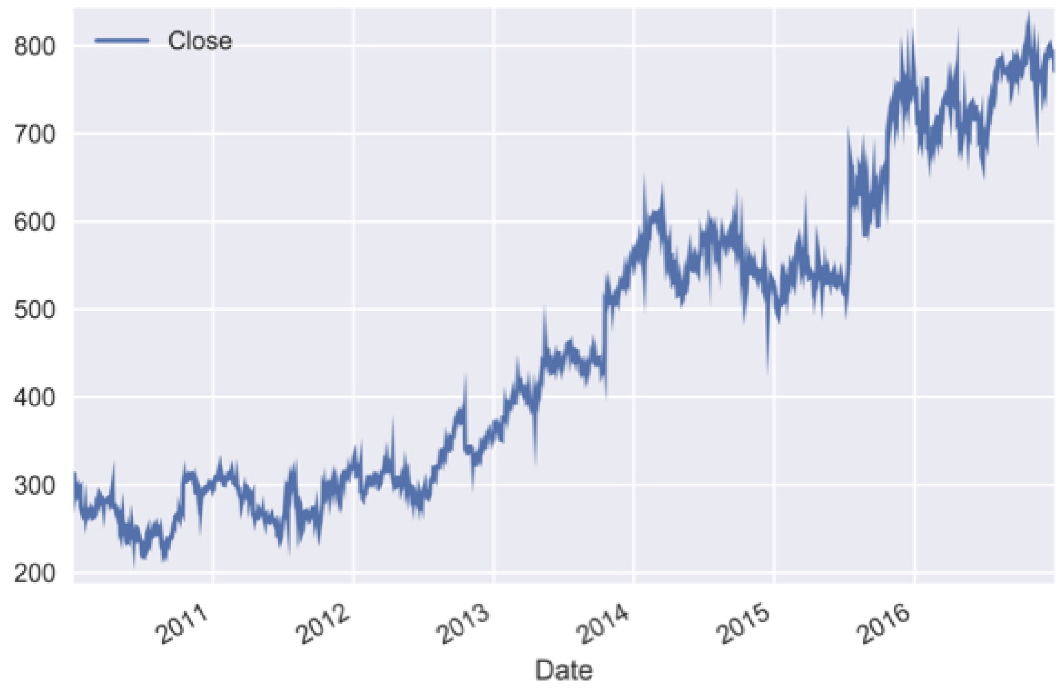
# Window functions in pandas

- 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

```
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
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

When you choose an integer-based window size, pandas will only calculate the mean if the window has no missing values.
You can change this default by setting the min_periods parameter to a value smaller than the window size of 30.

# Calculating a rolling average

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

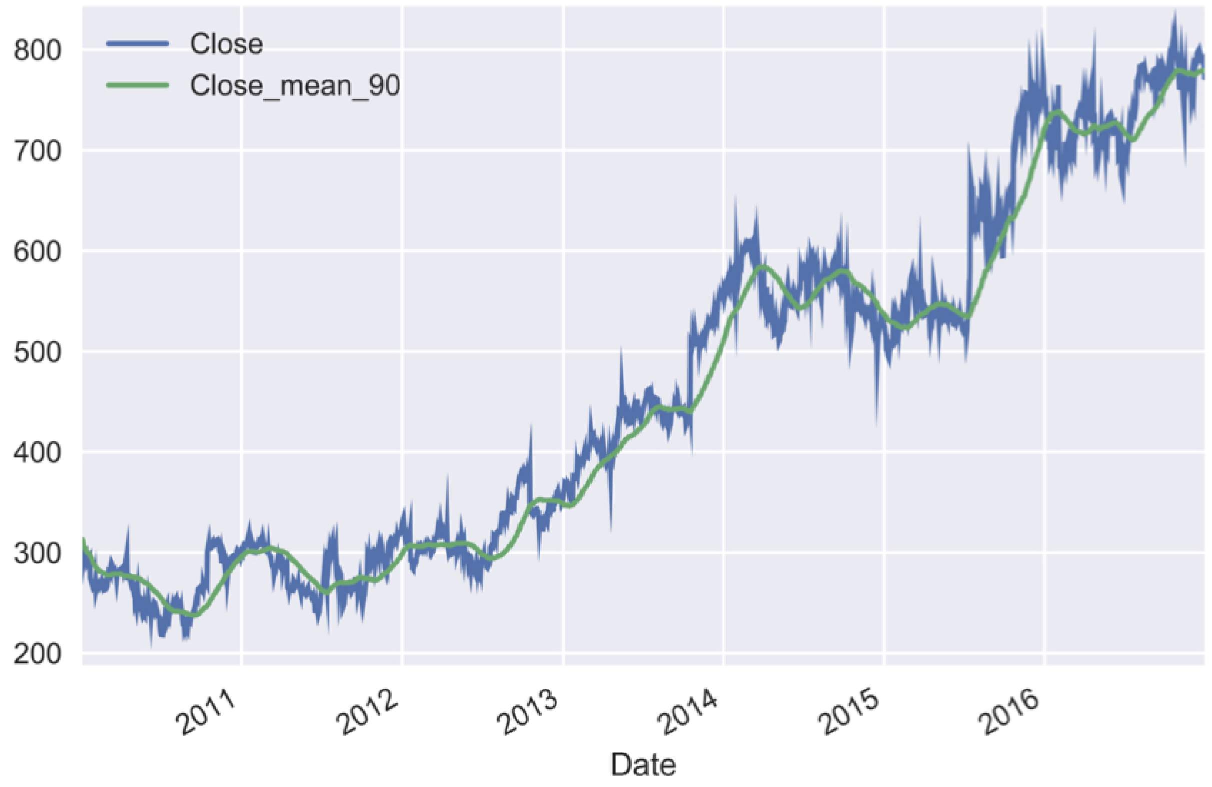
```
DatetimeIndex: 1761 entries, 2010-01-04 to 2017-05-24
Data columns (total 1 columns):
price    1761 non-null float64
dtypes: float64(1)
```

- `30D` : # calendar days

The window will contain the days when stocks were traded during the last 30 calendar days.
While the window is fixed in terms of period length, the number of observations will vary.

# 90 day rolling mean

```
r90 = data.rolling(window='90D').mean()

google.join(r90.add_suffix('_mean_90')).plot()
```




.join: **concatenate** Series **or** DataFrame **along** axis=1

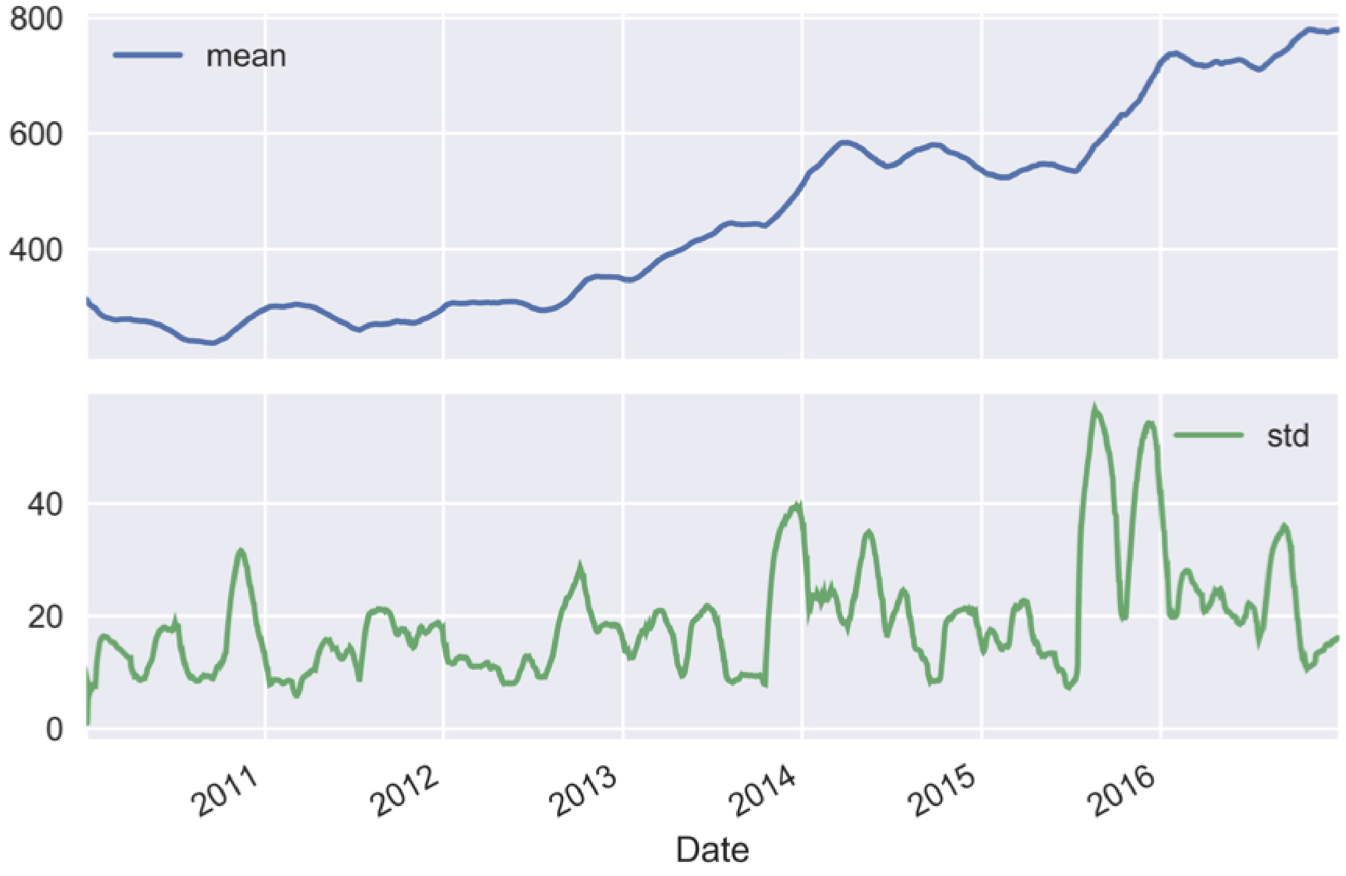
The new time series is much smoother because every data point is now the average of the preceding 90 calendar days.

# 90 & 360 day rolling means

```
data['mean90'] = r90
r360 = data['price'].rolling(window='360D'.mean()
data['mean360'] = r360; data.plot()
```

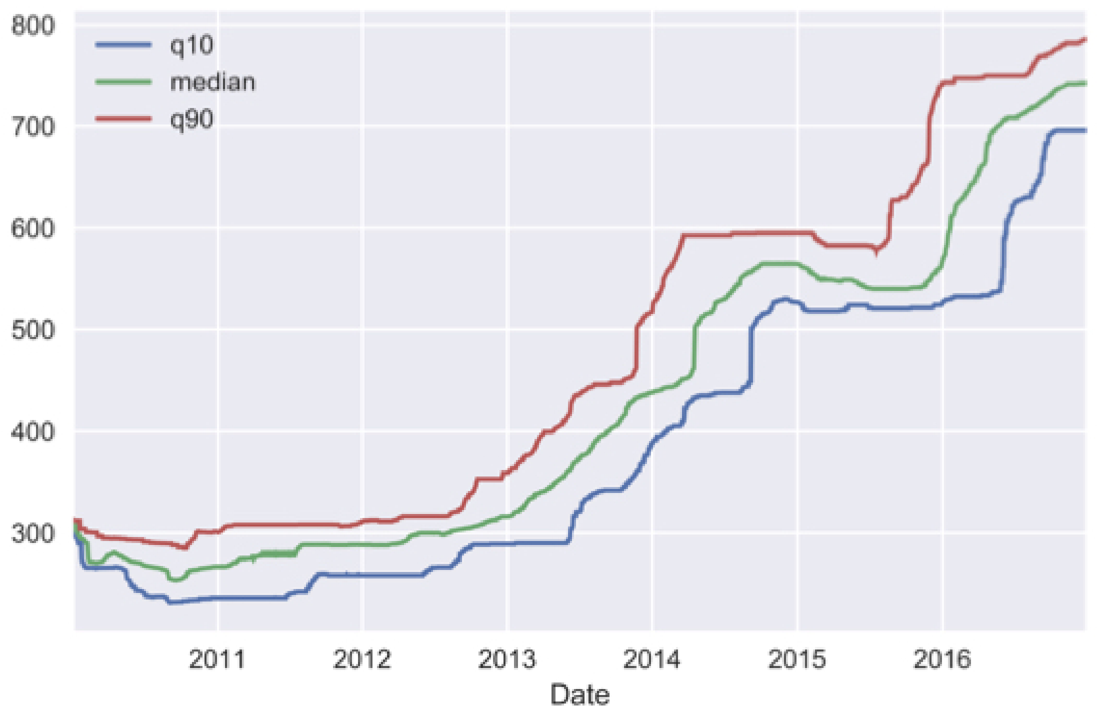
# Multiple rolling metrics (1)

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

r.plot(subplots = True)
```



# Multiple rolling metrics (2)

```
rolling = data.google.rolling('360D')
q10 = rolling.quantile(0.1).to_frame('q10')
median = rolling.median().to_frame('median')
q90 = rolling.quantile(0.9).to_frame('q90')
pd.concat([q10, median, q90], axis=1).plot()
```



# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

# Expanding window functions with pandas

MANIPULATING TIME SERIES DATA IN PYTHON

**Stefan Jansen**

Founder & Lead Data Scientist at Applied Artificial Intelligence

# 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

```
df = pd.DataFrame({'data': range(5)})
df['expanding sum'] = df.data.expanding().sum()
df['cumulative sum'] = df.data.cumsum()
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

```
data = pd.read_csv('sp500.csv', parse_dates=['date'], index_col='dat
```

```
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 $r_t$: current price over last price minus 1:

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

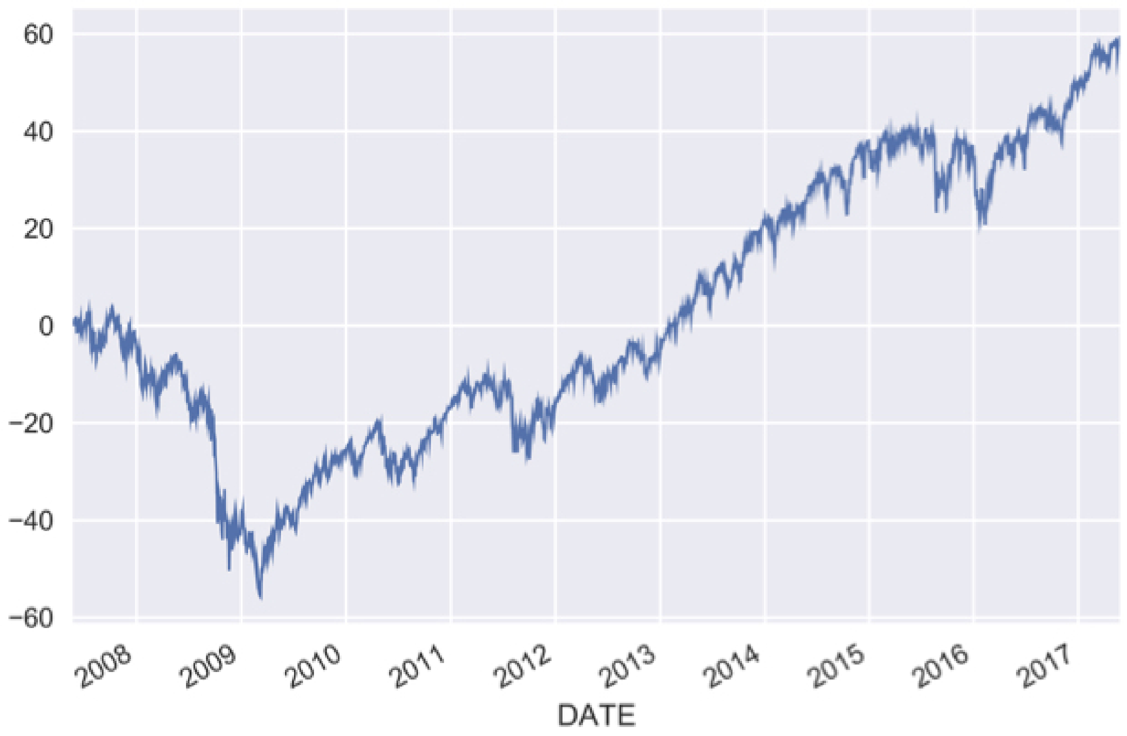
  - Multi-period return: product of $(1 + r_t)$ for all periods, minus 1:

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

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

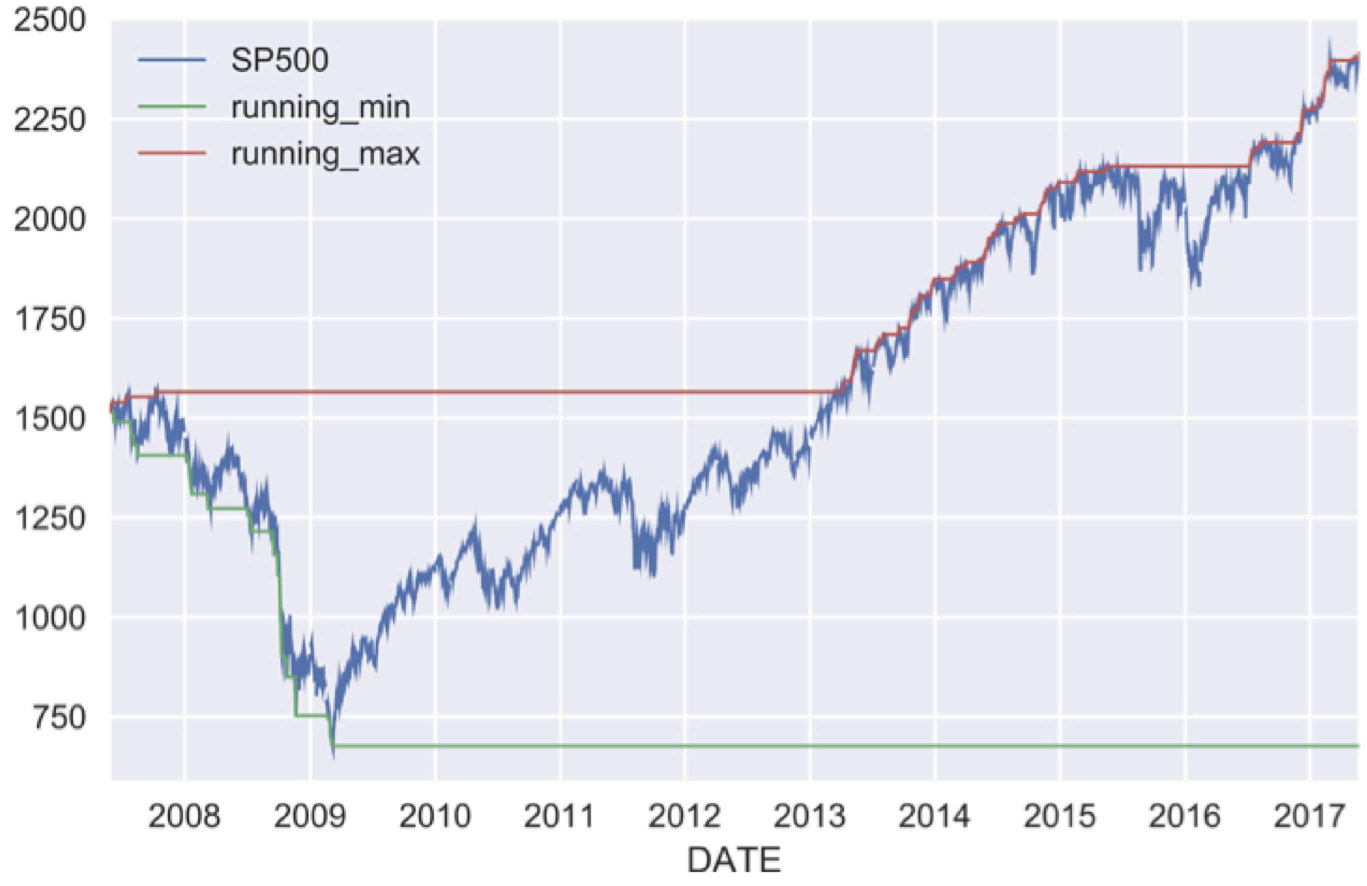
# Running rate of return in practice

```
pr = data.SP500.pct_change() # period return
pr_plus_one = pr.add(1)
cumulative_return = pr_plus_one.cumprod().sub(1)
cumulative_return.mul(100).plot()
```



# Getting the running min & max

```
data['running_min'] = data.SP500.expanding().min()
data['running_max'] = data.SP500.expanding().max()
data.plot()
```

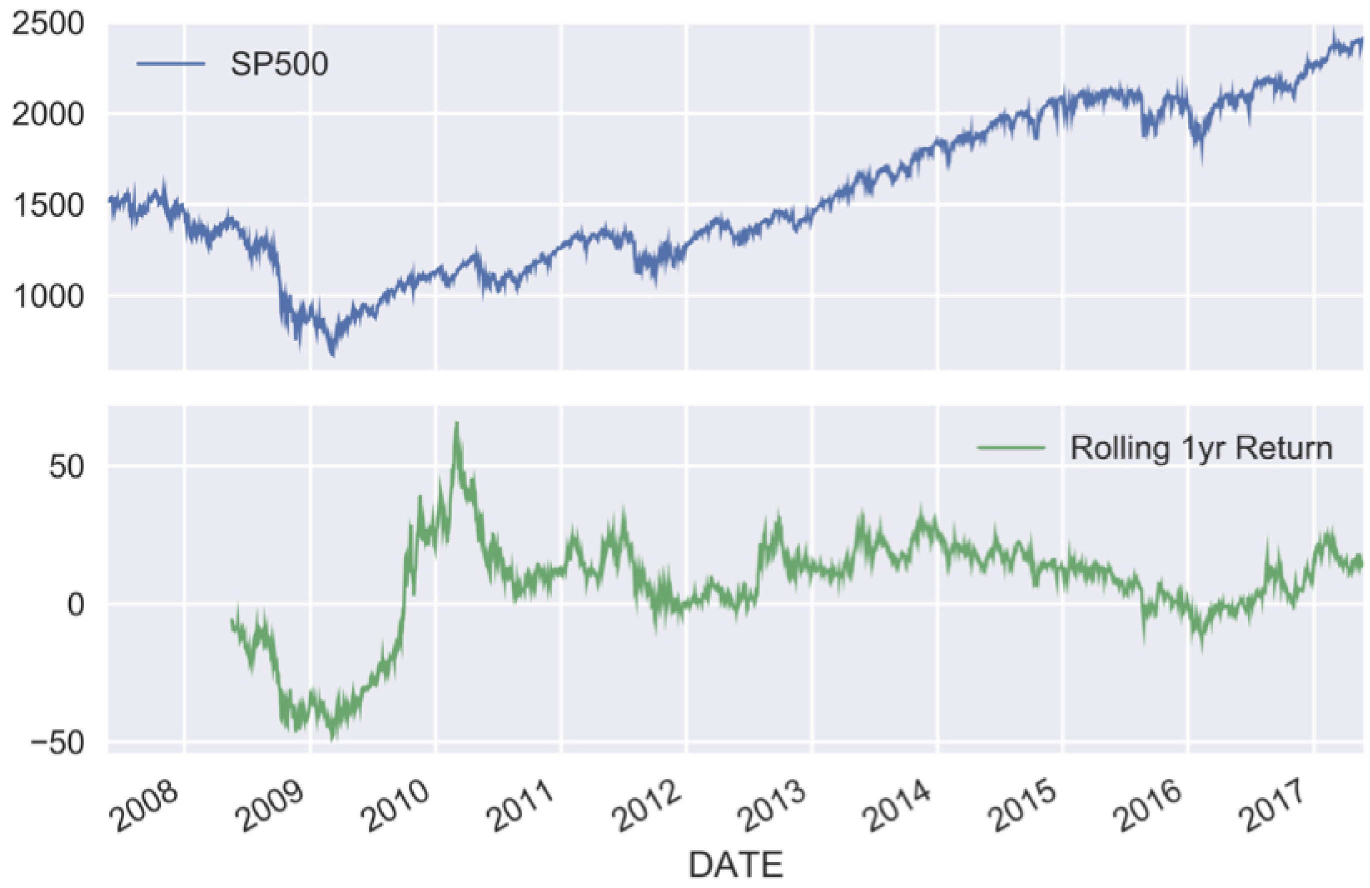


# Rolling annual rate of return

```
def multi_period_return(period_returns):
    return np.prod(period_returns + 1) - 1
pr = data.SP500.pct_change() # period return
r = pr.rolling('360D').apply(multi_period_return)
data['Rolling 1yr Return'] = r.mul(100)
data.plot(subplots=True)
```

# Rolling annual rate of return

```
data['Rolling 1yr Return'] = r.mul(100)
data.plot(subplots=True)
```



# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

# Case study: S&P500 price simulation

MANIPULATING TIME SERIES DATA IN PYTHON

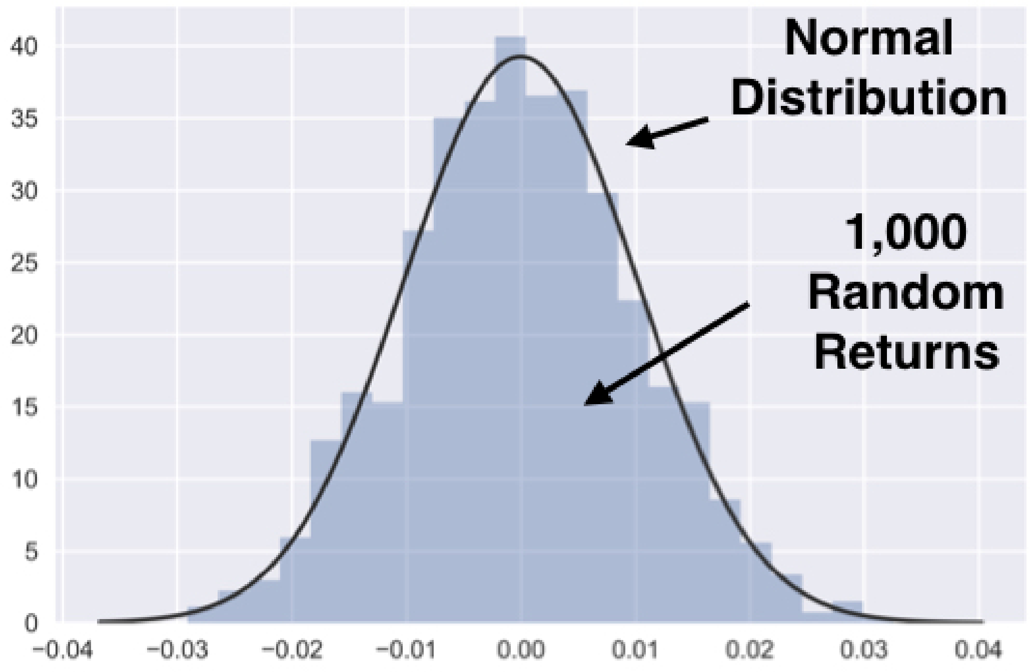
**Stefan Jansen**
Founder & Lead Data Scientist at Applied Artificial Intelligence

# 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

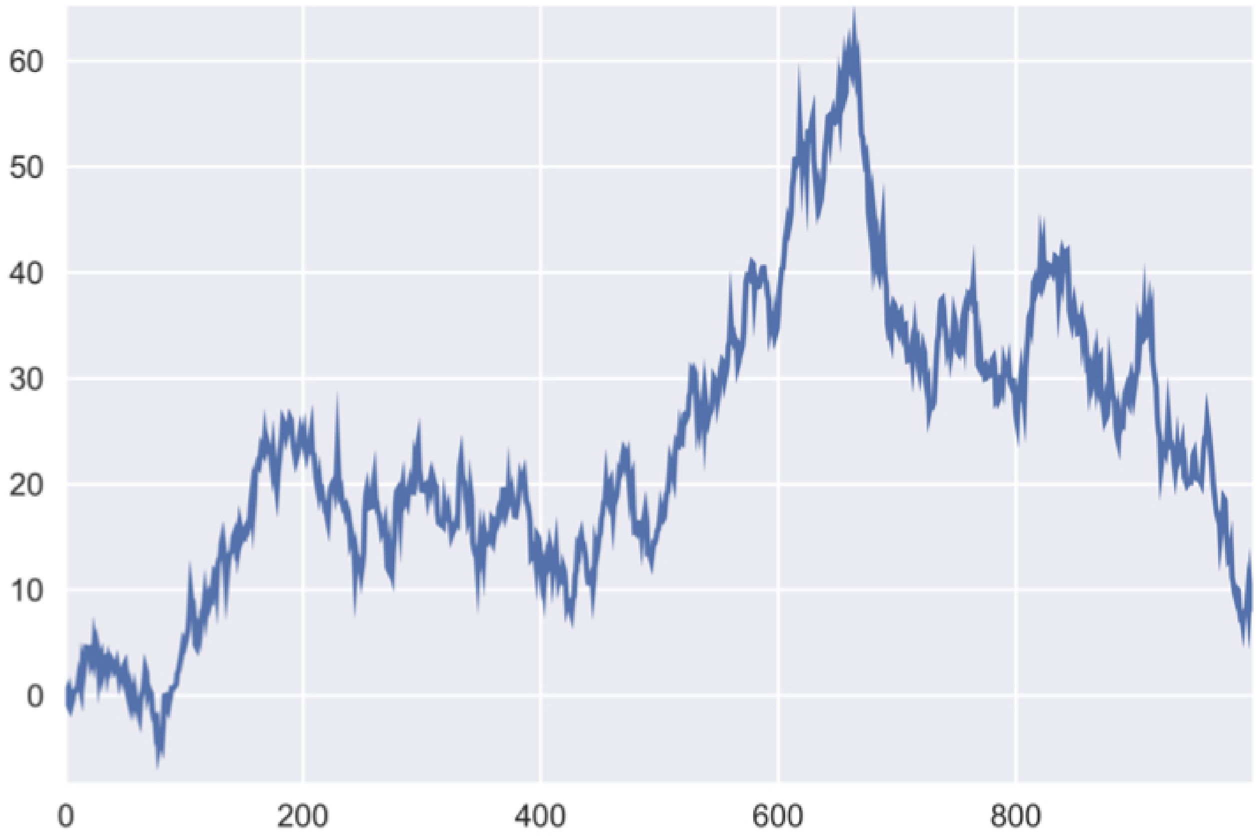
# Generate random numbers

```
from numpy.random import normal, seed
from scipy.stats import norm
seed(42)
random_returns = normal(loc=0, scale=0.01, size=1000)
sns.distplot(random_returns, fit=norm, kde=False)
```



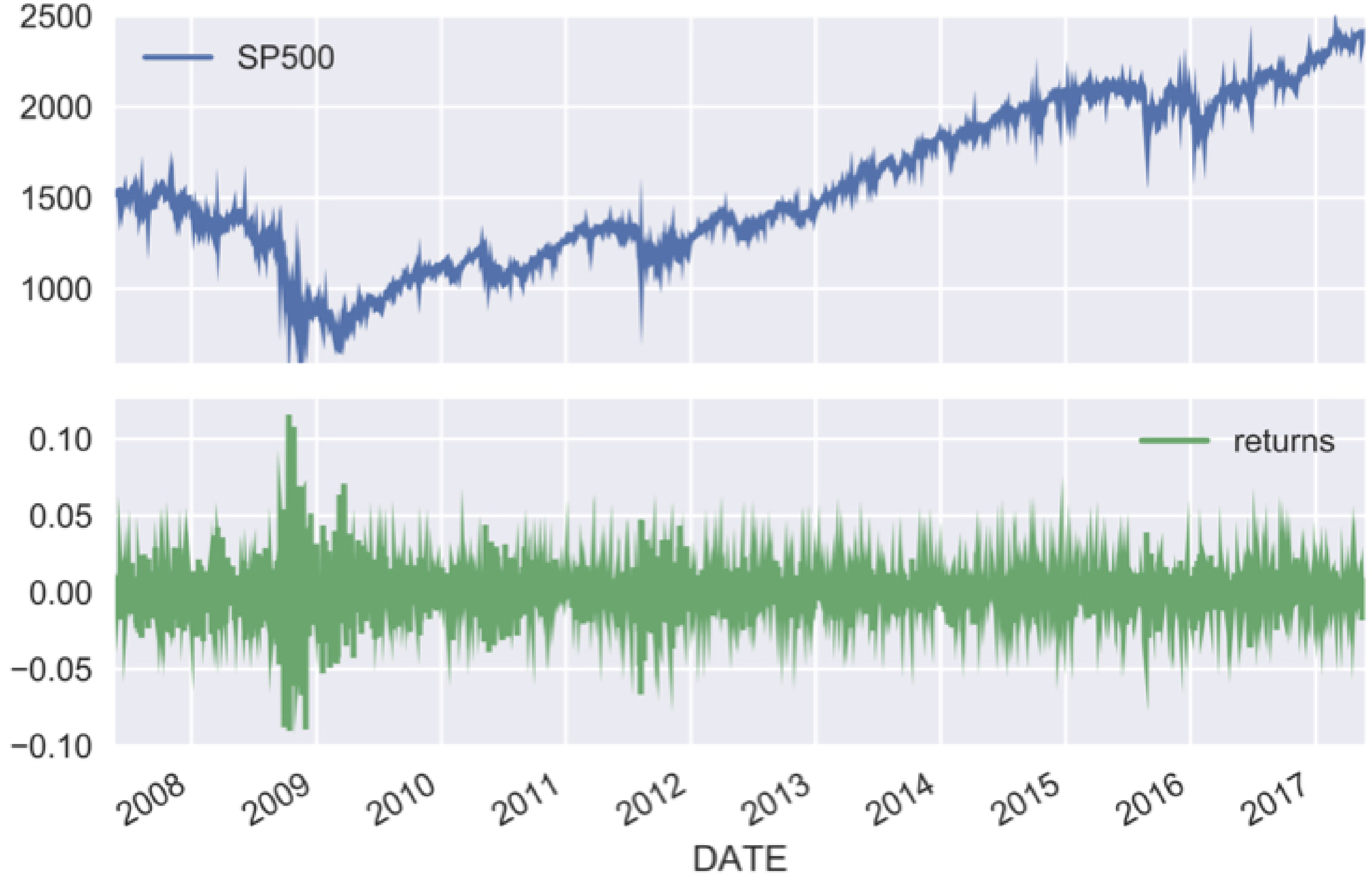
# Create a random price path

```
return_series = pd.Series(random_returns)
random_prices = return_series.add(1).cumprod().sub(1)
random_prices.mul(100).plot()
```



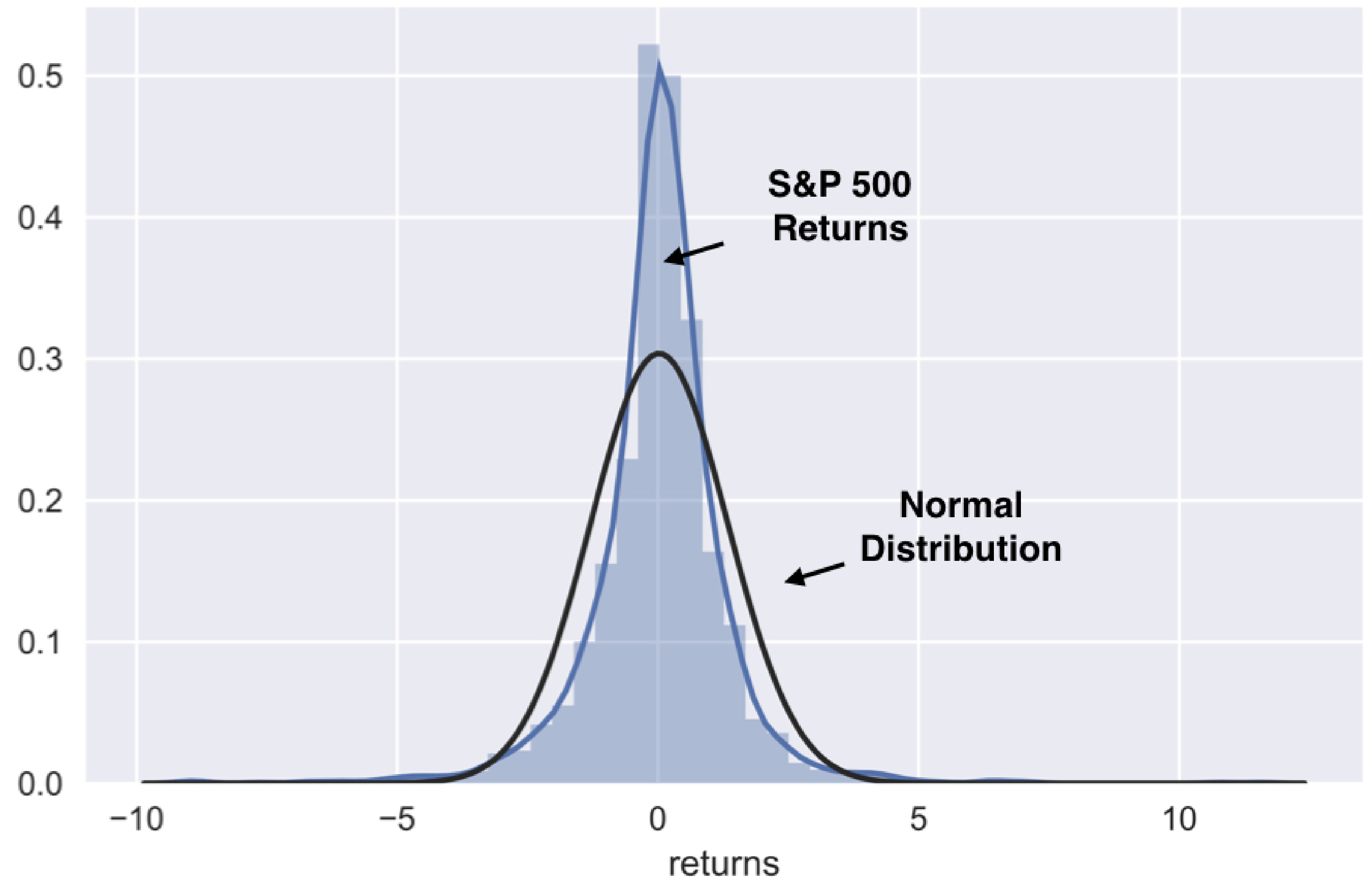
# S&P 500 prices & returns

```
data = pd.read_csv('sp500.csv', parse_dates=['date'], index_col='date')
data['returns'] = data.SP500.pct_change()
data.plot(subplots=True)
```



# S&P return distribution

```
sns.distplot(data.returns.dropna().mul(100), fit=norm)
```



# Generate random S&P 500 returns

```python
from numpy.random import choice
sample = data.returns.dropna()
n_obs = data.returns.count()
random_walk = choice(sample, size=n_obs)
random_walk = pd.Series(random_walk, index=sample.index)
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)

```
start = data.SP500.first('D')
```

Use the 'first' method with calendar day offset to select the first S&P 500 price.

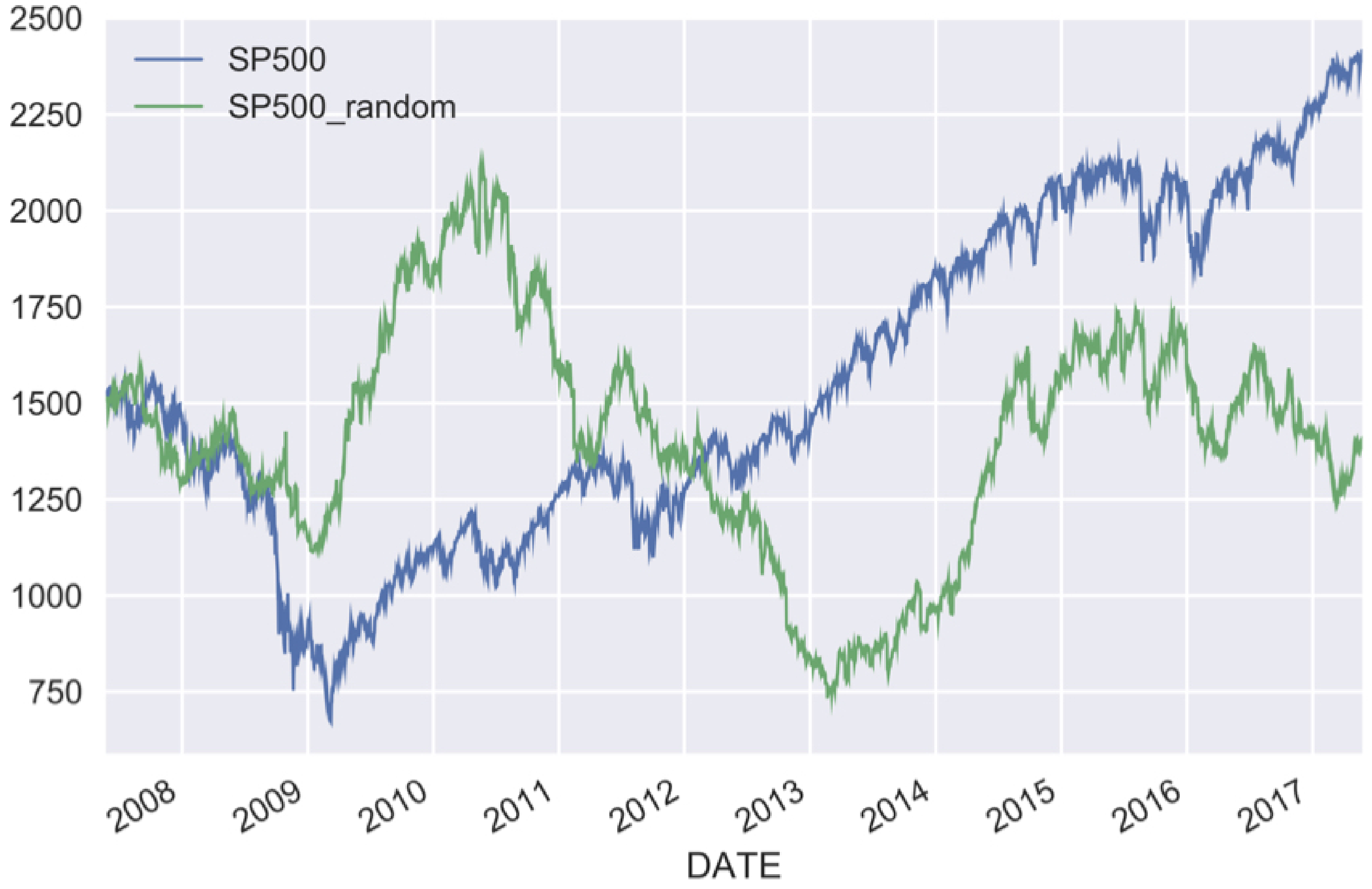
```
DATE
2007-05-25    1515.73
Name: SP500, dtype: float64
```

```
sp500_random = start.append(random_walk.add(1))
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)

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

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



# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

# Relationships between time series: correlation

MANIPULATING TIME SERIES DATA IN PYTHON

**Stefan Jansen**
Founder & Lead Data Scientist at Applied Artificial Intelligence

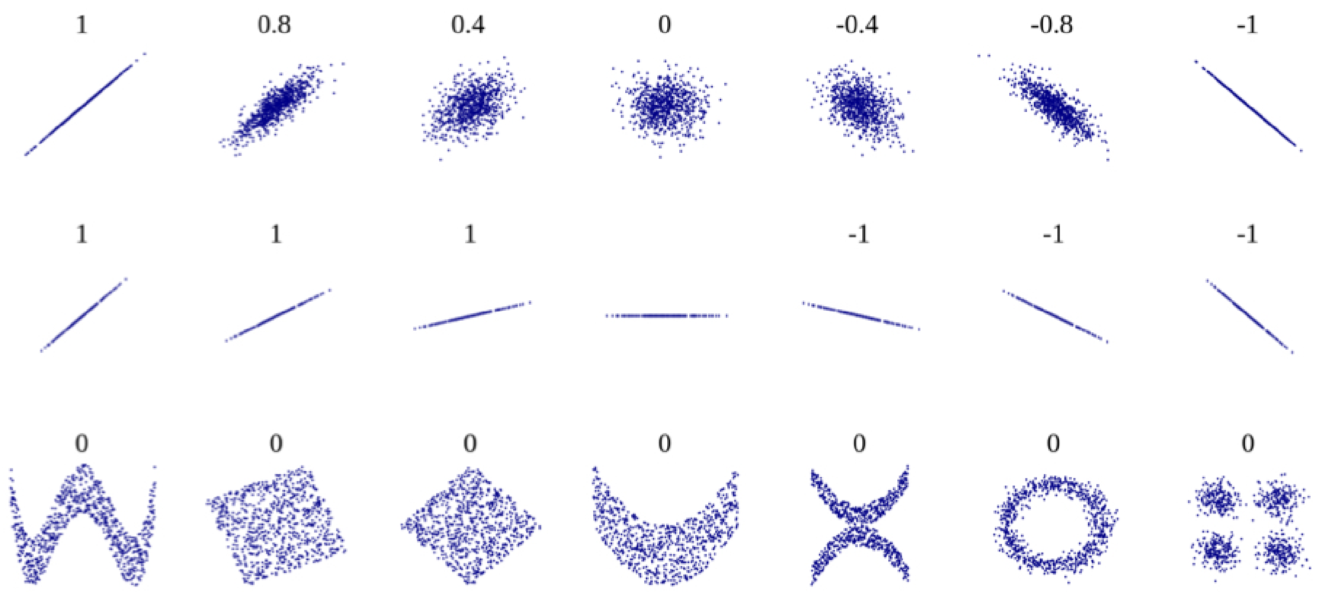
# Correlation & relations between series

- 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` & `seaborn` 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`

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{s_x s_y}$$




**Strength of linear relationship**

**Positive or negative**

**Not: non-linear relationships**

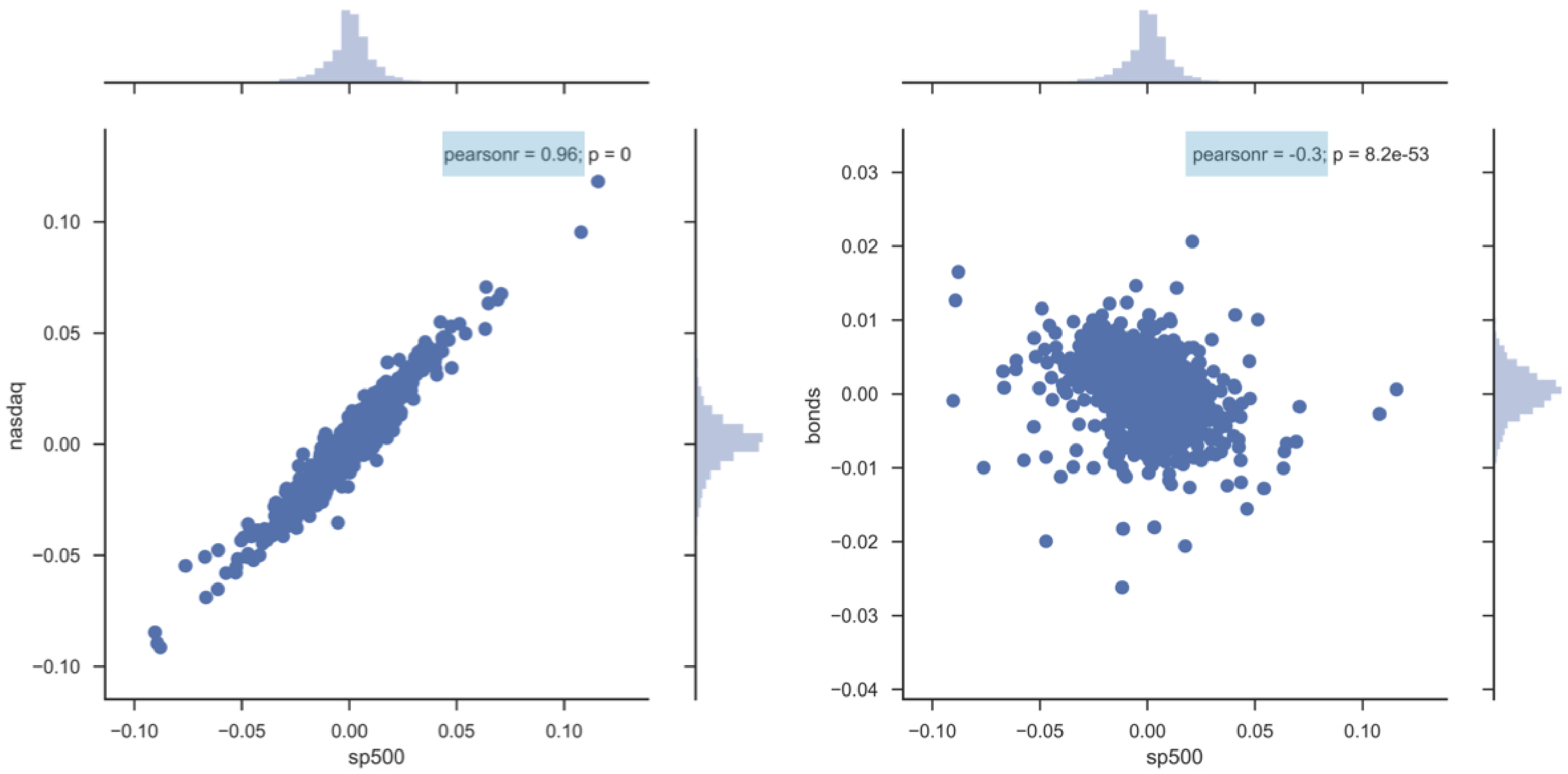
# Importing five price time series

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

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

```
daily_returns = data.pct_change()

sns.jointplot(x='sp500', y='nasdaq', data=data_returns);
```



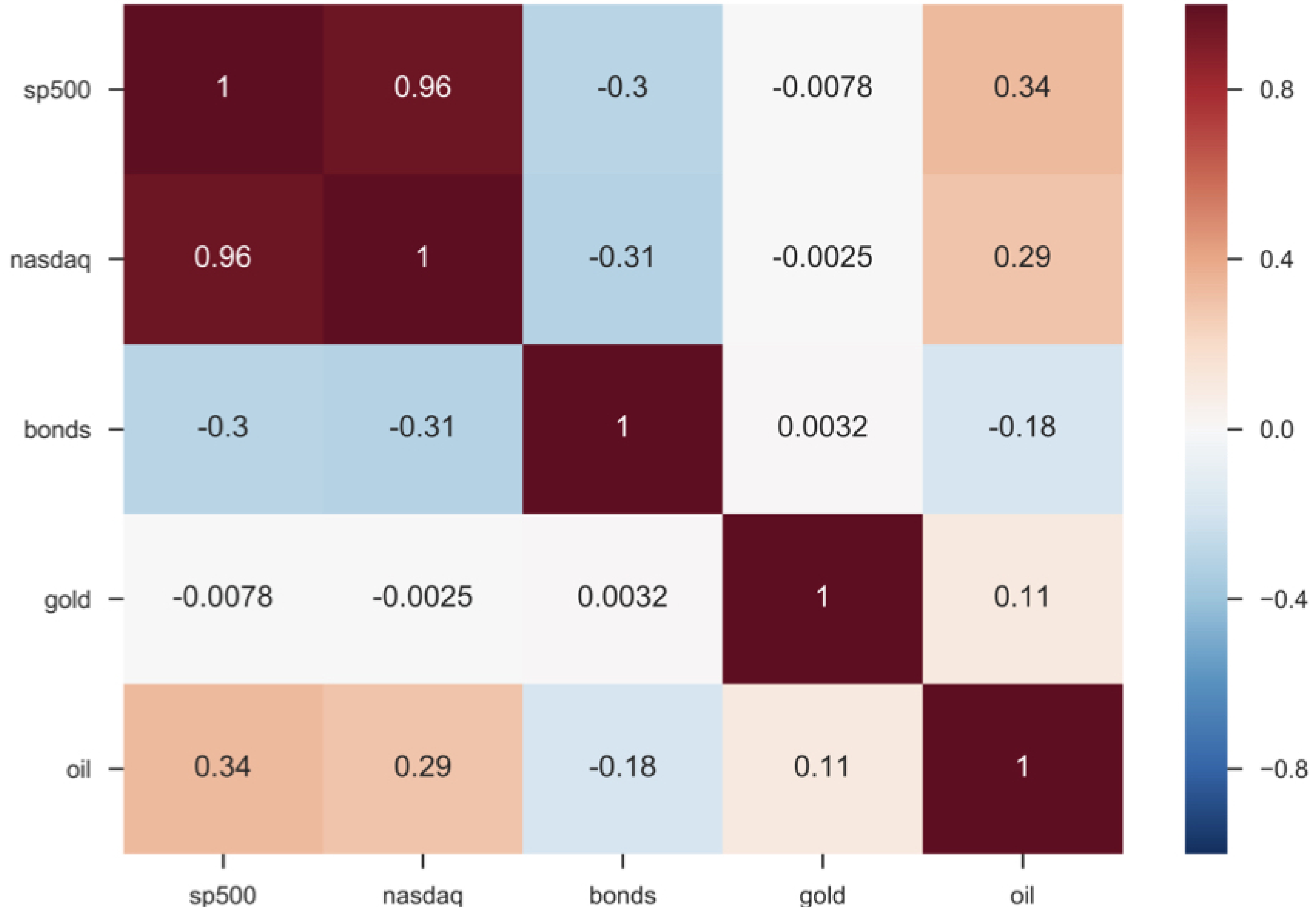
# Calculate all correlations

```
correlations = returns.corr()
correlations
```

```
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

```
sns.heatmap(correlations, annot=True)
```



# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON