# Compare time series growth rates

MANIPULATING TIME SERIES DATA IN PYTHON

**Stefan Jansen**

Founder & Lead Data Scientist at Applied Artificial Intelligence

# Comparing stock performance

- Stock price series: hard to compare at different levels
- Simple solution: normalize price series to start at 100
- Divide all prices by first in series, multiply by 100
  - Same starting point
  - All prices relative to starting point
  - Difference to starting point in percentage points

# Normalizing a single series (1)

```
google = pd.read_csv('google.csv', parse_dates=['date'], index_col='date')
google.head(3)
```

```
            price
date
2010-01-04  313.06
2010-01-05  311.68
2010-01-06  303.83
```

```
first_price = google.price.iloc[0]  # int-based selection
first_price
```

```
313.06
```

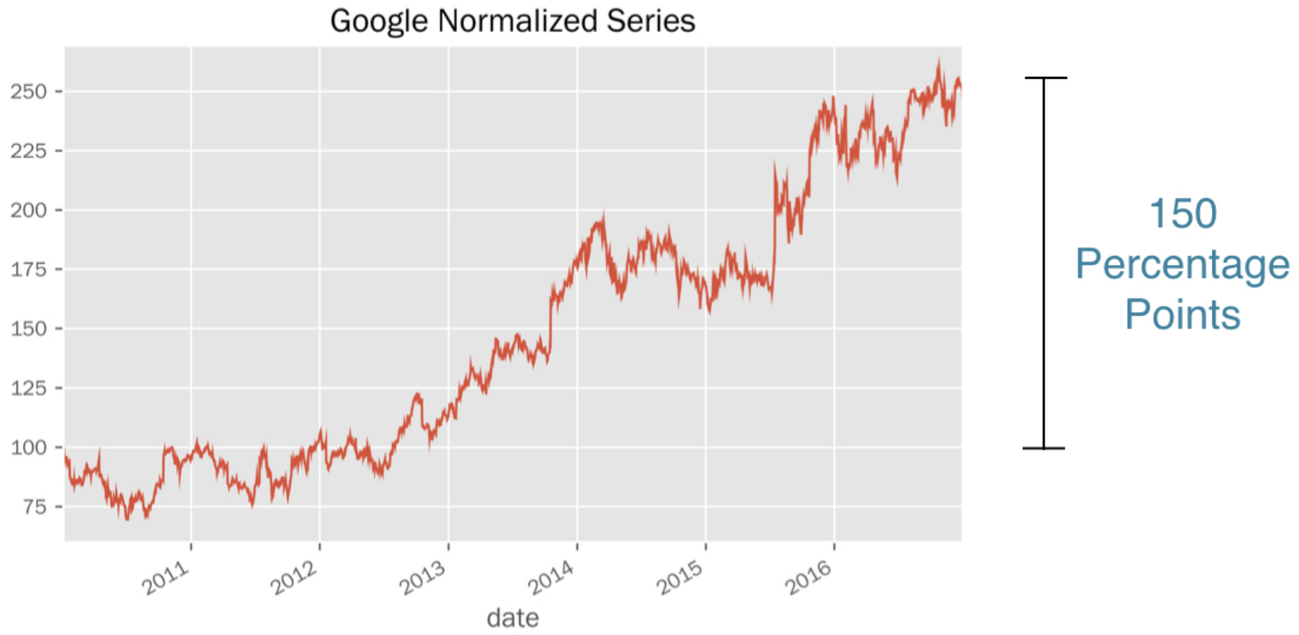
```
first_price == google.loc['2010-01-04', 'price']
```

```
True
```

# Normalizing a single series (2)

```
normalized = google.price.div(first_price).mul(100)

normalized.plot(title='Google Normalized Series')
```



# Normalizing multiple series (1)

```
prices = pd.read_csv('stock_prices.csv',
                     parse_dates=['date'],
                     index_col='date')
prices.info()
```

```
DatetimeIndex: 1761 entries, 2010-01-04 to 2016-12-30
Data columns (total 3 columns):
AAPL    1761 non-null float64
GOOG    1761 non-null float64
YHOO    1761 non-null float64
dtypes: float64(3)
```

```
prices.head(2)
```

```
             AAPL    GOOG   YHOO
Date
2010-01-04  30.57  313.06  17.10
2010-01-05  30.63  311.68  17.23
```

# Normalizing multiple series (2)

```
prices.iloc[0]
```

```
AAPL     30.57
GOOG    313.06
YHOO     17.10
Name: 2010-01-04 00:00:00, dtype: float64
```

```
normalized = prices.div(prices.iloc[0])
normalized.head(3)
```

```
                AAPL      GOOG      YHOO
Date
2010-01-04  1.000000  1.000000  1.000000
2010-01-05  1.001963  0.995592  1.007602
2010-01-06  0.985934  0.970517  1.004094
```

- `.div()` : automatic alignment of Series index & DataFrame columns

# Comparing with a benchmark (1)

```
index = pd.read_csv('benchmark.csv', parse_dates=['date'], index_col='date')
index.info()
```

```
DatetimeIndex: 1826 entries, 2010-01-01 to 2016-12-30
Data columns (total 1 columns):
SP500    1762 non-null float64
dtypes: float64(1)
```

```
prices = pd.concat([prices, index], axis=1).dropna()
prices.info()
```

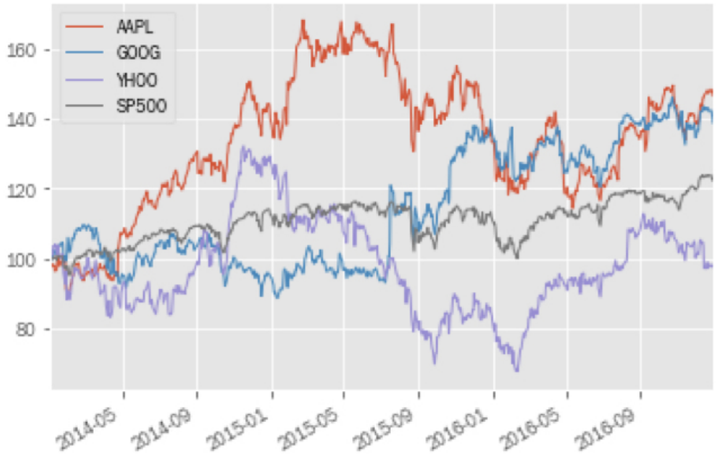
```
DatetimeIndex: 1761 entries, 2010-01-04 to 2016-12-30
Data columns (total 4 columns):
AAPL     1761 non-null float64
GOOG     1761 non-null float64
YHOO     1761 non-null float64
SP500    1761 non-null float64
dtypes: float64(4)
```

# Comparing with a benchmark (2)

```
prices.head(1)
```

```
              AAPL    GOOG   YHOO    SP500
2010-01-04  30.57  313.06  17.10  1132.99
```

```
normalized = prices.div(prices.iloc[0]).mul(100)
normalized.plot()
```



# Plotting performance difference

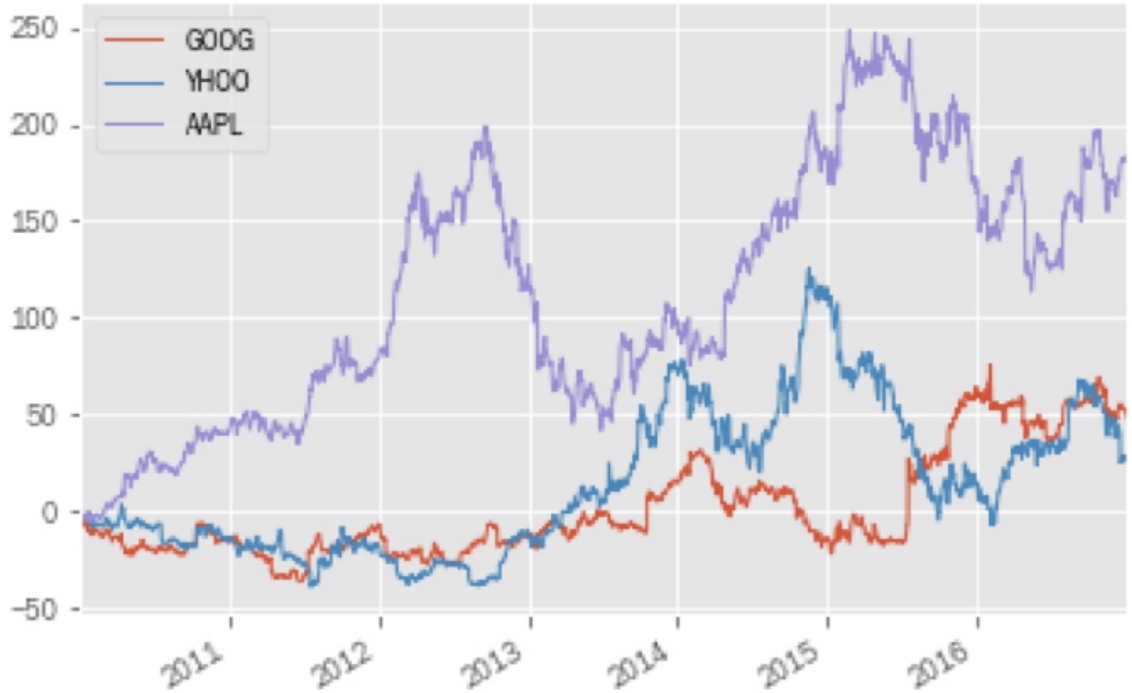
```
diff = normalized[tickers].sub(normalized['SP500'], axis=0)
```

```
                GOOG      YHOO      AAPL
2010-01-04  0.000000  0.000000  0.000000
2010-01-05 -0.752375  0.448669 -0.115294
2010-01-06 -3.314604  0.043069 -1.772895
```

- `.sub(..., axis=0)` : Subtract a Series from each DataFrame column by aligning indexes

# Plotting performance difference

```
diff.plot()
```



# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

# Changing the time series frequency: resampling

MANIPULATING TIME SERIES DATA IN PYTHON

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

# Changing the frequency: resampling

- `DateTimeIndex` : set & change freq using `.asfreq()`
- But frequency conversion affects the data
  - Upsampling: fill or interpolate missing data
  - Downsampling: aggregate existing data
- `pandas` API:
  - `.asfreq()` , `.reindex()`
  - `.resample()` + transformation method

# Getting started: quarterly data

```
dates = pd.date_range(start='2016', periods=4, freq='Q')

data = range(1, 5)

quarterly = pd.Series(data=data, index=dates)

quarterly
```

```
2016-03-31    1
2016-06-30    2
2016-09-30    3
2016-12-31    4
Freq: Q-DEC, dtype: int64 # Default: year-end quarters
```

# Upsampling: quarter => month

```
monthly = quarterly.asfreq('M') # to month-end frequency
```

```
2016-03-31    1.0
2016-04-30    NaN
2016-05-31    NaN
2016-06-30    2.0
2016-07-31    NaN
2016-08-31    NaN
2016-09-30    3.0
2016-10-31    NaN
2016-11-30    NaN
2016-12-31    4.0
Freq: M, dtype: float64
```

- Upsampling creates missing values

```
monthly = monthly.to_frame('baseline') # to DataFrame
```

# Upsampling: fill methods

```
monthly['ffill'] = quarterly.asfreq('M', method='ffill')
monthly['bfill'] = quarterly.asfreq('M', method='bfill')
monthly['value'] = quarterly.asfreq('M', fill_value=0)
```

# Upsampling: fill methods

- `bfill` : backfill
- `ffill` : forward fill

```
            baseline  ffill  bfill  value
2016-03-31       1.0      1      1      1
2016-04-30       NaN      1      2      0
2016-05-31       NaN      1      2      0
2016-06-30       2.0      2      2      2
2016-07-31       NaN      2      3      0
2016-08-31       NaN      2      3      0
2016-09-30       3.0      3      3      3
2016-10-31       NaN      3      4      0
2016-11-30       NaN      3      4      0
2016-12-31       4.0      4      4      4
```

# Add missing months: .reindex()

```
dates = pd.date_range(start='2016',
                      periods=12,
                      freq='M')
```

```
DatetimeIndex(['2016-01-31',
               '2016-02-29',
               ...,
               '2016-11-30',
               '2016-12-31'],
              dtype='datetime64[ns]', freq='M')
```

- `.reindex()` :
  - conform DataFrame to new index
  - same filling logic as `.asfreq()`

```
quarterly.reindex(dates)
```

```
2016-01-31    NaN
2016-02-29    NaN
2016-03-31    1.0
2016-04-30    NaN
2016-05-31    NaN
2016-06-30    2.0
2016-07-31    NaN
2016-08-31    NaN
2016-09-30    3.0
2016-10-31    NaN
2016-11-30    NaN
2016-12-31    4.0
```

# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

# Upsampling & interpolation with .resample()

MANIPULATING TIME SERIES DATA IN PYTHON

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

# Frequency conversion & transformation methods

- `.resample()` : similar to `.groupby()`
- Groups data within resampling period and applies one or several methods to each group
- New date determined by offset - start, end, etc
- Upsampling: fill from existing or interpolate values
- Downsampling: apply aggregation to existing data

# Getting started: monthly unemployment rate

```
unrate = pd.read_csv('unrate.csv', parse_dates['Date'], index_col='Date')
unrate.info()
```

```
DatetimeIndex: 208 entries, 2000-01-01 to 2017-04-01
Data columns (total 1 columns):
UNRATE    208 non-null float64 # no frequency information
dtypes: float64(1)
```

```
unrate.head()
```

```
            UNRATE
DATE
2000-01-01     4.0
2000-02-01     4.1
2000-03-01     4.0
2000-04-01     3.8
2000-05-01     4.0
```

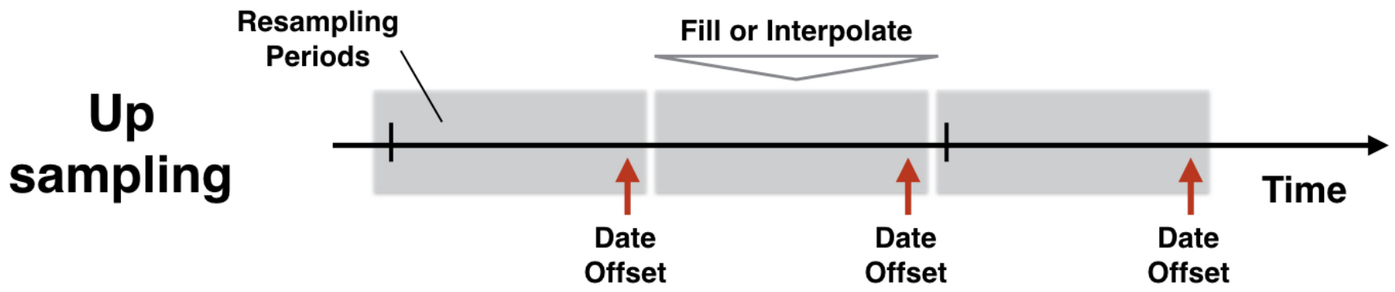
- Reporting date: 1st day of month

# Resampling Period & Frequency Offsets

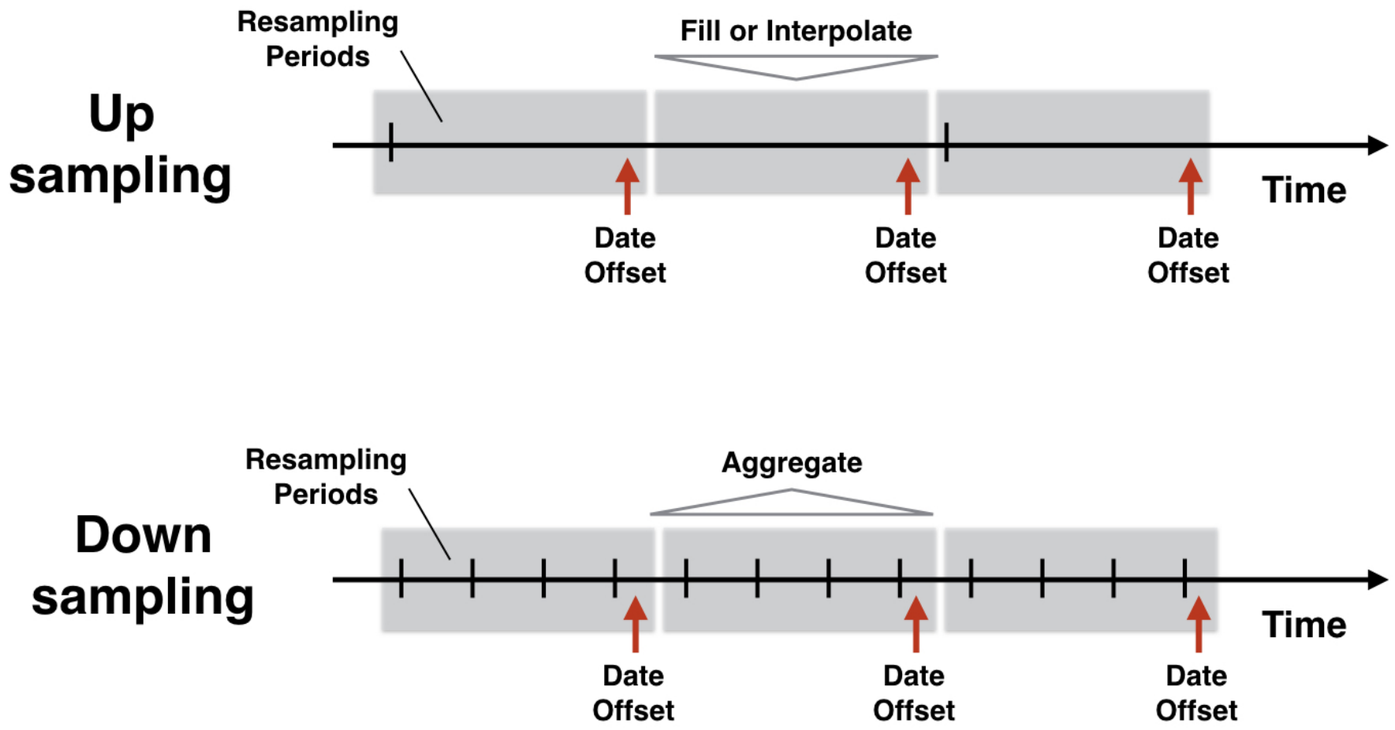
- Resample creates new date for frequency offset
- Several alternatives to calendar month end

| Frequency | Alias | Sample Date |
|---|---|---|
| Calendar Month End | M | 2017-04-30 |
| Calendar Month Start | MS | 2017-04-01 |
| Business Month End | BM | 2017-04-28 |
| Business Month Start | BMS | 2017-04-03 |

# Resampling logic



# Resampling logic



# Assign frequency with .resample()

```
unrate.asfreq('MS').info()
```

```
DatetimeIndex: 208 entries, 2000-01-01 to 2017-04-01
Freq: MS
Data columns (total 1 columns):
UNRATE    208 non-null float64
dtypes: float64(1)
```

```
unrate.resample('MS') # creates Resampler object
```

```
DatetimeIndexResampler [freq=<MonthBegin>, axis=0, closed=left,
                        label=left, convention=start, base=0]
```

# Assign frequency with .resample()

```
unrate.asfreq('MS').equals(unrate.resample('MS').asfreq())
```

```
True
```

- `.resample()` : returns data only when calling another method

# Quarterly real GDP growth

```
gdp = pd.read_csv('gdp.csv')
gdp.info()
```

```
DatetimeIndex: 69 entries, 2000-01-01 to 2017-01-01
Data columns (total 1 columns):
gpd    69 non-null float64 # no frequency info
dtypes: float64(1)
```

```
gdp.head(2)
```

```
            gpd
DATE
2000-01-01  1.2
2000-04-01  7.8
```

# Interpolate monthly real GDP growth

```
gdp_1 = gdp.resample('MS').ffill().add_suffix('_ffill')
```

```
            gpd_ffill
DATE
2000-01-01        1.2
2000-02-01        1.2
2000-03-01        1.2
2000-04-01        7.8
```

# Interpolate monthly real GDP growth

```
gdp_2 = gdp.resample('MS').interpolate().add_suffix('_inter')
```

```
            gpd_inter
DATE
2000-01-01   1.200000
2000-02-01   3.400000
2000-03-01   5.600000
2000-04-01   7.800000
```

- `.interpolate()` : finds points on straight line between existing data

# Concatenating two DataFrames

```
df1 = pd.DataFrame([1, 2, 3], columns=['df1'])
df2 = pd.DataFrame([4, 5, 6], columns=['df2'])
pd.concat([df1, df2])
```

```
   df1  df2
0  1.0  NaN
1  2.0  NaN
2  3.0  NaN
0  NaN  4.0
1  NaN  5.0
2  NaN  6.0
```

# Concatenating two DataFrames

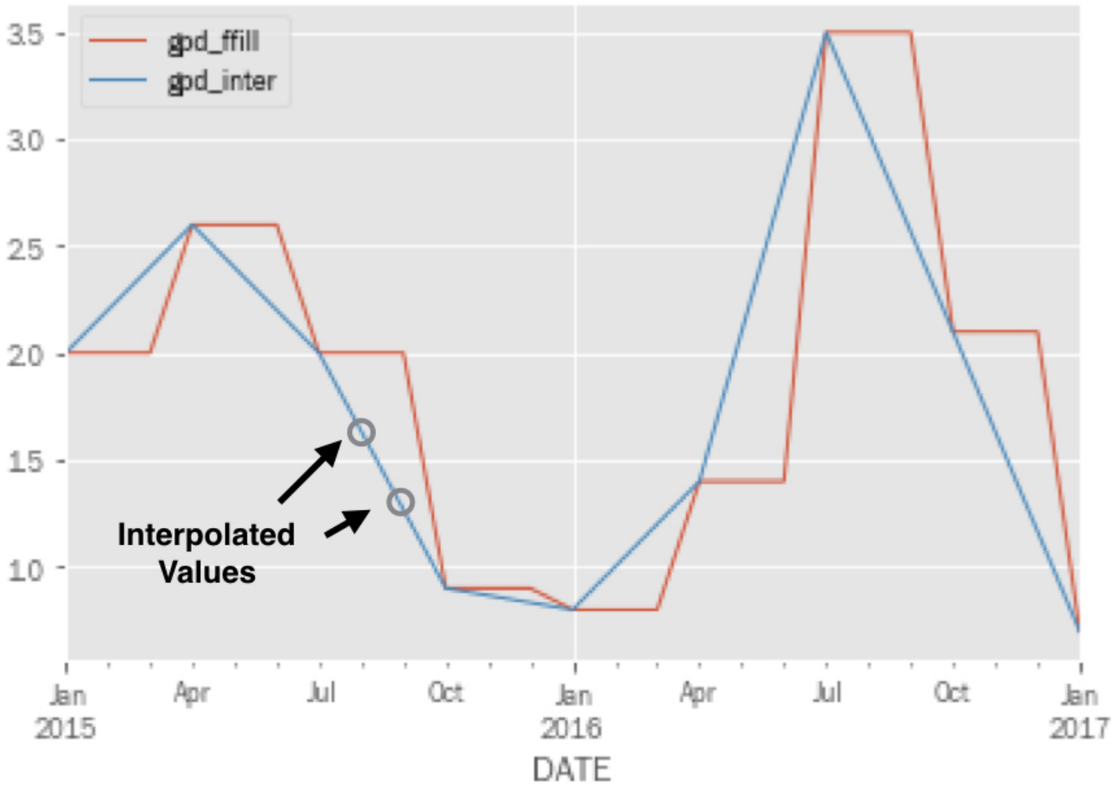
```
pd.concat([df1, df2], axis=1)
```

```
   df1  df2
0    1    4
1    2    5
2    3    6
```

- `axis=1` : concatenate horizontally

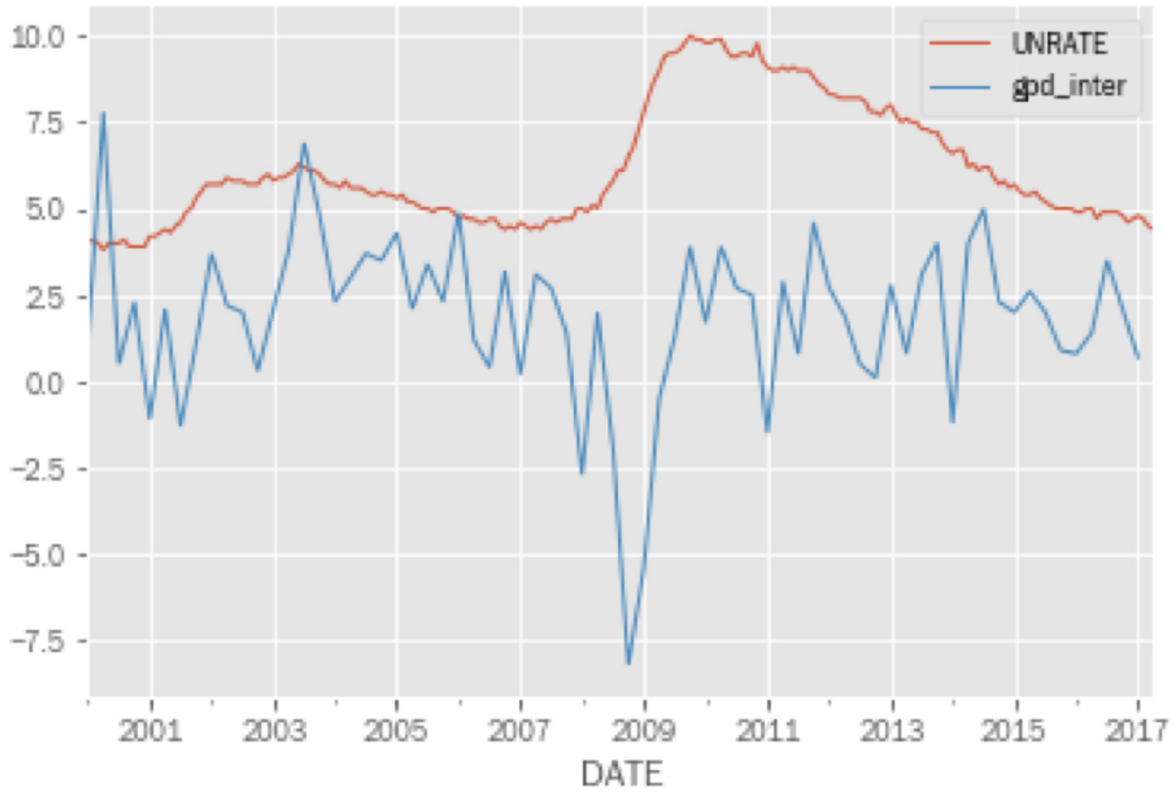
# Plot interpolated real GDP growth

```
pd.concat([gdp_1, gdp_2], axis=1).loc['2015':].plot()
```



# Combine GDP growth & unemployment

```
pd.concat([unrate, gdp_inter], axis=1).plot();
```



# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

# Downsampling & aggregation

MANIPULATING TIME SERIES DATA IN PYTHON

**Stefan Jansen**

Founder & Lead Data Scientist at Applied Artificial Intelligence

# Downsampling & aggregation methods

- So far: upsampling, fill logic & interpolation
- Now: downsampling
  - hour to day
  - day to month, etc
- How to represent the existing values at the new date?
  - Mean, median, last value?

# Air quality: daily ozone levels

```
ozone = pd.read_csv('ozone.csv',
                    parse_dates=['date'],
                    index_col='date')
ozone.info()
```

```
DatetimeIndex: 6291 entries, 2000-01-01 to 2017-03-31
Data columns (total 1 columns):
Ozone    6167 non-null float64
dtypes: float64(1)
```

```
ozone = ozone.resample('D').asfreq()
ozone.info()
```

```
DatetimeIndex: 6300 entries, 1998-01-05 to 2017-03-31
Freq: D
Data columns (total 1 columns):
Ozone    6167 non-null float64
dtypes: float64(1)
```

# Creating monthly ozone data

```
ozone.resample('M').mean().head()
```

```
            Ozone
date
2000-01-31  0.010443
2000-02-29  0.011817
2000-03-31  0.016810
2000-04-30  0.019413
2000-05-31  0.026535
```

```
ozone.resample('M').median().head()
```

```
            Ozone
date
2000-01-31  0.009486
2000-02-29  0.010726
2000-03-31  0.017004
2000-04-30  0.019866
2000-05-31  0.026018
```

`.resample().mean()` : Monthly average, assigned to end of calendar month

# Creating monthly ozone data

```
ozone.resample('M').agg(['mean', 'std']).head()
```

```
              Ozone
               mean       std
date
2000-01-31  0.010443  0.004755
2000-02-29  0.011817  0.004072
2000-03-31  0.016810  0.004977
2000-04-30  0.019413  0.006574
2000-05-31  0.026535  0.008409
```

- `.resample().agg()` : List of aggregation functions like groupby

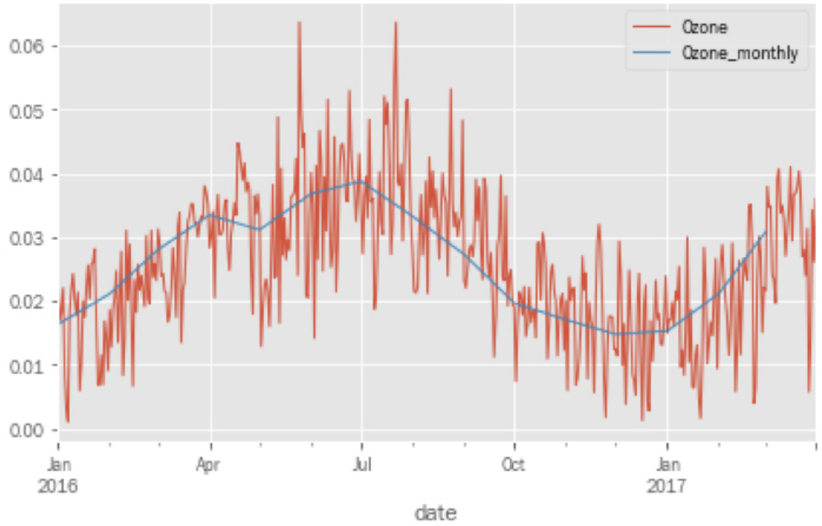
# Plotting resampled ozone data

```
ozone = ozone.loc['2016':]

ax = ozone.plot()

monthly = ozone.resample('M').mean()

monthly.add_suffix('_monthly').plot(ax=ax)
```




ax=ax:
Matplotlib let's you plot again on the axes object returned by the first plot

# Resampling multiple time series

```
data = pd.read_csv('ozone_pm25.csv',
                   parse_dates=['date'],
                   index_col='date')

data = data.resample('D').asfreq()

data.info()
```

```
DatetimeIndex: 6300 entries, 2000-01-01 to 2017-03-31
Freq: D
Data columns (total 2 columns):
Ozone    6167 non-null float64
PM25     6167 non-null float64
dtypes: float64(2)
```

# Resampling multiple time series

```
data = data.resample('BM').mean()

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 207 entries, 2000-01-31 to 2017-03-31
Freq: BM
Data columns (total 2 columns):
ozone    207 non-null float64
pm25     207 non-null float64
dtypes: float64(2)
```

# Resampling multiple time series

```
df.resample('M').first().head(4)
```

```
              Ozone       PM25
date
2000-01-31  0.005545  20.800000
2000-02-29  0.016139   6.500000
2000-03-31  0.017004   8.493333
2000-04-30  0.031354   6.889474
```

```
df.resample('MS').first().head()
```

```
              Ozone       PM25
date
2000-01-01  0.004032  37.320000
2000-02-01  0.010583  24.800000
2000-03-01  0.007418  11.106667
2000-04-01  0.017631  11.700000
2000-05-01  0.022628   9.700000
```

# Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON