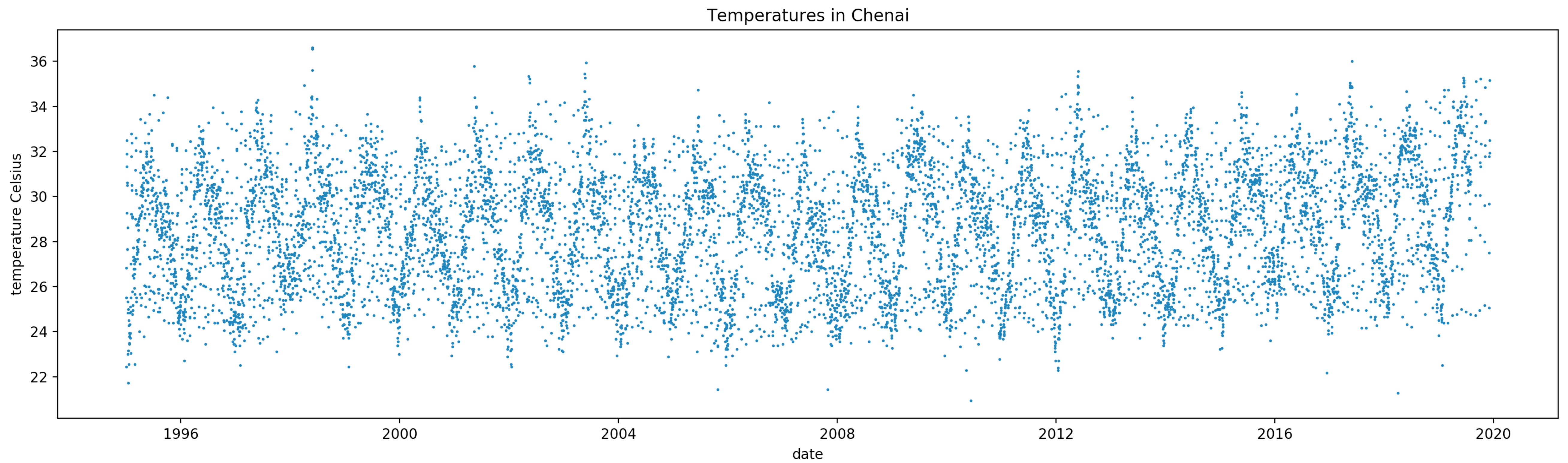
Time Series Analysis with Pandas

Thomas Schwarz, SJ

Time Series

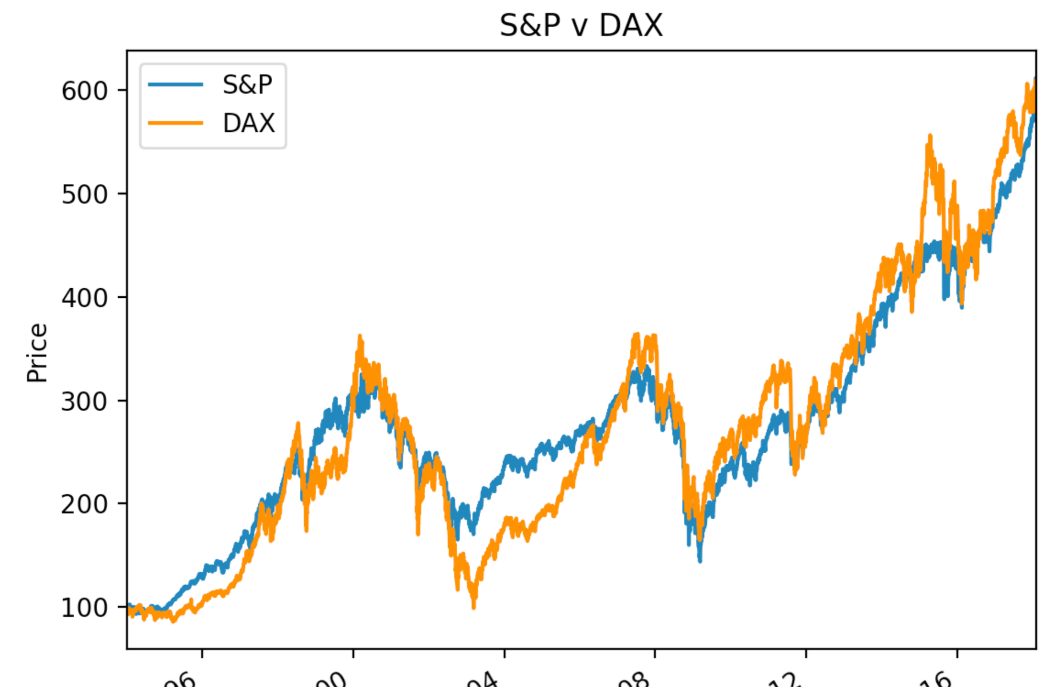
- Study of statistical data that depends on the time
  - Examples:
    - births in the US on a given day
    - names given to children in a certain year
    - exchange rates
    - …

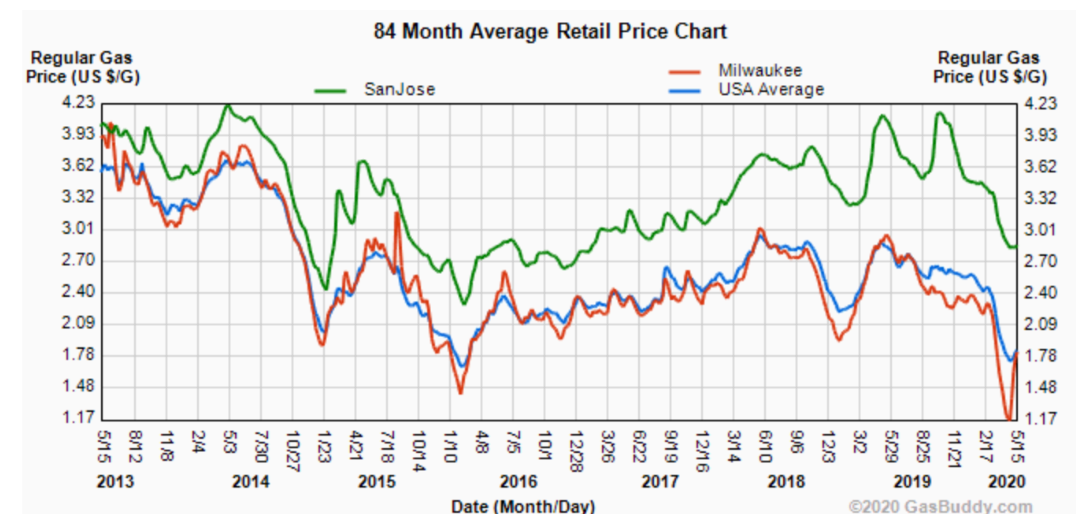
Introduction to Time Series



Introduction to Time Series

- Time Series Data is highly correlated with itself
    - Normal statistical descriptions such as mean are not very useful
    - Temperature, stock market, gas prices have long-term trends
    - Temperature and gas prices have seasonal trends





Introduction to Time Series

- Dealing with time data:
  - Generate time plot to see what is happening
    - Usually import from csv and transform data
  - Determine optically trends, cycles, outliers, undefined or obviously wrong values
  - Determine whether there is a need for transformation
    - e.g. our stock exchange data is normalized to make DOW and DAX comparable

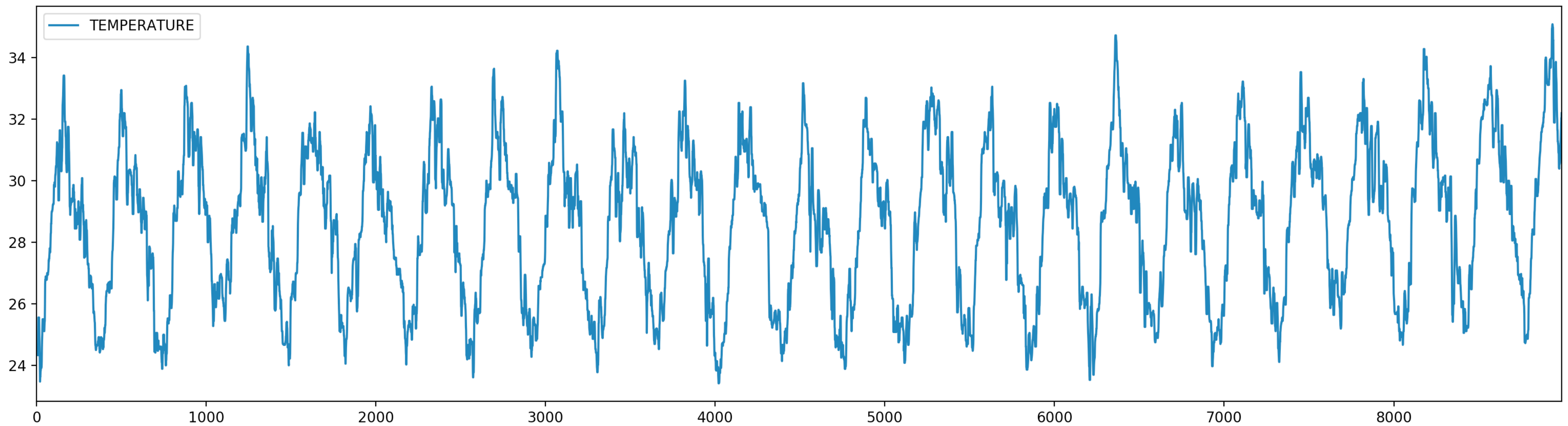
Introduction to Time Series

- Typical transformations:
  - Linear normalizations
  - Logarithmic, exponential
    - E.g. variance grows with mean —> Logarithmic transform
    - Make a multiplicative dependence additive
      - timevalue = trend * seasonal * random —> Logarithmic: timevalue = trend+seasonal+random

Introduction to Time Series

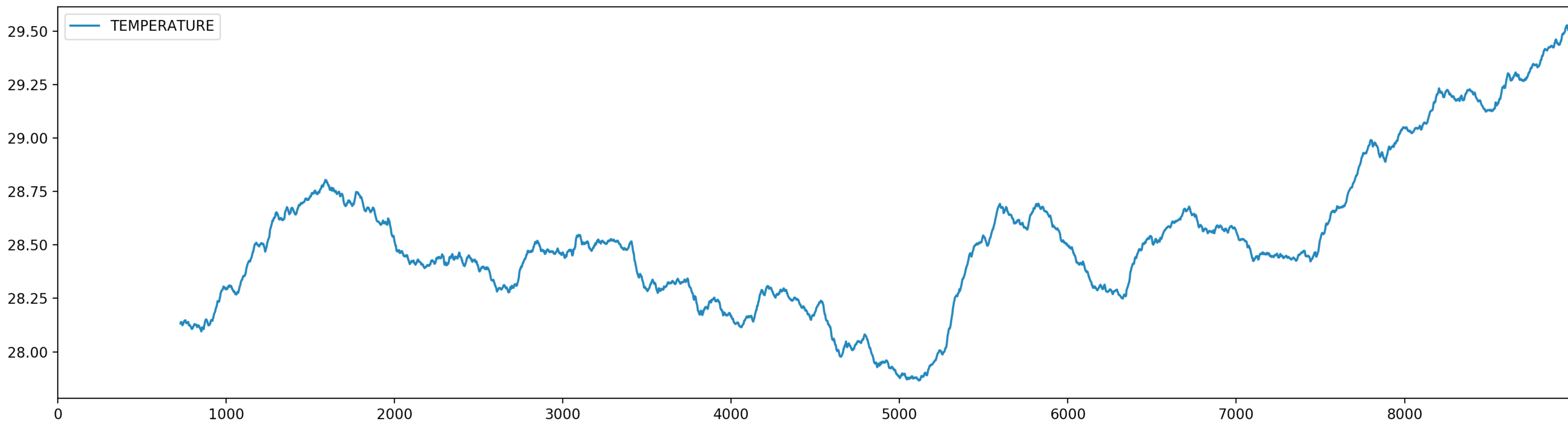
- Filtering: Transform time series (x_t) into other time series (y_t)
  - Smoothing out local variations
    - E.g. Linear smoothing with weights $a_{-s}, a_{-s+1}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{r-1}, a_r$
    - $y_t = \sum_{\nu=-s}^{r} a_\nu x_{t+\nu}$
    - Eg. Moving average: $y_t = \frac{1}{2n+1} \sum_{\nu=-n}^{n} x_{t+\nu}$

Introduction to Time Series



Median Smoothing with a Window of 10 of the Chenai Temperature Data

Introduction to Time Series



Smoothing over 2 years with r

Introduction to Time Series

- Seasonal variations:
  - Three main models
    - $x_t = m_t + s_t + \epsilon_t$
    - $x_t = m_t \cdot s_t + \epsilon_t$
    - $x_t = m_t \cdot s_t \cdot \epsilon_t$

- Where
  - x_t time-series value
  - m_t long-time trend
  - s_t seasonal component
  - ϵ_t random noise

Auto-Correlation

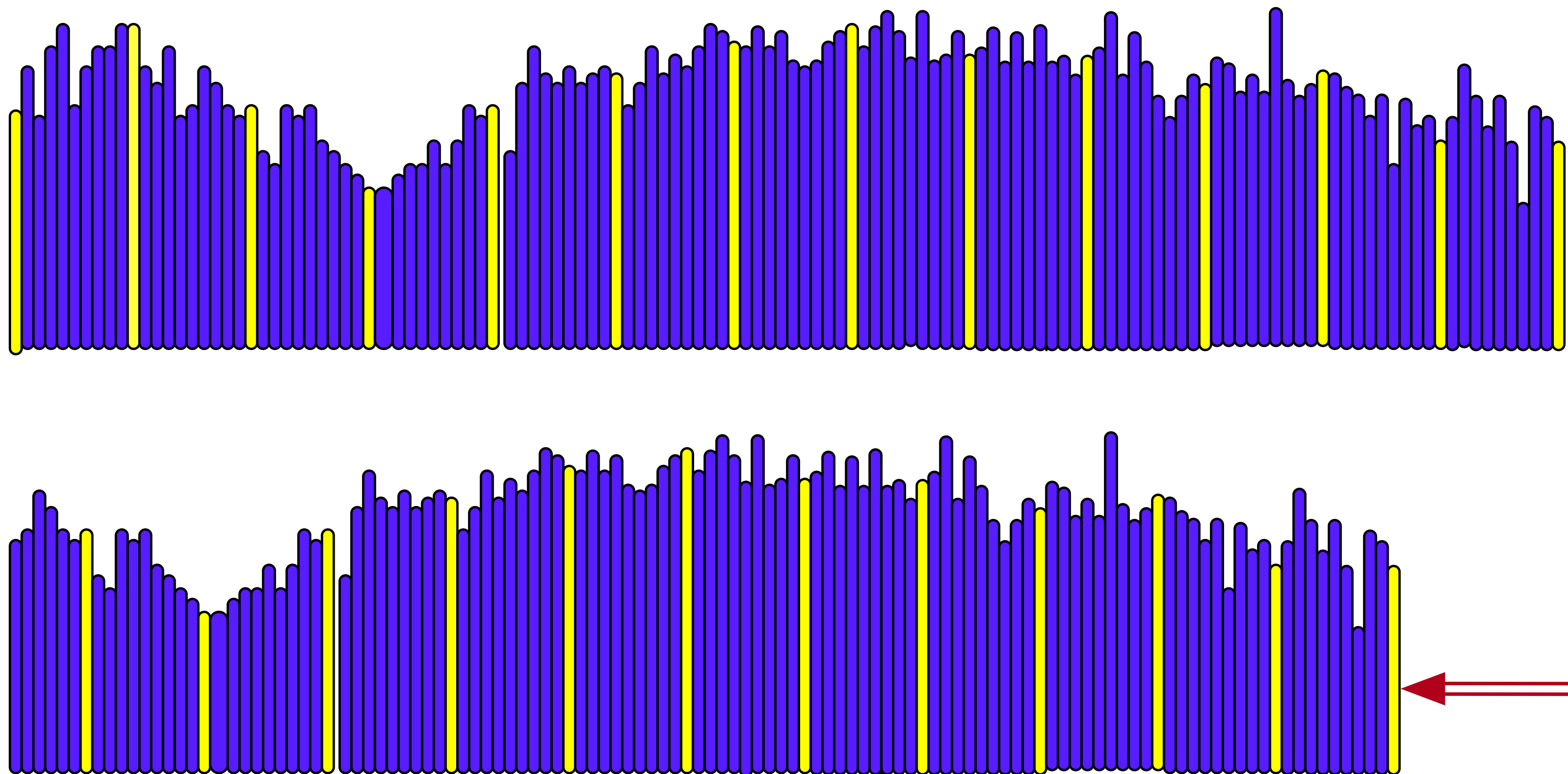
- Covariance:
  - Two random variables X, Y with means $E(X), E(Y)$
  - Covariance $\text{Cov}_{X,Y} = E\left((X - E(X)) \cdot (Y - E(Y))\right)$
  - If X and Y are independent:
    - $\text{Cov}_{X,Y} = E(XY - E(X)Y - E(Y)X + E(X)E(Y)) = 0$
  - If $\text{Cov}_{X,Y} > 0$:
    - High values for X go with high values for Y
  - If $\text{Cov}_{X,Y} < 0$:
    - High values for X go with low values for Y

Auto-Correlation

- Covariance depends on the units in which X and Y are given
- Therefore, we look at the correlation coefficient
    - If X and Y have standard deviations σ_X and σ_Y respectively:
    - $\rho_{X,Y} = \dfrac{\text{Cov}_{X,Y}}{\sigma_X \sigma_Y} \in [-1,1]$

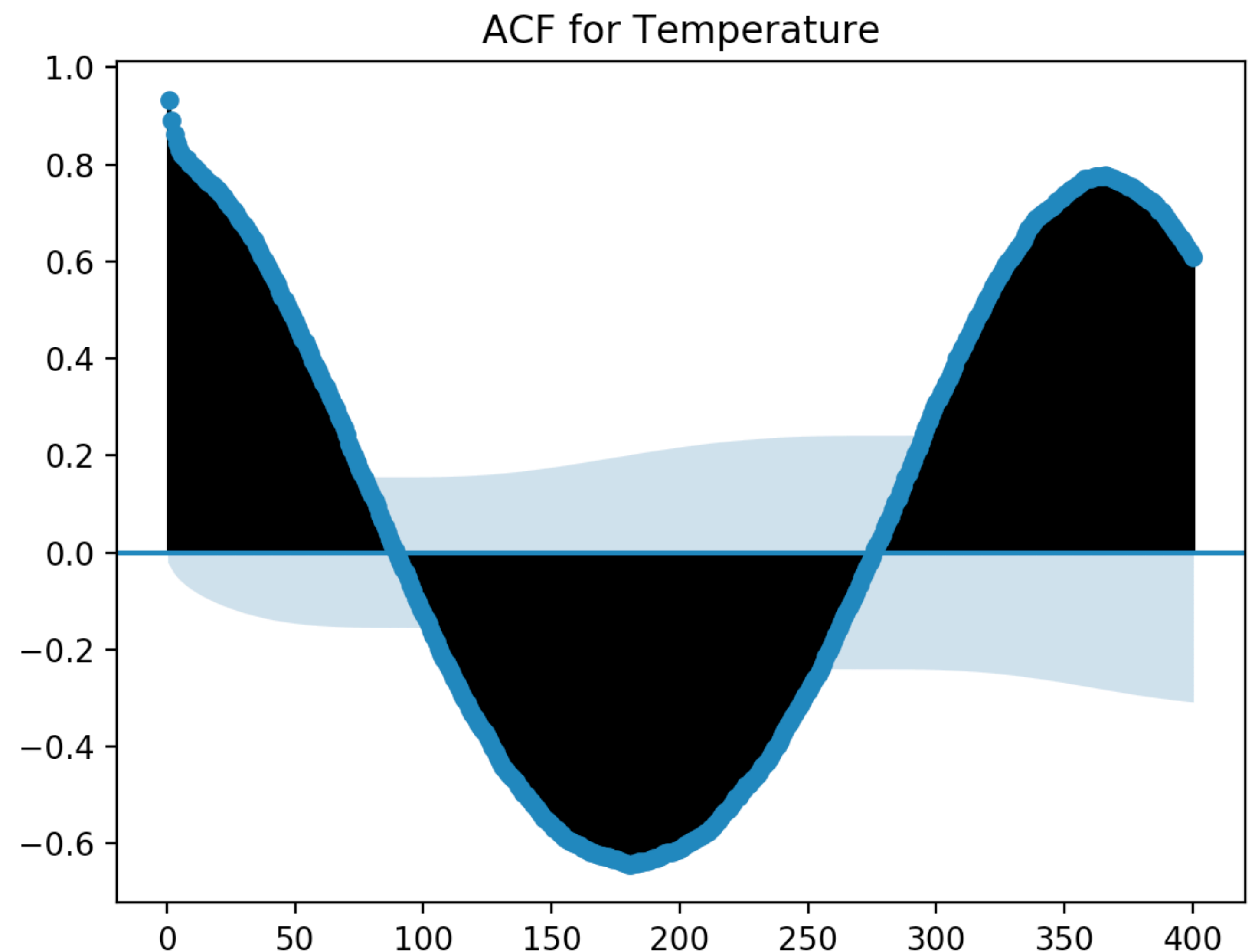
Introduction to Time Series

- Auto-correlation: Compare the correlation of a time series with itself moved by k positions



Introduction to Time Series

- Autocorrelation for Chenai temperatures
    - High auto-correlation for the next day and for a year afterwards
    - The light blue denotes significance



Introduction to Time Series

- Goal:
  - Prediction of future value
- Testing:
  - Use latest data and predict based on previous data

Time Series in Pandas Overview

- Time series can be
    - *fixed frequency*
        - data points occur at regular intervals
    - *irregular*
        - no fixed unit of time / offset between data points

Time Series in Pandas Overview

- Time is given in
  - Timestamps
  - Fixed periods (January 2007)
  - Intervals of time such as periods
  - Experiment or elapsed time
    - Each timestamp measures the difference relative to a particular start time
      - For this: use `timedelta` indices in Pandas

Times in Python

- Python time in datetime

```
from datetime import datetime
my_date(year=2020, month = 2, day = 14)
```

  - Can take hours, minutes, seconds, microseconds as well
  - Also timezone information
- Can be combined with timedelta in order to calculate with dates

Times in Python

- datetime has a μsec resolution
- timedelta for the difference between two datetime objects

```
>>> from datetime import datetime
>>> now = datetime.now()
>>> now
datetime.datetime(2020, 6, 30, 14, 11, 29, 82191)
>>> now.year, now.month, now.day
(2020, 6, 30)

>>> delta = now - datetime(2001,6,16,11)
>>> delta
datetime.timedelta(days=6954, seconds=11489,
microseconds=82191)
```

Times in Python

- We can calculate with datetime and timedelta objects

```
>>> from datetime import datetime, timedelta
>>> timedelta(12)
datetime.timedelta(days=12)

>>> now = datetime.now()
>>> now + 3.5*timedelta(10)
datetime.datetime(2020, 8, 4, 14, 17, 1, 562436)
```

Times in Python

- We can convert between string and datetime (or date or time)
    - `strftime` takes a format argument

```
>>> from datetime import date, time, datetime, time
>>> stamp = datetime(1776, 7, 4)
>>> str(stamp)
'1776-07-04 00:00:00'
>>> stamp.strftime('%d-%m-%Y')
'04-07-1776'
```

Times in Python

- Format codes:
  - Can also be used to convert from string to date

Type	Description
%Y	4-digit year
%y	2-digit year
%m	2-digit month
%d	2-digit day
%H	24 hour clock
%I	12 hour clock
%M	2-digit minute
%S	2-digit seconds
%w	weekday as integer
%U	week number in year
%W	week number (monday)
%z	UTC time zone offset
%F	Shortcut %Y-%m-%d
%D	Shortcut %m/%d/%y

Times in Python

```
>>> datetime.strptime('7/4/1776', '%m/%d/%Y')
datetime.datetime(1776, 7, 4, 0, 0)
```

Times in Pandas

- Pandas makes this easier
  - It comes with dateutil.parser
  - Which can guess most formats

```
>>> from dateutil.parser import parse
>>> parse('2020-06-30')
datetime.datetime(2020, 6, 30, 0, 0)
>>> parse('July 4, 2020')
datetime.datetime(2020, 7, 4, 0, 0)
>>> parse('4th of July, 2020')
datetime.datetime(2020, 7, 4, 0, 0)
>>> parse(1/2/2021, dayfirst=True)
```

Times in Pandas

- For the non-us world, need to use `dayfirst=True`

```
>>> parse('1/2/2021', dayfirst=True)
datetime.datetime(2021, 2, 1, 0, 0)
```

Times in Pandas

- A better solution than Python's for us is numpy's datetime
  - Called **datetime64**
    - Example:

```
np.array(['2020-01-01', '2020-01-02'], dtype='datetime64')
```

  - For instance, we can create equidistant arrays of timestamps
    - Notice the array notation " [Y] "

```
np.arange('2000', '2020', dtype = 'datetime64[Y]')
```

Times in Pandas

- In Pandas, there is DatetimeIndex

```
pd.date_range('2020-01-01', periods=12, freq='M')

DatetimeIndex(['2020-01-31', '2020-02-29', '2020-03-31',
'2020-04-30', '2020-05-31', '2020-06-30', '2020-07-31',
'2020-08-31','2020-09-30', '2020-10-31', '2020-11-30',
'2020-12-31'], dtype='datetime64[ns]', freq='M')
```

  - Notice the 'ns' as the default for the precision
  - Pandas is good at inferring the format of the string
  - For reading in data, we can use pd.to_datetime with the format parameter

Times in Pandas

- Example:
    - Let's generate some random data and then add a time index to it.
    - Use this function to generate some values

```
import numpy as np
import pandas as pd
import random
import math

def v(i,j):
    return 10+math.floor(
          0.3*i+0.1*i**0.4 + random.normalvariate(0,5
          )
```

Times in Pandas

- We create a value array from numpy with the confusing fromfunction function
- Then we create a time index, the dates being May 1, 2020 to May 31, 2020
- And finally make it into a data-frame

```
values = np.fromfunction(
                   np.vectorize(v),
                  (31,1)
             )
idx = pd.date_range('2020-05-01', periods = 31, freq='D')
df = pd.DataFrame(values, index=idx, columns=['values'])

print(df)
```

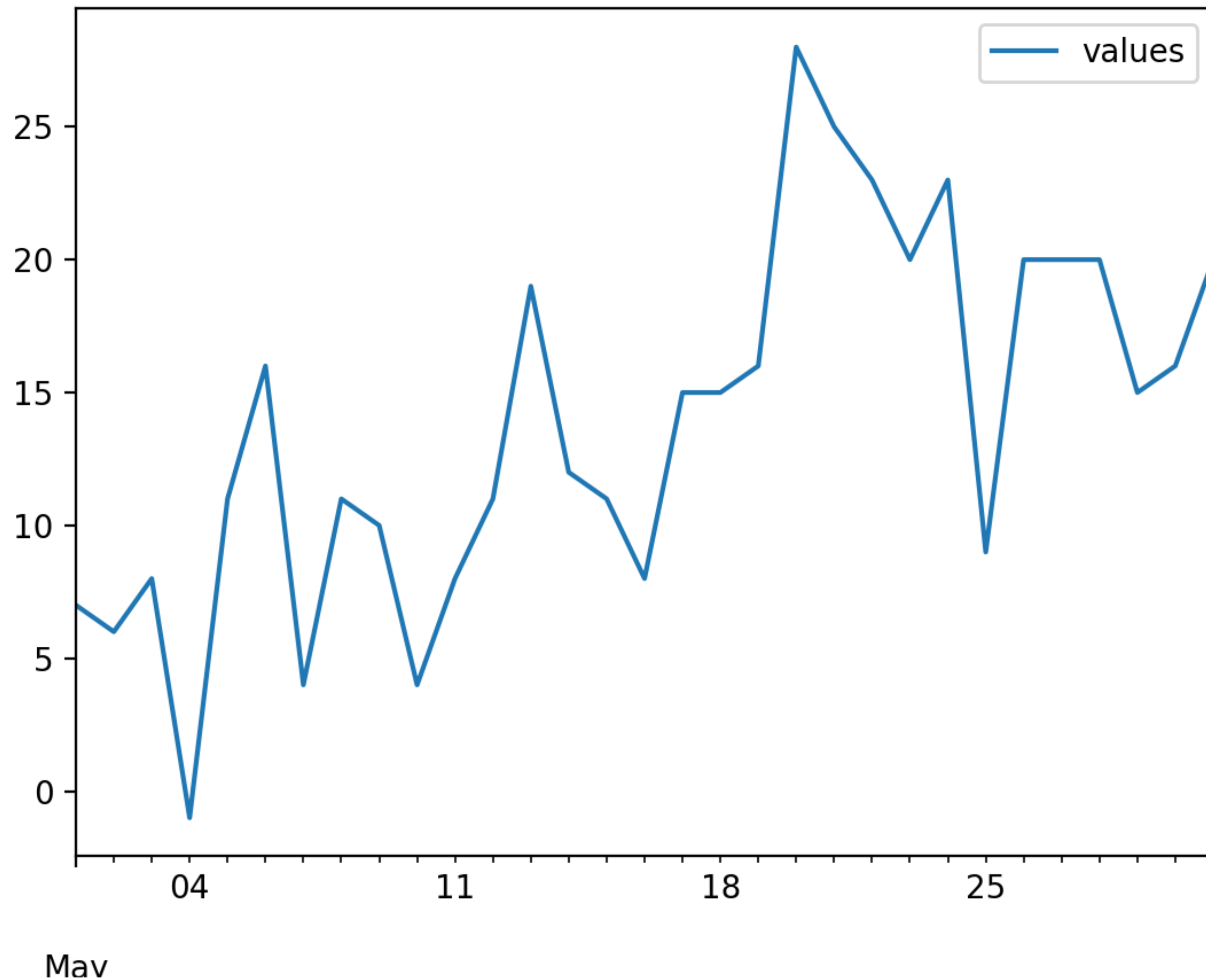
Times in Pandas

- Result:
  - (your values will differ)

```
            values
2020-05-01      13
2020-05-02       8
2020-05-03      13
2020-05-04      14
2020-05-05      14
2020-05-06      11
2020-05-07      19
2020-05-08      11
2020-05-09      12
2020-05-10      24
2020-05-11      22
2020-05-12      12
2020-05-13       3
2020-05-14      13
2020-05-15      16
2020-05-16      12
2020-05-17      13
2020-05-18      16
2020-05-19      19
2020-05-20      16
2020-05-21       8
2020-05-22      16
2020-05-23      23
2020-05-24      20
2020-05-25      17
2020-05-26      18
2020-05-27      13
2020-05-28      19
2020-05-29      15
2020-05-30      16
2020-05-31      11
```

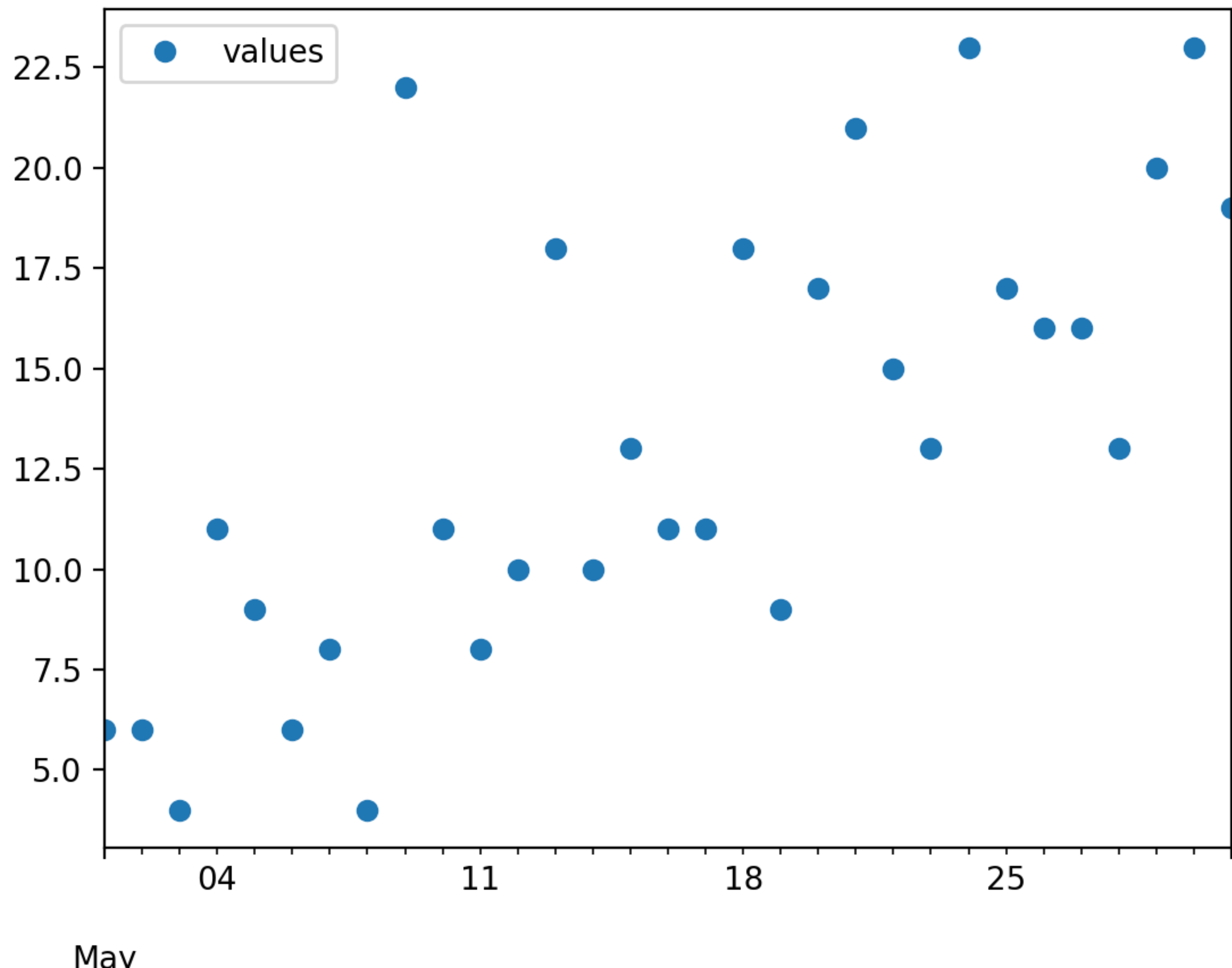
Time Series in Pandas Overview

```
df.plot()
plt.show()
```



Time Series in Pandas

```
df.plot(style ='o')
plt.show()
print(df)
```



Time Series Basics

- Usually a series or data frame object indexed by time-stamps
  - Often, the time stamps are hidden in the raw data as strings
- Arithmetic operations between differently indexed time-series automatically align on the dates

Time Series Basics

- Example:
  - Take previous times series and do average over three points
  - For this, need to create a dataframe with the indices shifted by one day
    - For this we use the timedelta

```
from datetime import timedelta

dfpre = pd.DataFrame(values,
            index=(idx-timedelta(1)),
            columns=['values']
)
```

Time Series Basics

- Example:

```
dfpre = pd.DataFrame(values,
            index=(idx-timedelta(1)),
            columns=['values'])
dfpost = pd.DataFrame(values,
            index=(idx+timedelta(1)),
            columns=['values'])
dfr = 0.5*df+0.25*dfpre+0.25*dfpost

ax = dfr.plot(style ='r-')
df.plot(style = 'b:', ax = ax)
```

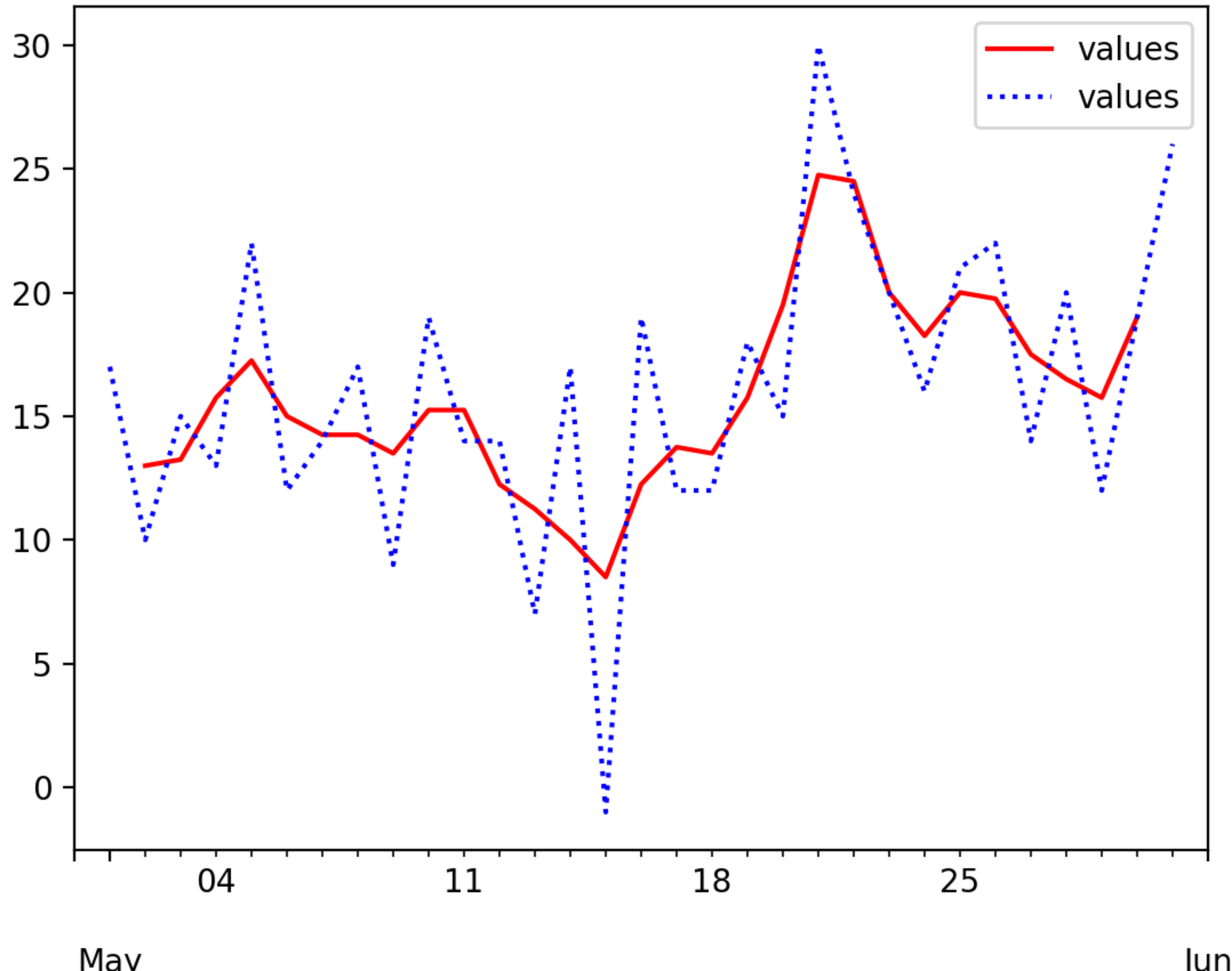
Time Series Basics

- Also: two display to graphs in the same figure:
  - df.plot returns an axes object
  - Which we can specify in the second plot:

```
dfpre = pd.DataFrame(values,
            index=(idx-timedelta(1)),
            columns=['values'])
dfpost = pd.DataFrame(values,
            index=(idx+timedelta(1)),
            columns=['values'])
dfr = 0.5*df+0.25*dfpre+0.25*dfpost

ax = dfr.plot(style ='r-')
df.plot(style = 'b:', ax = ax)
```

Time Series Basics



Time Series Indexing

- Notice that the "smoothed" value has no values for the first and last of the month

Time Series Indexing

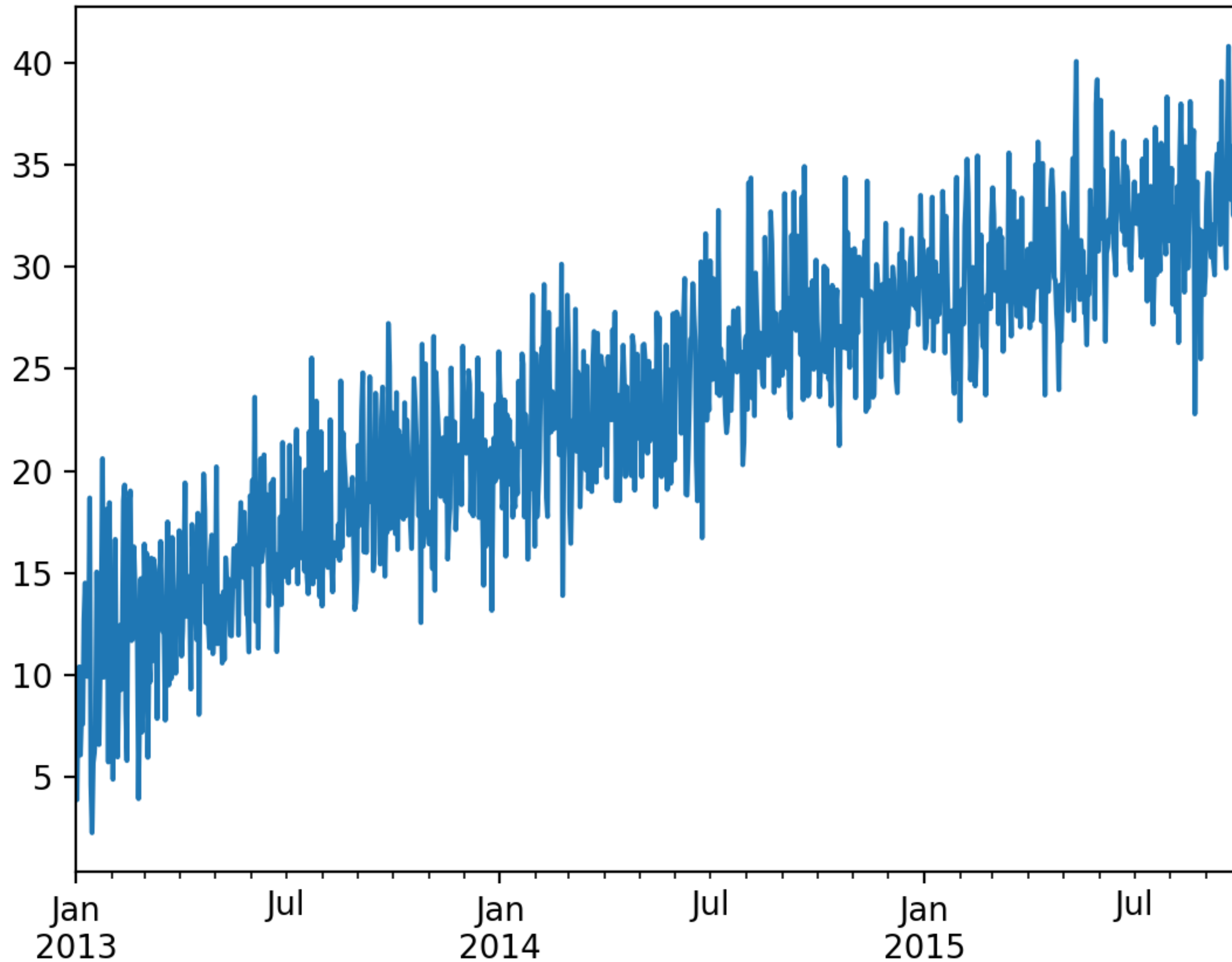
- Indexing and selection works as before
  - Create a **time series** with random component and a trend as sqrt.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

ts = pd.Series(
    3*np.random.randn(1000)
    +np.sqrt(np.linspace(100, 1100, 1000)),
 index = pd.date_range('1/1/2013', periods=1000))

ts.plot()
plt.show()
```

Time Series Indexing



Time Series Indexing

- We can access the value of the timeseries using a string

```
>>> ts['07-01-2014']
30.29958531626109

>>> ts['2014/07/01']
30.29958531626109
```

Time Series Indexing

- We can even slice with a partial date:

```
>>> ts['2015/01']
2015-01-01    29.018824
2015-01-02    26.018584
2015-01-03    26.322633
2015-01-04    29.884471
2015-01-05    30.861241
2015-01-06    27.578311
                 …
2015-01-28    31.386897
2015-01-29    34.393147
2015-01-30    24.519918
2015-01-31    28.588378
Freq: D, dtype: float64
```

Time Series Indexing

- We can even slice with a partial date

```
>>> ts['2013']
2013-01-01     6.579922
2013-01-02     3.886306
2013-01-03     8.624116
2013-01-04    10.406576
2013-01-05     6.078220
                 ...
2013-12-27    21.622836
2013-12-28    19.501775
2013-12-29    23.271005
2013-12-30    19.694463
2013-12-31    25.848146
Freq: D, Length: 365, dtype: float64
```

- Or can do range queries

```
>>> ts['2013/12/12' : '2014/1/3']
2013-12-12    22.549265
2013-12-13    25.545776
2013-12-14    17.717606
2013-12-15    17.886024
2013-12-16    23.793512
2013-12-17    17.883687
2013-12-18    14.392229
2013-12-19    21.520931
2013-12-20    16.309788
2013-12-21    20.269989
2013-12-22    20.047367
2013-12-23    20.333124
2013-12-24    21.111016
2013-12-25    13.164632
2013-12-26    19.161525
2013-12-27    21.622836
2013-12-28    19.501775
2013-12-29    23.271005
2013-12-30    19.694463
2013-12-31    25.848146
2014-01-01    23.794902
2014-01-02    23.508325
2014-01-03    18.175757
```

Time Series Indexing

- Works for data frames as well, but remember to use loc or iloc

```
>>> dfr.loc['05-05-2020' : '05-08-2020']
             values
2020-05-05   12.75
2020-05-06   12.25
2020-05-07   13.00
2020-05-08   10.50
```

Time Series Indexing

- Duplicate indices
  - Time series can have multiple observations per time-stamp
  - Use `index.is_unique` method on data frame to find out
  - We would get slices for non-unique indices
  - We can aggregate them using groupby

Time Series Date Ranges

- Can easily generate data ranges with `pd.date_range`
  - Can specify the frequency and number

```
>>> pd.date_range('2020-06-01', periods = 30, freq = 'W')
DatetimeIndex(['2020-06-07', '2020-06-14', '2020-06-21', '2020-06-28',
               '2020-07-05', '2020-07-12', '2020-07-19', '2020-07-26',
               '2020-08-02', '2020-08-09', '2020-08-16', '2020-08-23',
               '2020-08-30', '2020-09-06', '2020-09-13', '2020-09-20',
               '2020-09-27', '2020-10-04', '2020-10-11', '2020-10-18',
               '2020-10-25', '2020-11-01', '2020-11-08', '2020-11-15',
               '2020-11-22', '2020-11-29', '2020-12-06', '2020-12-13',
               '2020-12-20', '2020-12-27'],
              dtype='datetime64[ns]', freq='W-SUN')
```

Time Series Ranges

- Frequencies:
  - D — day
  - B — business day
  - H — hourly
  - min T — minute
  - M — month end
  - MS — month beginning
  - …

Time Series Ranges

- Can even get third Friday of each month

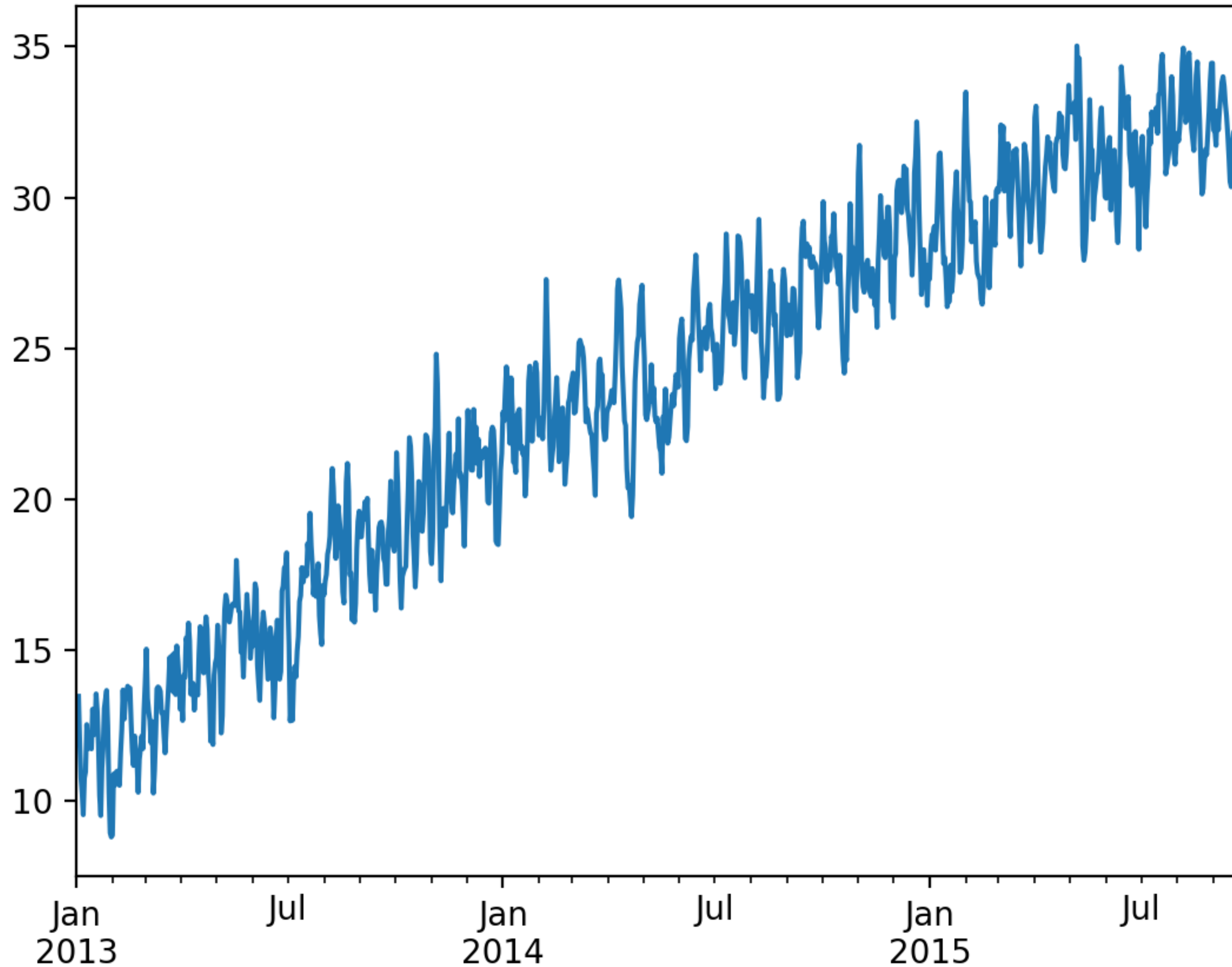
```
>>> pd.date_range('2020-06-28', freq = 'WOM-3FRI', periods = 20)
DatetimeIndex(['2020-07-17', '2020-08-21', '2020-09-18', '2020-10-16',
               '2020-11-20', '2020-12-18', '2021-01-15', '2021-02-19',
               '2021-03-19', '2021-04-16', '2021-05-21', '2021-06-18',
               '2021-07-16', '2021-08-20', '2021-09-17', '2021-10-15',
               '2021-11-19', '2021-12-17', '2022-01-21',
'2022-02-18'],
              dtype='datetime64[ns]', freq='WOM-3FRI')
```

Time Series Shifting

- Shifting data — move data backward or forward
  - Can do so be ourselves
  - Or use shift

```
>>> smoother = 0.1*ts.shift(-2) + 0.2*ts.shift(-1) + 0.4*ts
0.2*ts.shift(1)+0.1*ts.shift(2)
>>> smoother.plot()
<matplotlib.axes._subplots.AxesSubplot object at
0x7f9c72f47ee0>
>>> plt.show()
```

Time Series Shifting

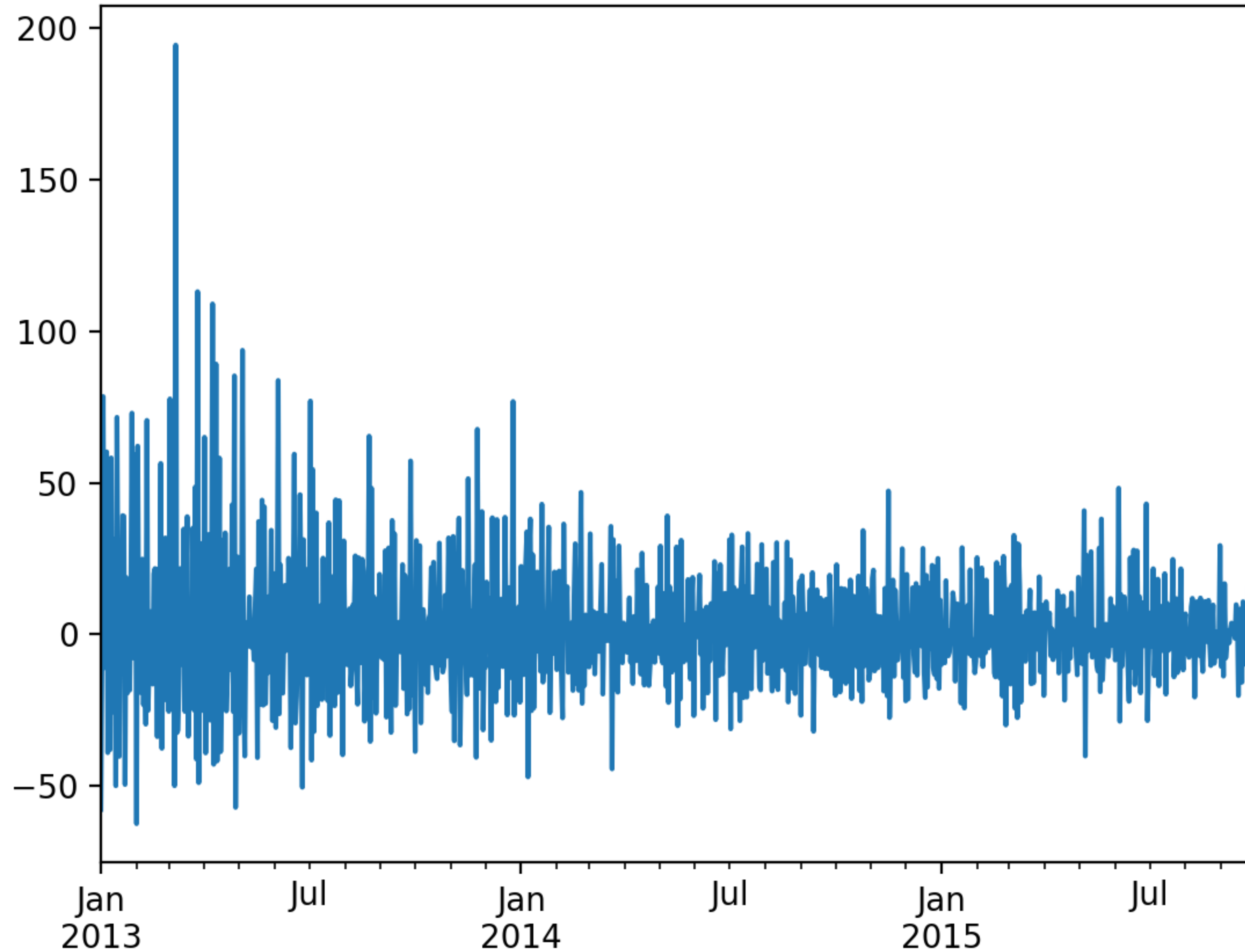


Time Series Shifting

- Using percentage changes

```
>>> plt.show()
>>> pct = (ts/ts.shift(-1)-1)*100
>>> pct.plot()
<matplotlib.axes._subplots.AxesSubplot object at
0x7f9c72f28760>
>>> plt.show()
```

Time Series Shifting



Time Series Shifting

- We can use offsets with shifting
  - Need to import the timestamp objects:
    - `from pandas.tseries.offsets import Day`
    - Can use arithmetic:

```
>>> pd.Timestamp.now()
Timestamp('2020-06-30 16:30:49.660811')
>>> pd.Timestamp.now()+3*Day()
Timestamp('2020-07-03 16:30:51.640833')
```

Time Zones

- Very unpleasant to deal with different time zones
  - Easier to use Coordinated Universal Time (UTC)
    - Successor to Greenwich Mean Time
  - Python: Use Olson database
    - Need to install pytz with pip

Time Zones

```
>>> kol = pd.date_range('3/9/2020 9:30', periods = 10, freq = 'H',
tz=pytz.timezone('Asia/Kolkata'))

>>> kol.tz_convert('America/Chicago')
DatetimeIndex(['2020-03-08 23:00:00-05:00', '2020-03-09 00:00:00-05:00',
               '2020-03-09 01:00:00-05:00', '2020-03-09 02:00:00-05:00',
               '2020-03-09 03:00:00-05:00', '2020-03-09 04:00:00-05:00',
               '2020-03-09 05:00:00-05:00', '2020-03-09 06:00:00-05:00',
               '2020-03-09 07:00:00-05:00', '2020-03-09 08:00:00-05:00'],
              dtype='datetime64[ns, America/Chicago]', freq='H')
```

Time Zones

- Many countries outside the tropics have daylight savings or summer times
  - This year: change on November 1, 2020

```
pd.date_range('10/31/2020 20:00', periods = 10, freq = 'H',
tz=pytz.timezone('America/Chicago'))
DatetimeIndex(['2020-10-31 20:00:00-05:00', '2020-10-31 21:00:00-05:00',
               '2020-10-31 22:00:00-05:00', '2020-10-31 23:00:00-05:00',
               '2020-11-01 00:00:00-05:00', '2020-11-01 01:00:00-05:00',
               '2020-11-01 01:00:00-06:00', '2020-11-01 02:00:00-06:00',
               '2020-11-01 03:00:00-06:00', '2020-11-01 04:00:00-06:00'],
              dtype='datetime64[ns, America/Chicago]', freq='H')
```

Time Zones

- Many countries outside the tropics have daylight savings or summer times
    - This year: change on November 1, 2020

```
>>> from pandas.tseries.offsets import Hour
>>> pd.Timestamp('10/31/2020 20:00') + 5*Hour()
Timestamp('2020-11-01 01:00:00')
```

Periods

- Periods represents time spans
  - Days
  - Months
  - Quarters (with variously defined business quarters)
    - Q-Jan, Q-Feb, Q-MAR, … —> last calendar day of each month
    - BQ-Jan, BQ-Feb, …
    - QS-Jan, QS-Feb, … —> beginning quarter
    - BQS-Jan, BQS-Feb, …
  - years: A - calendar year end, BA business year end,

Periods

- Examples:

```
>>> pd.date_range('2020', periods = 5, freq = 'QS')
DatetimeIndex(['2020-01-01', '2020-04-01',
'2020-07-01', '2020-10-01', '2021-01-01'],
          dtype='datetime64[ns]', freq='QS-JAN')
```

```
>>> pd.date_range('2020', periods = 5,
                  freq = 'BQS-MAR')

DatetimeIndex(['2020-03-02', '2020-06-01',
  '2020-09-01', '2020-12-01', '2021-03-01'],
          dtype='datetime64[ns]', freq='BQS-MAR')
```

Resampling

- Resampling: convert a time series from one frequency to another
  - Down-sampling: Aggregating higher frequency data to lower frequency
  - Up-sampling: Converting lower frequency to higher frequency

Resampling

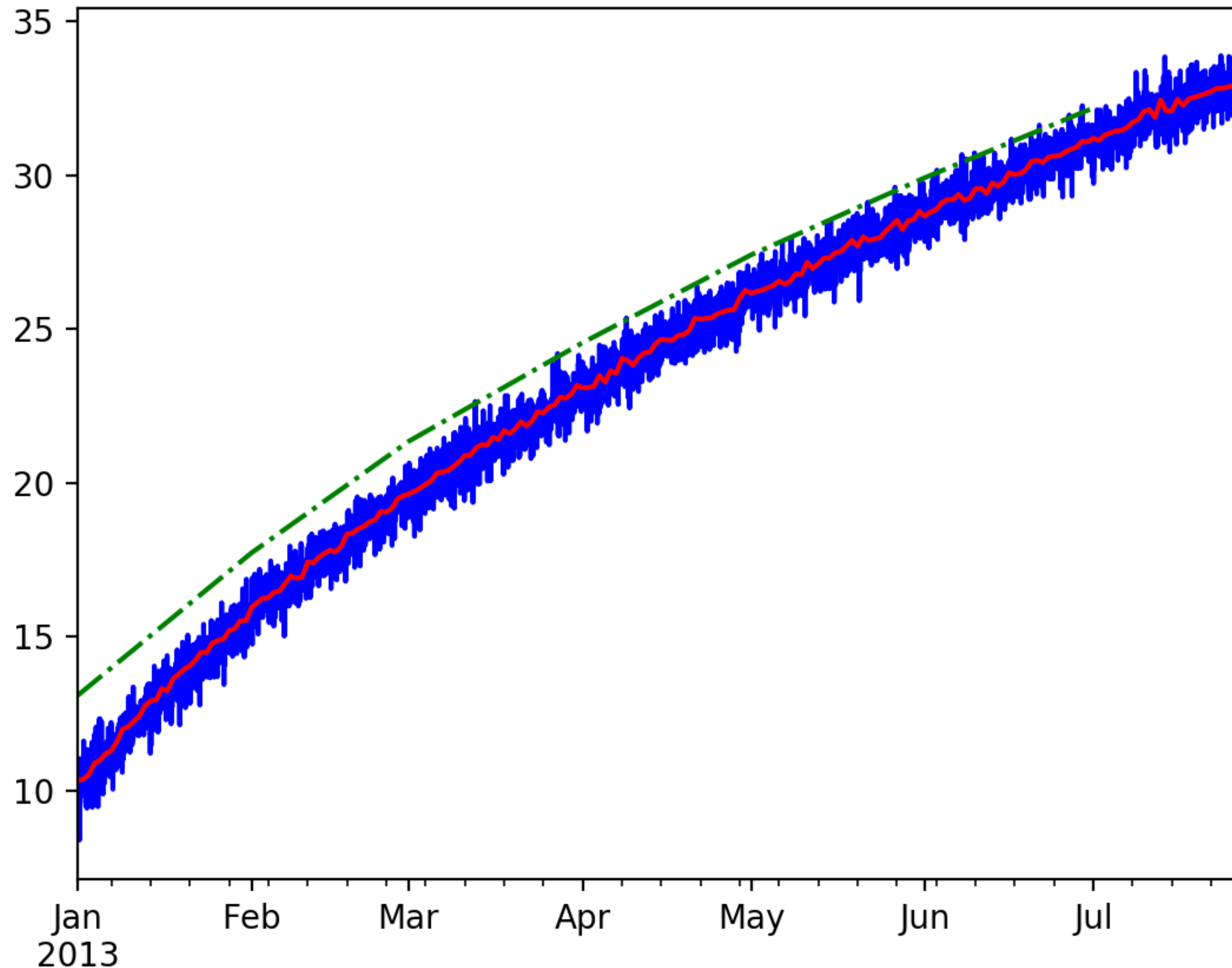
- Pandas has `resample`
  - Need an aggregation function
  - Example:
    - Create time series with hourly data
    - Down-sample to hours and days
      - Down-samples go to the beginning

Resampling

```
ts = pd.Series(0.5*np.random.randn(5000)+
          np.sqrt(np.linspace(100, 1100, 5000)),
               index = pd.date_range('1/1/2013',
               periods=5000, freq='H'))
ax = ts.plot(style = 'b-')
ts = ts.resample('D').mean()
ts.plot( style='r-', ax=ax)
ts = ts.resample('M').mean()
ts.plot(style = 'g-.', ax= ax)
plt.show()
```

Resampling

- The monthly sampling starts January 1

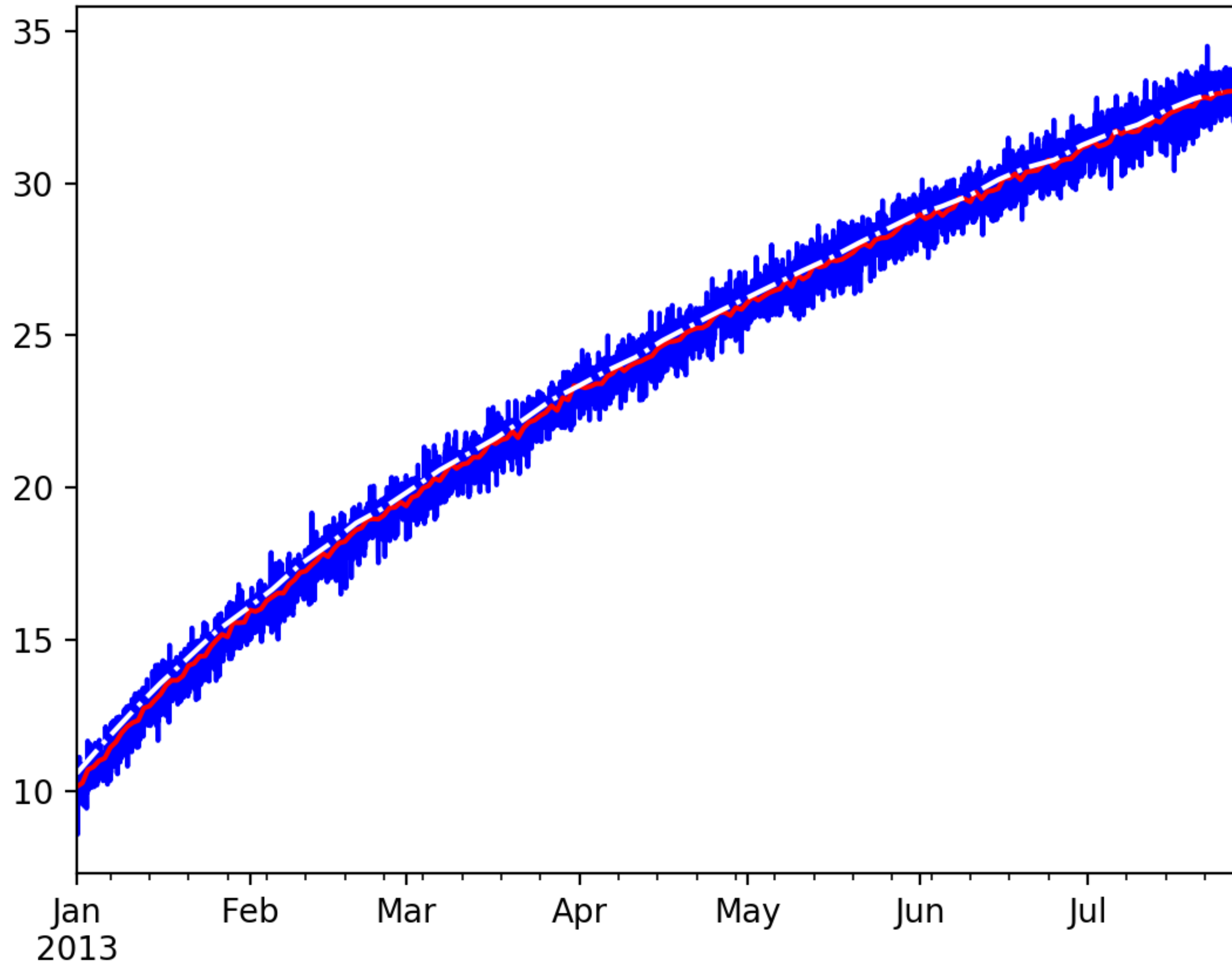


Resampling

- We can use scalars for the resample arguments

```
ts = ts.resample('5D').mean()
ts.plot(style = 'w-.', ax= ax)
```

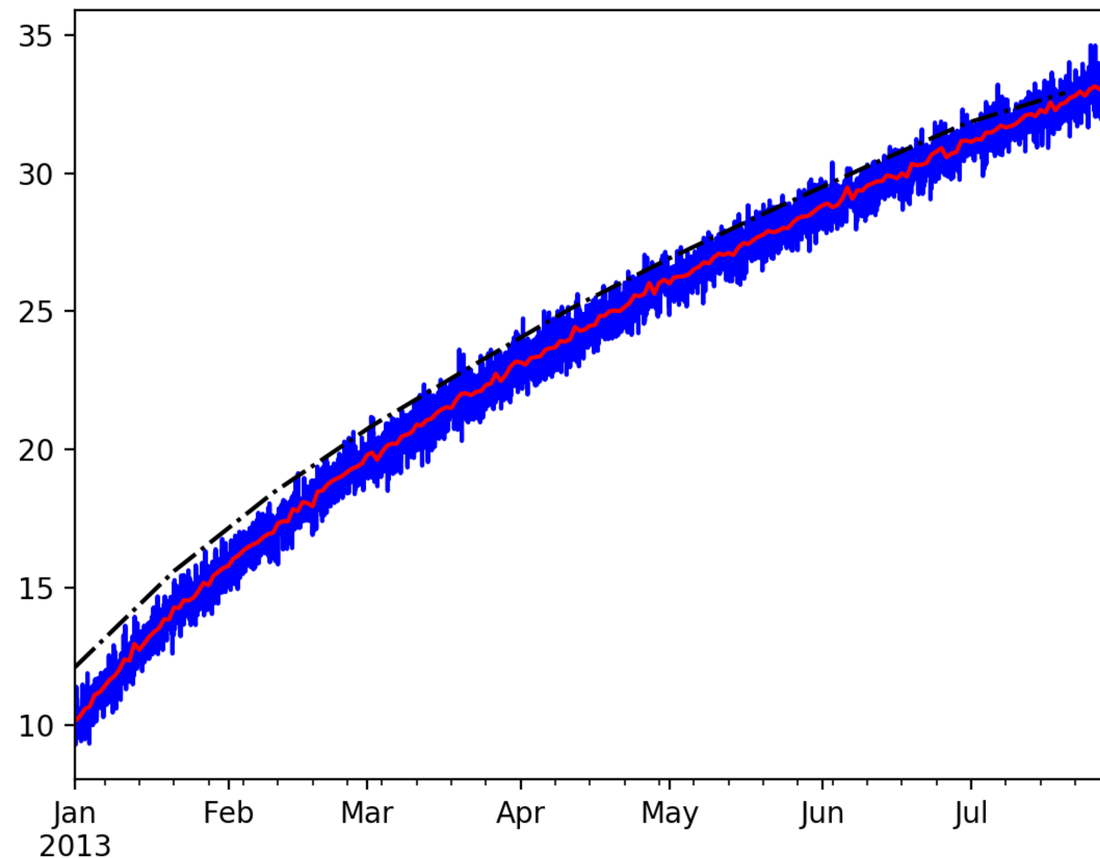
Resampling

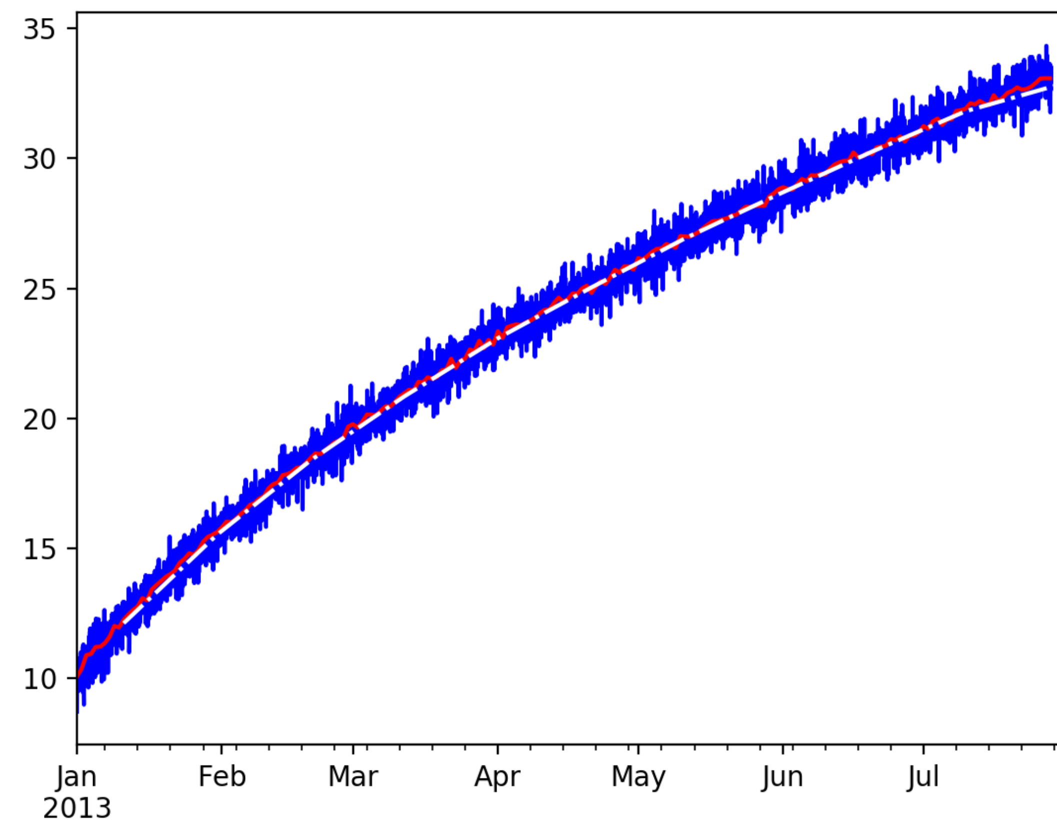


Resampling

- Can use loffset to shift the sample

```
ts = ts.resample('20D', loffset='10D').mean()
ts.plot(style = 'w-.', ax= ax)
```





Resampling

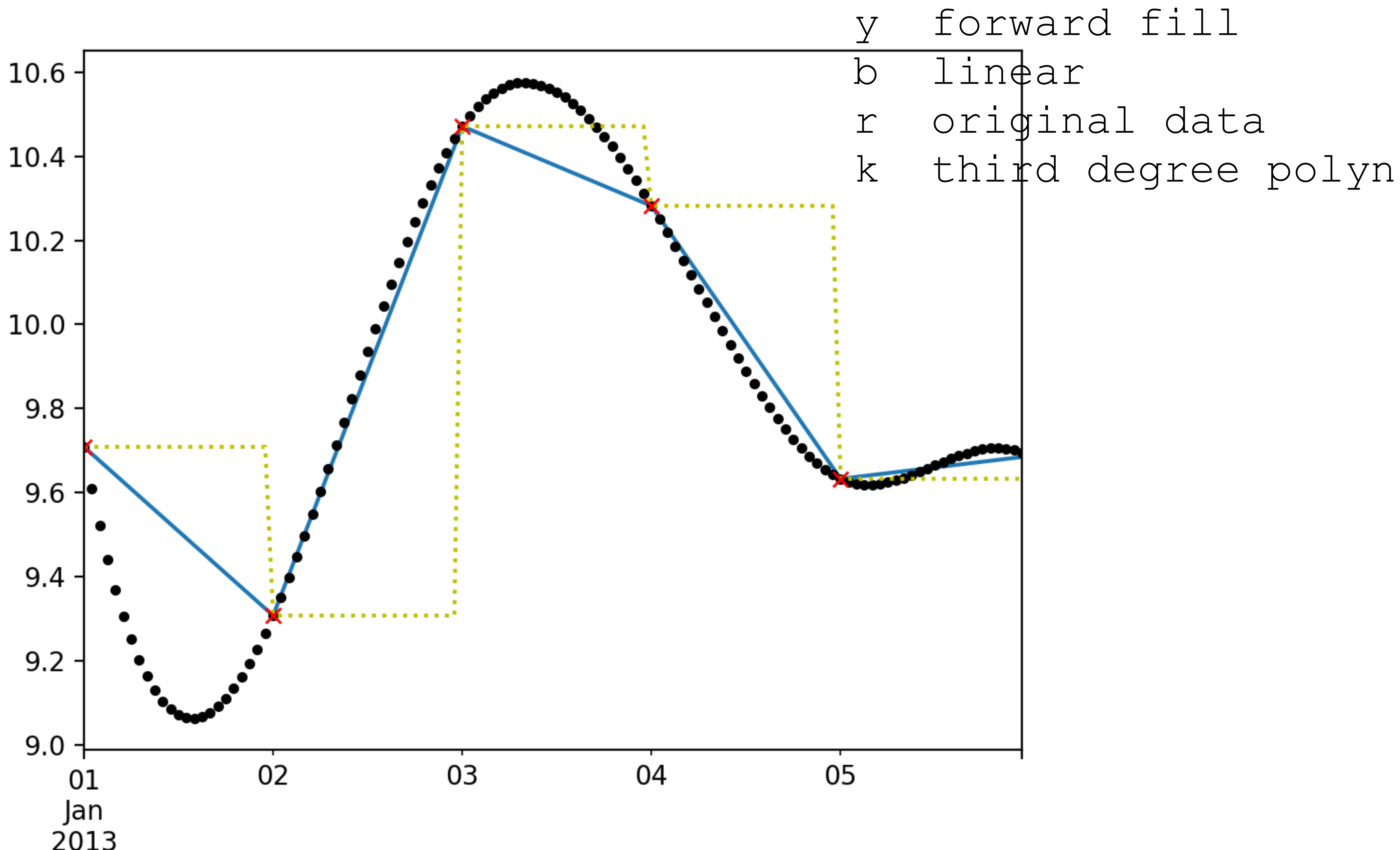
- When up sampling, we do not aggregate but need to interpolate
    - ffill - forward fill
    - bfill -backward fill
    - interpolate
        - needs method and sometimes degree

Resampling

- Example:

```
tshourly = ts.resample('H').interpolate(method='linear')
tshourly2 =
ts.resample('H').interpolate(method='polynomial',
                                                    order=3)
tshourly3 = ts.resample('H').ffill()
ax = tshourly[:5*24].plot()
tshourly2[:5*24].plot(style='k.', ax = ax)
tshourly3[:5*24].plot(style='y:', ax = ax)
ts[:5].plot(style = 'rx', ax = ax)
```

Resampling



Moving Windows

- An important type of resampling is using moving windows
- `rolling` uses a sliding window
- Together with mean, this provides a way to see a trend

Moving Windows

- Example:
  - min_periods=10: start calculating whenever you have 10 values
  - center=True: place value in the center of sliding window

```
rol = ts.rolling(250, min_periods=10).mean()
rol2 = ts.rolling(250, min_periods=10,
center=True).mean()
myax = ts.plot(style = 'b-')
rol.plot(style = 'r-', ax = myax)
rol2.plot(style = 'k-.', ax = myax)
```

Moving Windows

- rolling window can also accept a fixed-size time offset
  - `rolling('20D')`

Moving Windows

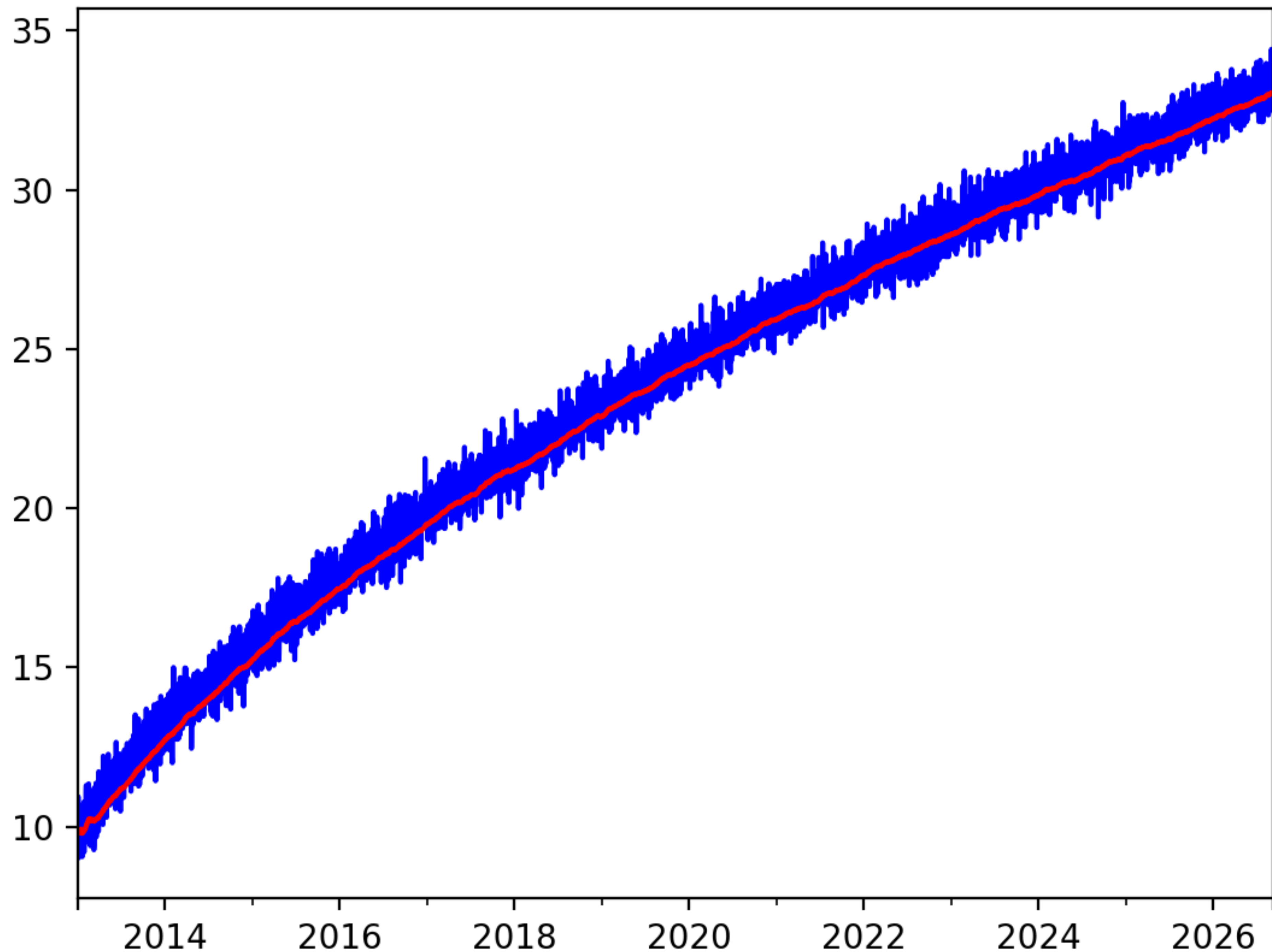
- All data points in a sliding window contribute equally to the aggregator
- Can also use exponential decay
- Pandas has ewm
  - With span parameter for half the size

Moving Windows

- Example

```
rol = ts.ewm(span=250//2, min_periods=10).mean()
myax = ts.plot(style = 'b-')
rol.plot(style = 'r-', ax = myax)
```

Moving Windows



Pandas Time Series

- Time series need and have their own transformation tools
- Next steps
  - theory of time series
  - recognize static time series
  - learn how to use auto-correlations

Loading Time Series

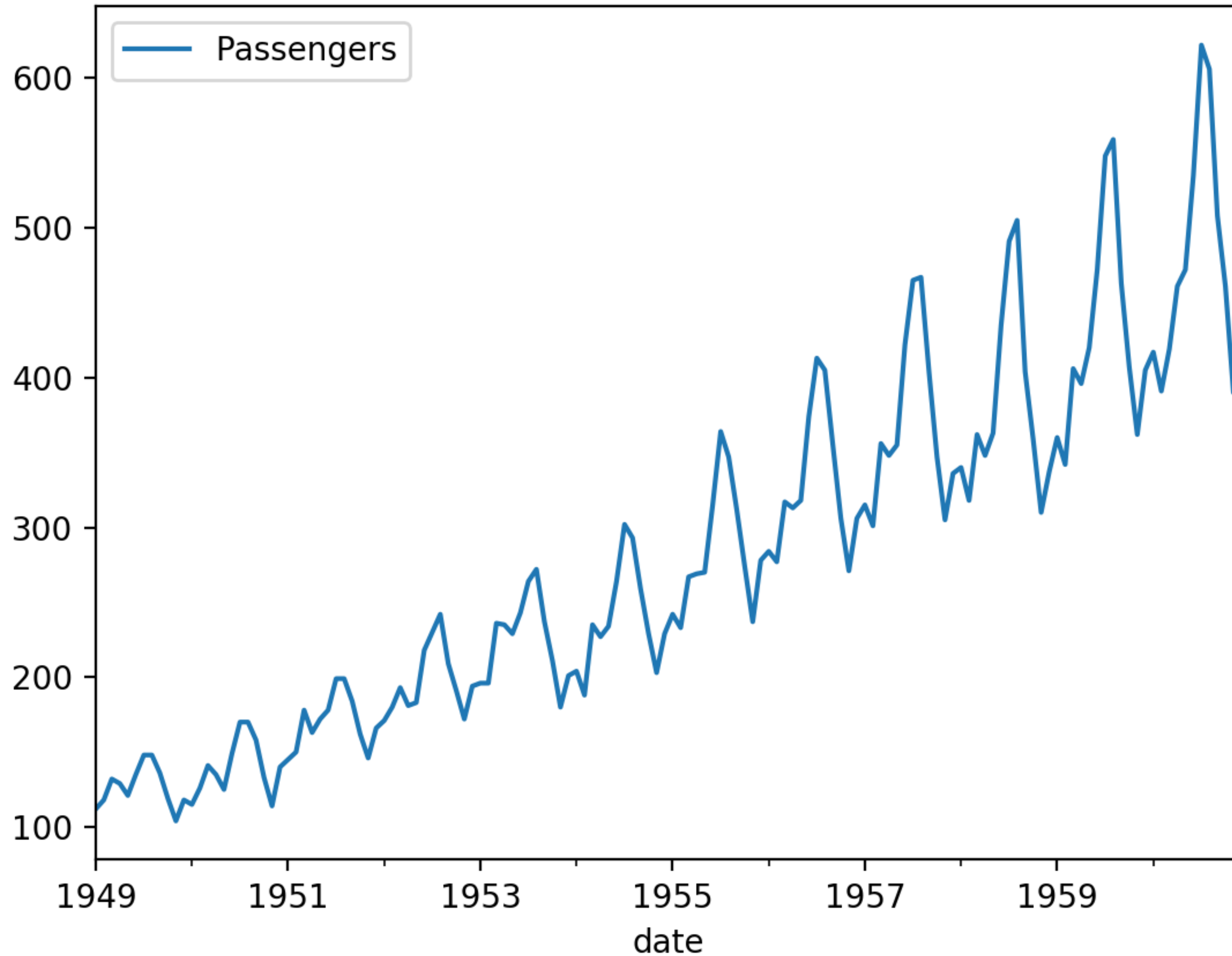
- Pandas can use `parse_dates` in order to extract date information
  - Example: Airline data
    - Has a column called 'date' with month and year

```
                Passengers
date
1949-01-01             112
1949-02-01             118
1949-03-01             132
1949-04-01             129
1949-05-01             121
```

Loading Time Series

```
def get_data_1():
    df = pd.read_csv('airline-passengers.csv',
                     parse_dates= {'date' : ["Month"]})
    df.set_index('date', inplace=True)
    return df
```

Loading Time Series



Loading Time Series

- The date information can also be split over several columns

```
YEAR,LOCATION,STATE ANSI,ASD CODE,COUNTY
ANSI,REFERENCE PERIOD,COMMODITY,"DRY, NONFAT,
HUMAN in LB","SKIM, UNSWEETENED, CONDENSED in
LB"
2016,CENTRAL,  ,  ,   ,JAN,MILK,"25,815,000",
2016,CENTRAL,  ,  ,   ,FEB,MILK,"23,976,000",
2016,CENTRAL,  ,  ,   ,MAR,MILK,"30,956,000",
2016,CENTRAL,  ,  ,   ,APR,MILK,"30,393,000",
2016,CENTRAL,  ,  ,   ,MAY,MILK,"37,183,000",
2016,CENTRAL,  ,  ,   ,JUN,MILK,"29,760,000",
2016,CENTRAL,  ,  ,   ,JUL,MILK,"26,698,000",
```

Loading Time Series

```
def get_data_2():
    df = pd.read_csv('MILK.csv',
        parse_dates= {'date' : ["YEAR", "REFERENCE PERIOD"]})
    df.set_index('date', inplace=True)
    return df
```

Loading Time Series

```
df = get_data_2()

df['DRY, NONFAT, HUMAN in LB'] = df['DRY, NONFAT, HUMAN 
in LB'].str.replace(',','').astype(float)

df['DRY, NONFAT, HUMAN in LB'].plot()
```

Loading Time Series