

K Ö N I G S W E G

BUDAPEST | BI FORUM  
October 25-27, 2016

# Introduction to Pandas and Time Series Analysis

Alexander C. S. Hendorf





**Alexander C. S. Hendorf**  
**Königsweg GmbH**

Königsweg affiliate high-tech startups and the industry

EuroPython Organisator + Programm Chair  
mongoDB master 2016, MUG Leader  
Speaker mongoDB days, EuroPython, PyData...





## Origin und Goals

- Open Source Python Library
- practical real-world data analysis - fast, efficient & easy
- gapless workflow (no switching to e.g. R)
- 2008 started by Wes McKinney,  
now PyData stack at Continuum Analytics ("Anaconda")
- very stable project with regular updates
- <https://github.com/pydata/pandas>



## Main Features

- Support for CSV, Excel, JSON, SQL, SAS, clipboard, HDF5,...
- Data cleansing
- Re-shape & merge data (joins & merge) & pivoting
- Data Visualisation
- Well integrated in Jupyter (iPython) notebooks
- Database-like operations
- Performant



# Today

Part 1:

**Basic functionality of Pandas**

Teil 2:

**Time series analysis with Pandas**

Git featuring this presentation's code examples:

<https://github.com/Koenigsweg/data-timeseries-analysis-with-pandas>

2014-08-21T22:50:00,12.0  
2014-08-17T13:20:00,16.0  
2014-08-06T01:20:00,14.0  
2014-09-27T06:50:00,11.0  
2014-08-25T21:50:00,13.0  
2014-08-14T05:20:00,13.0  
2014-09-14T05:20:00,16.0  
2014-08-03T02:50:00,21.0  
2014-09-29T03:00:00,13  
2014-09-06T08:20:00,16.0  
2014-08-19T07:20:00,13.0  
2014-09-27T22:50:00,10.0  
2014-08-28T08:20:00,12.0  
2014-08-17T01:00:00,14  
2014-09-27T14:00:00,17  
2014-09-10T18:00:00,18  
2014-09-22T23:00:00,8  
2014-09-20T03:00:00,9  
2014-08-29T09:50:00,16.0  
2014-08-16T01:50:00,13.0  
2014-08-28T22:00:00,14

```
In [1]: import pandas as pd
```

```
In [4]: # read data from file
# Aarhus, Denmark (Open Data Aarhus)

df = pd.read_csv('raw_weather_data_aug_sep_2014/tempm.csv', header=None)
df.head(5)
```

Out[4]:

|   | 0                   | 1    |
|---|---------------------|------|
| 0 | 2014-09-26T03:50:00 | 14.0 |
| 1 | 2014-08-10T05:00:00 | 14.0 |
| 2 | 2014-08-21T22:50:00 | 12.0 |
| 3 | 2014-08-17T13:20:00 | 16.0 |
| 4 | 2014-08-06T01:20:00 | 14.0 |

```
In [ ]:
```



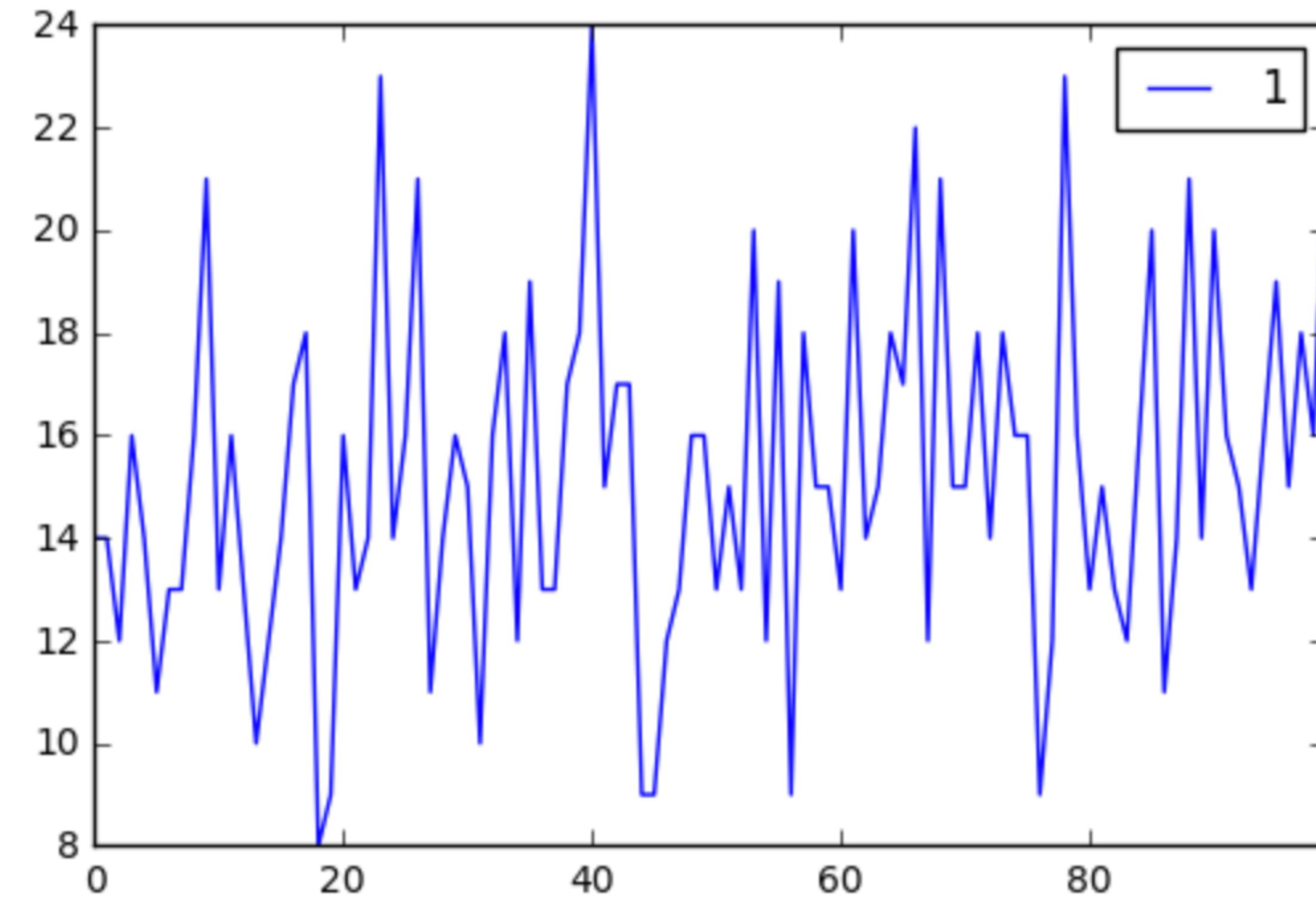
## I/O and viewing data

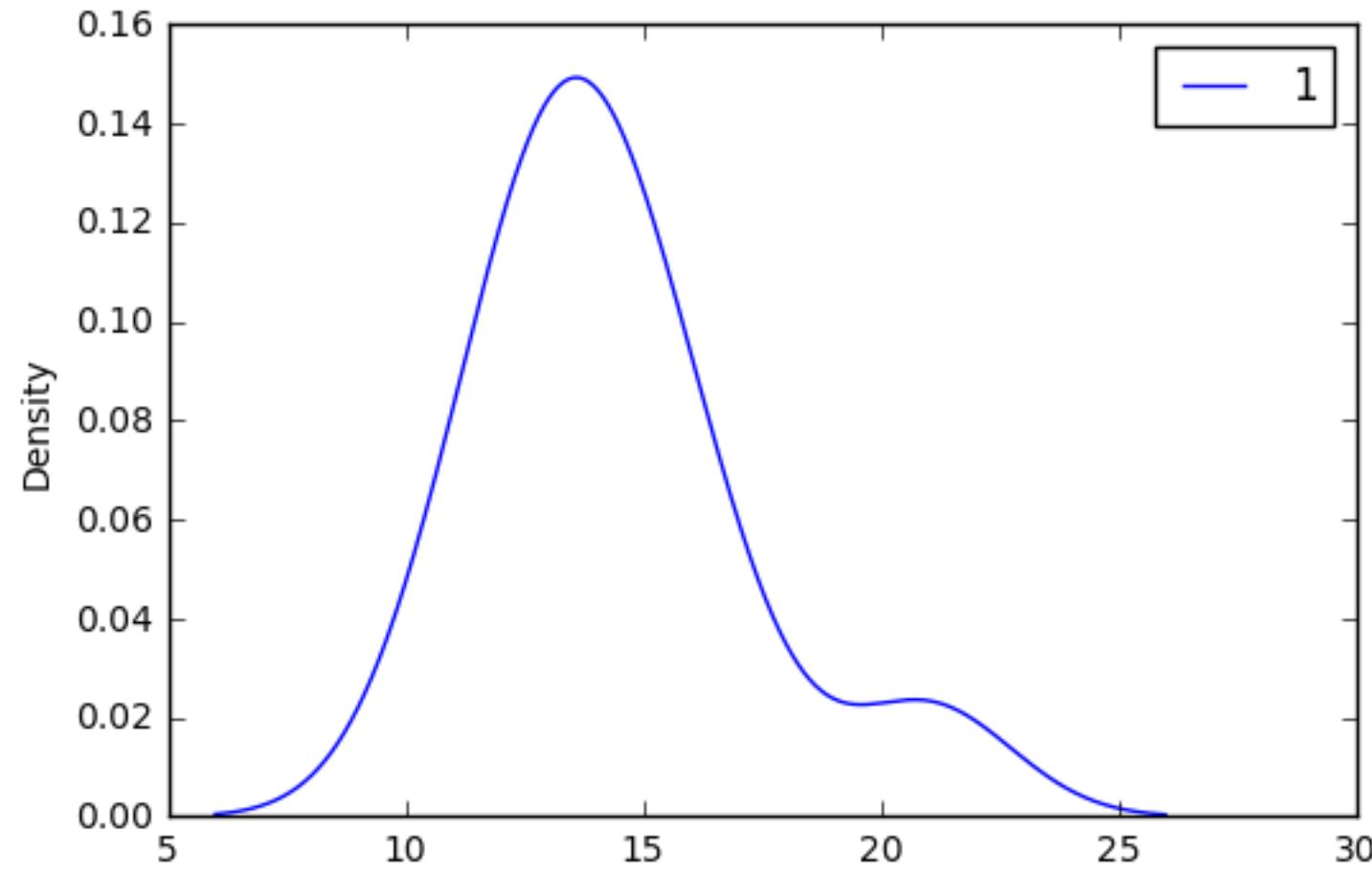
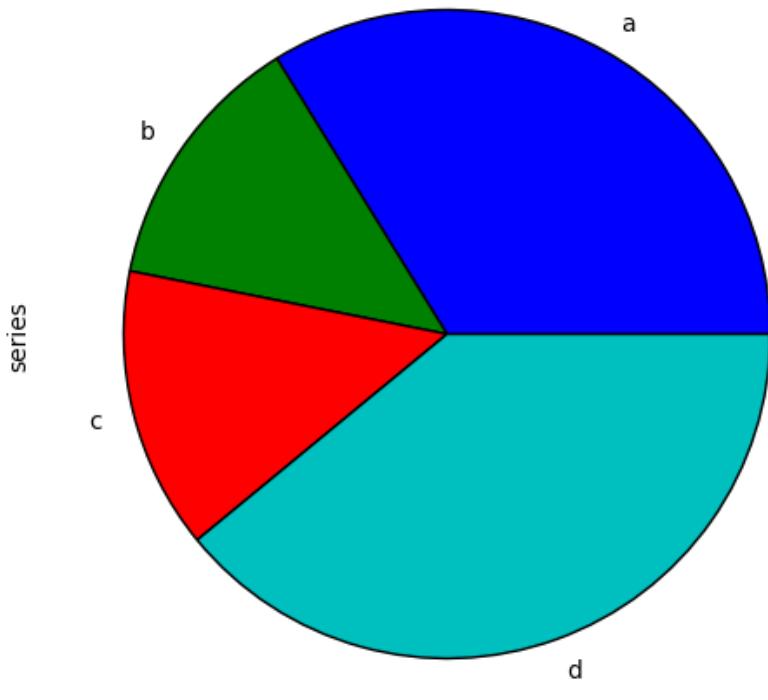
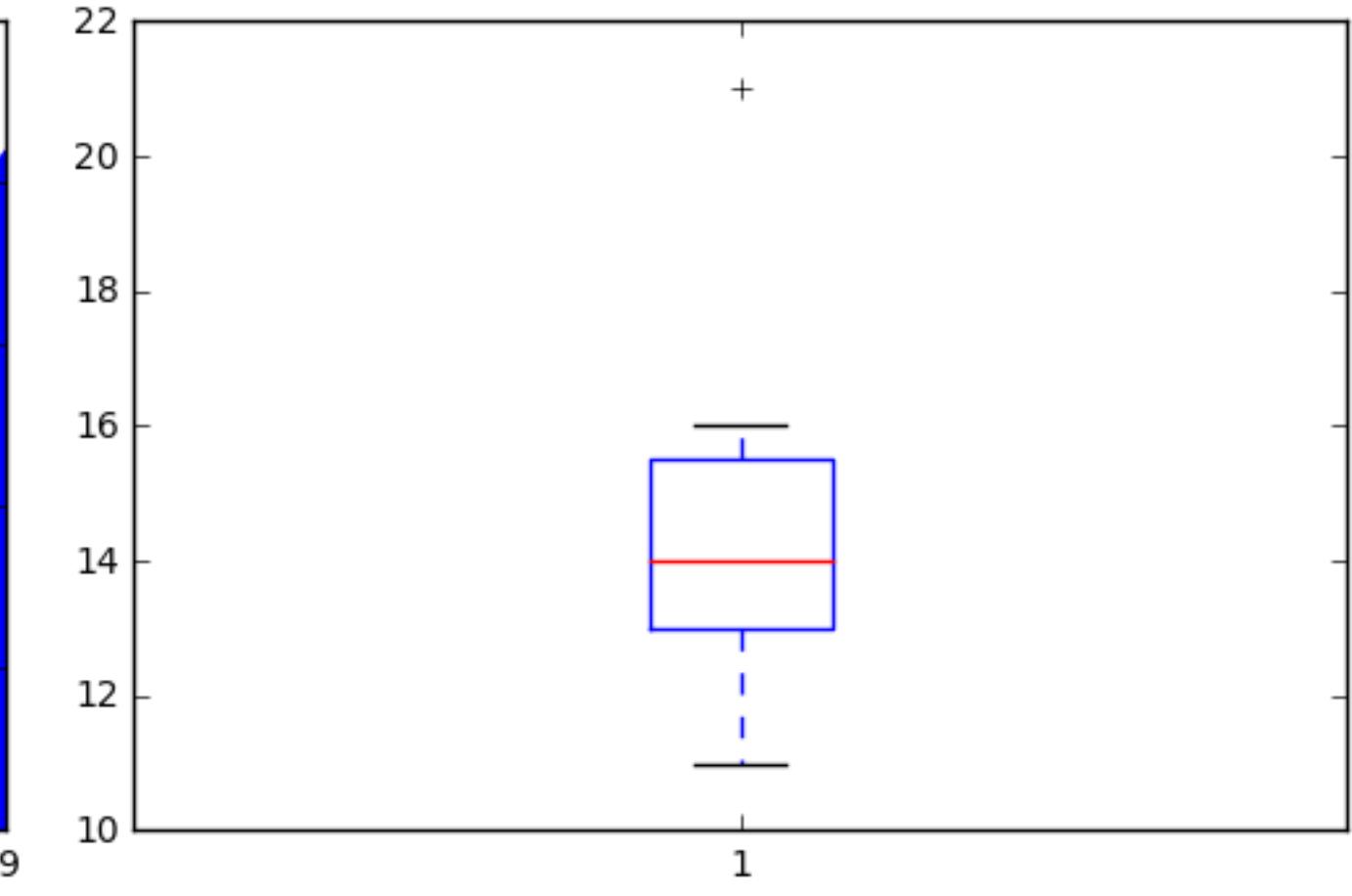
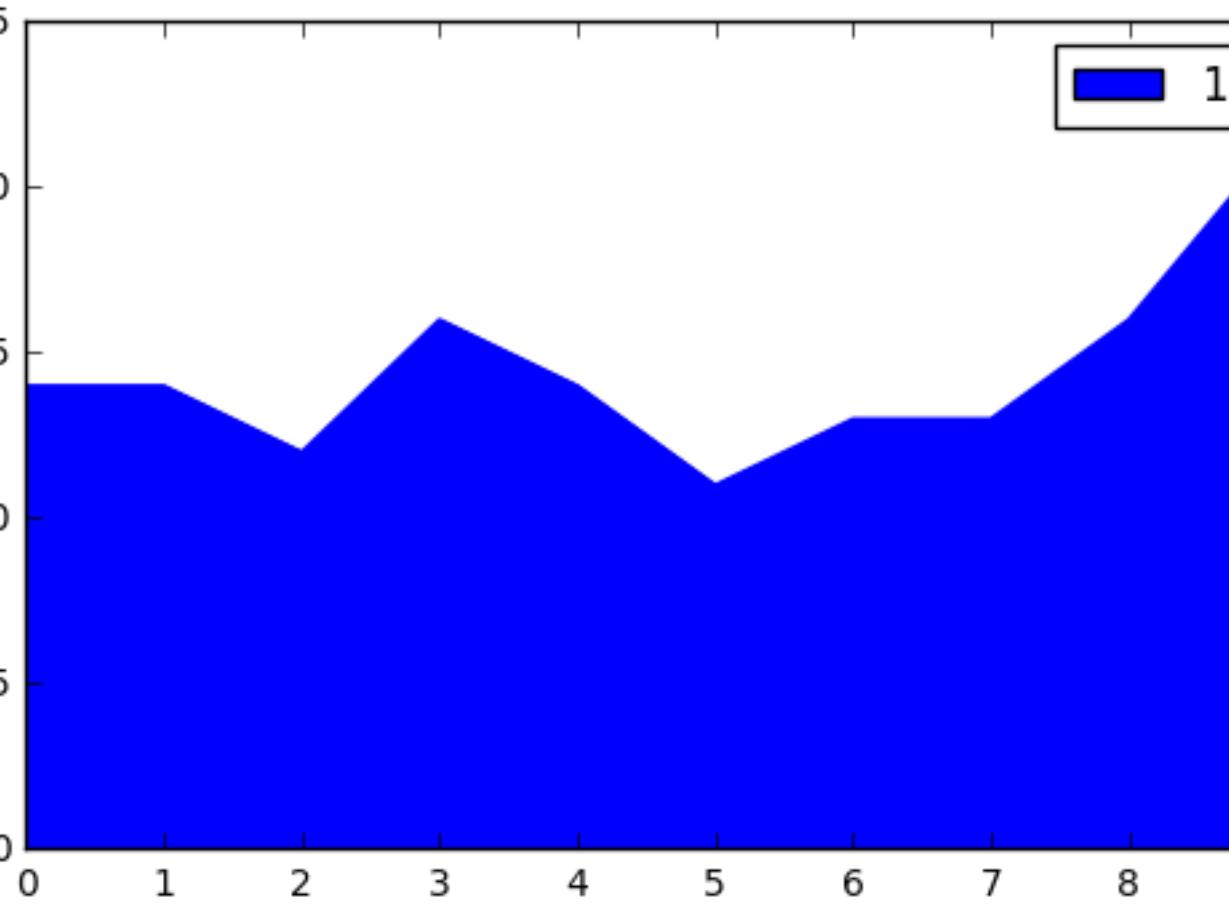
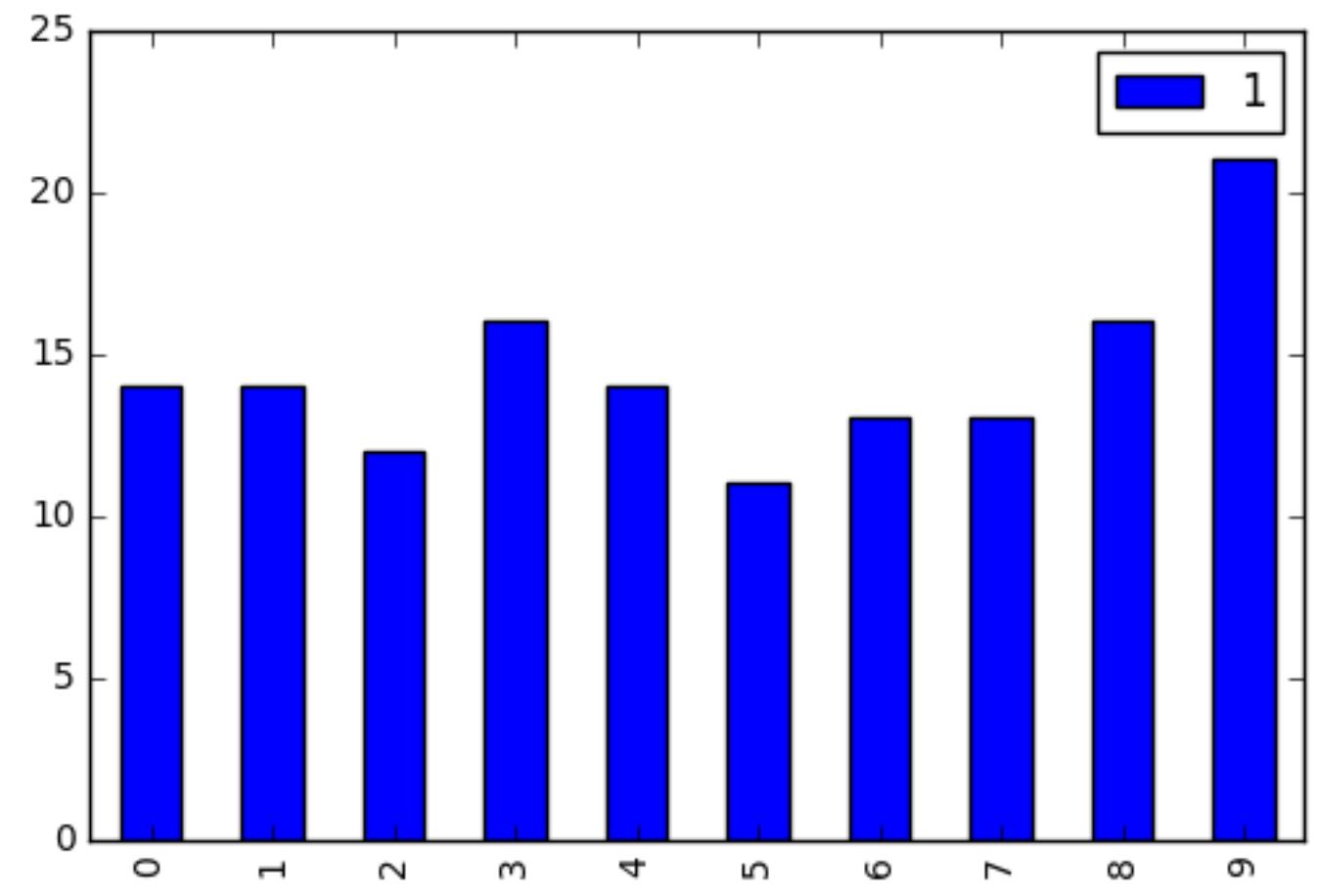
- convention    `import pandas as pd`
- example        `pd.read_csv()`
  - very flexible, ~40 *optional* parameters included (*delimiter*, *header*, *dtype*, *parse\_dates*,...)
- preview data with `.head(#number of lines)` and `.tail(#)`

```
In [4]: import matplotlib  
%matplotlib inline
```

```
In [11]: df[:100].plot()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x10bb13f98>
```

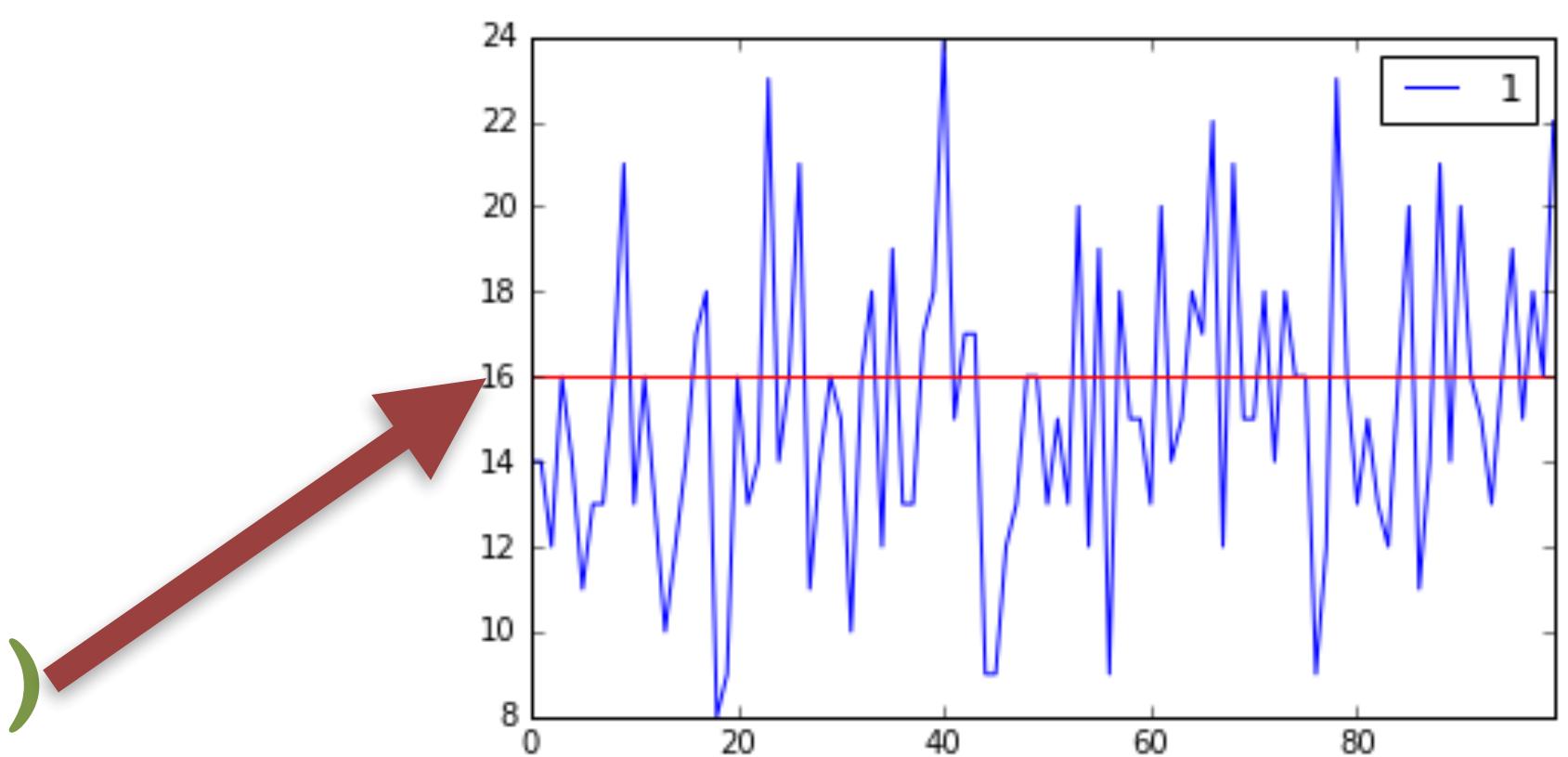




`df.plot(kind='bar')`

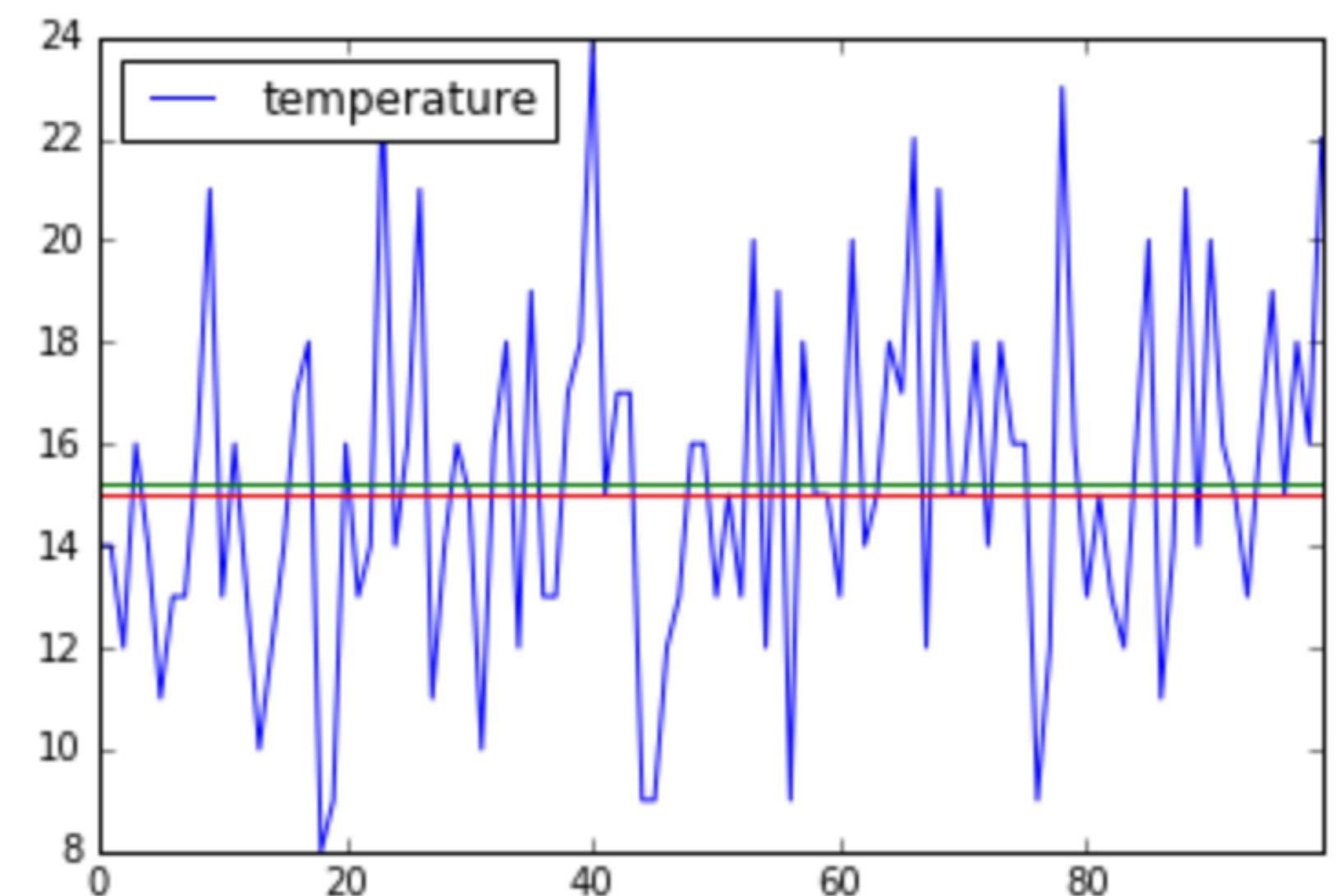
`ax = df[:100].plot()`

`ax.axhline(16, color='r', linestyle=' - ')`

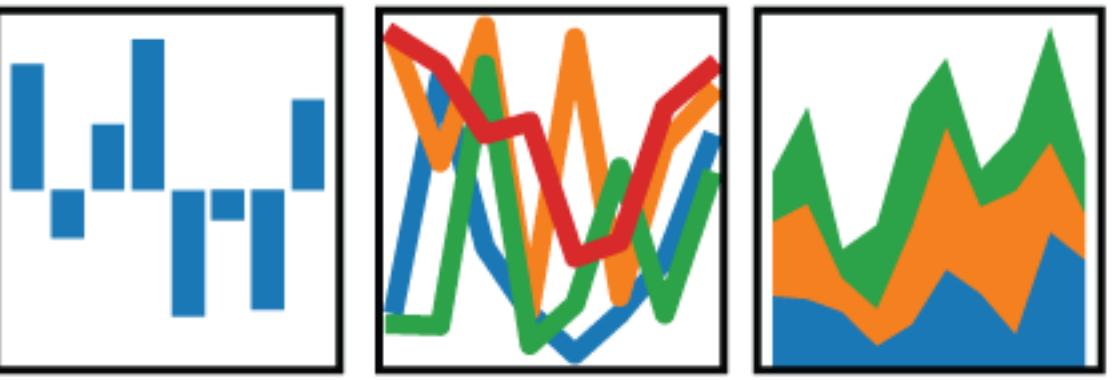


```
In [56]: ax = df[:100].plot()  
ax.axhline(df[:100]['temperature'].median(), color='r', linestyle='--')  
ax.axhline(df[:100]['temperature'].mean(), color='g', linestyle='--')
```

```
Out[56]: <matplotlib.lines.Line2D at 0x116c0bbe0>
```

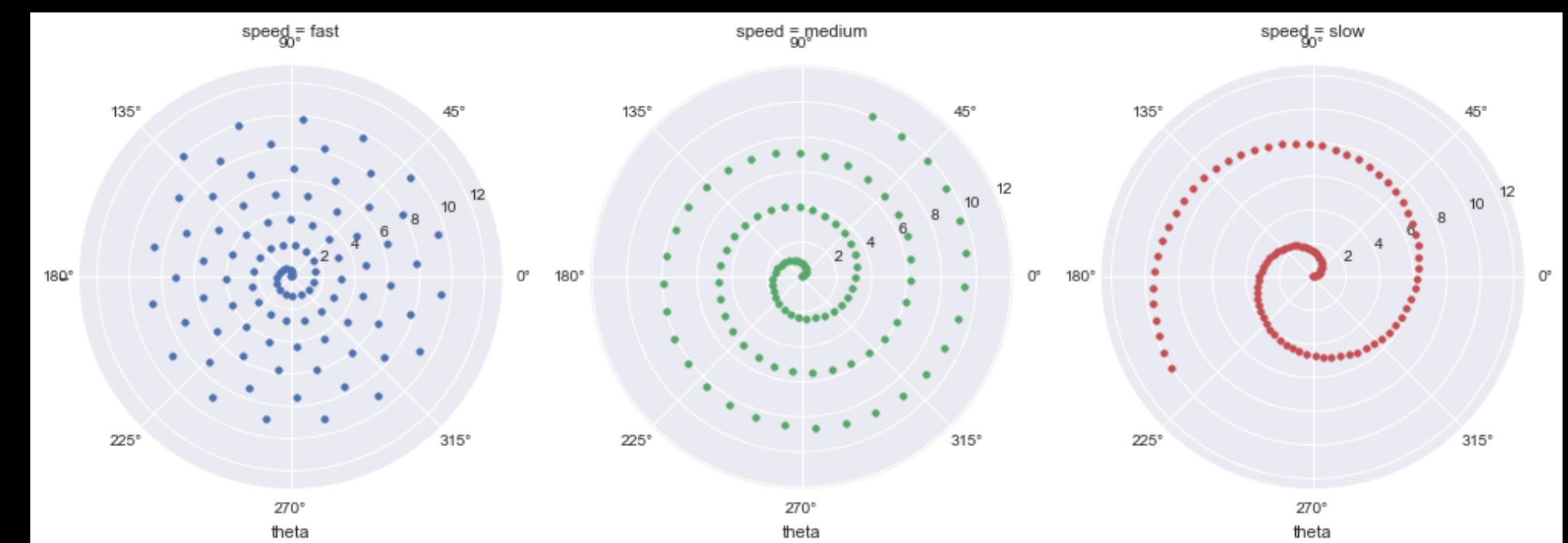
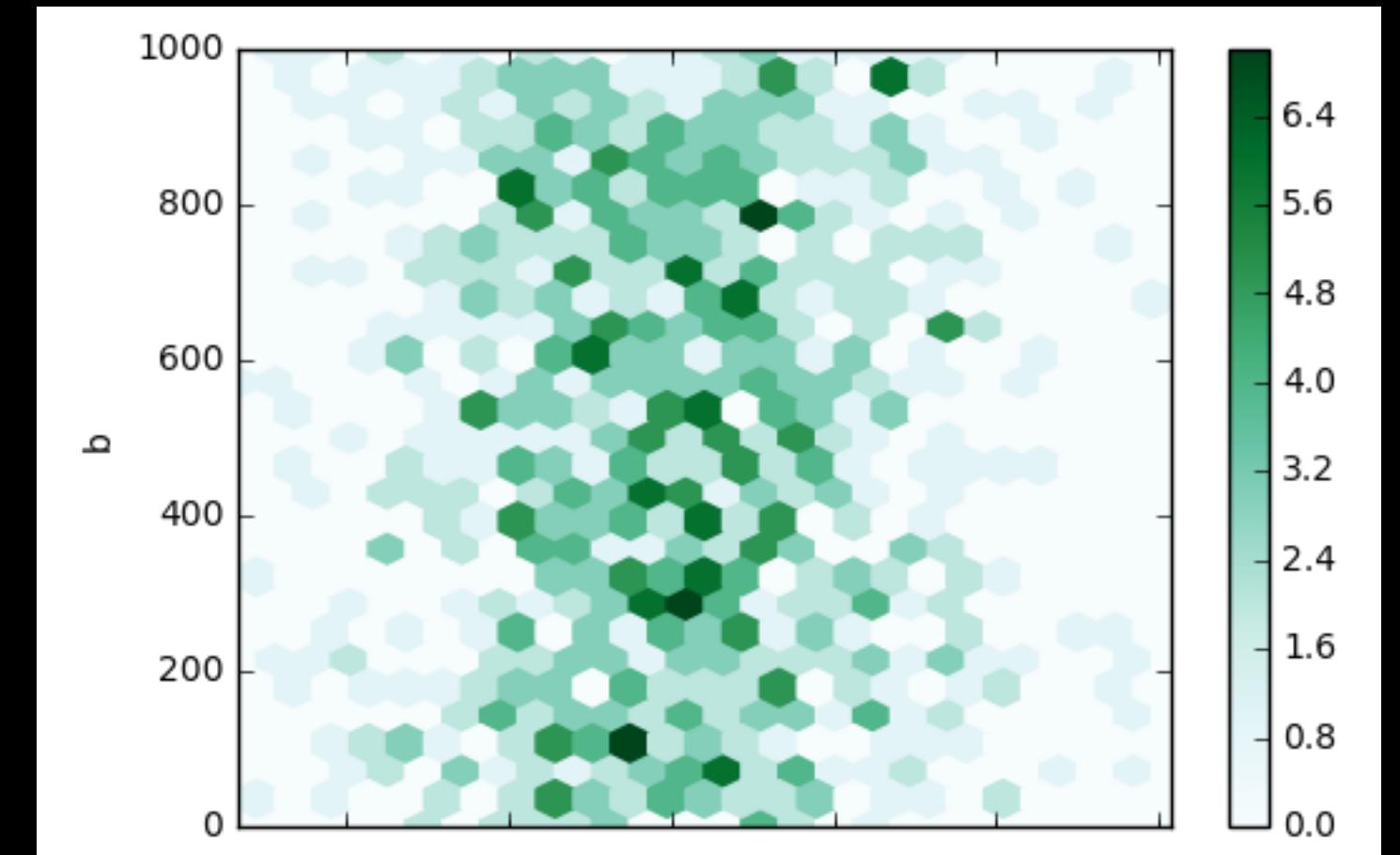


pandas  
 $y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$



## Visualisation

- matplotlib (<http://matplotlib.org>) integrated, `.plot()`
- custom- and extendable, `plot()` returns `ax`
- Bar-, Area-, Scatter-, Boxplots u.a.

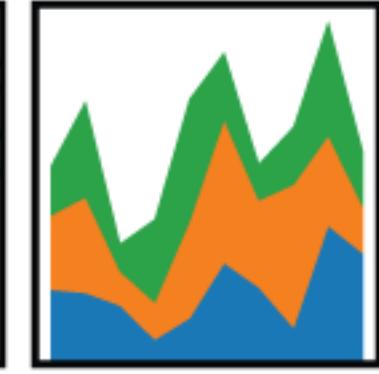
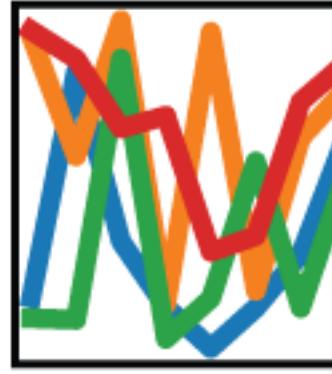


Bokeh (<http://bokeh.pydata.org/en/latest/>)

Seaborn (<https://stanford.edu/~mwaskom/software/seaborn/index.html>)

# pandas

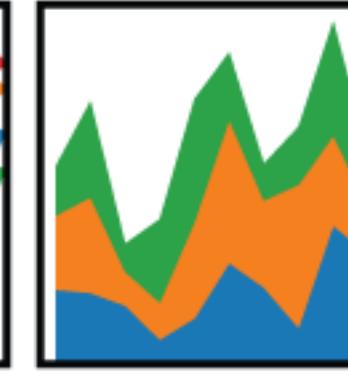
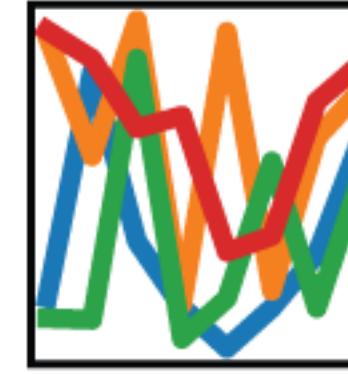
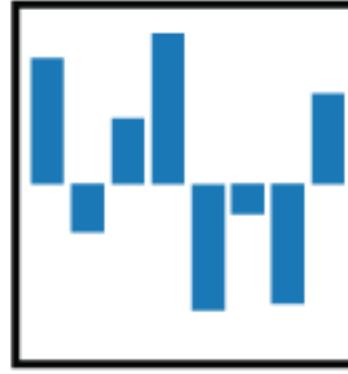
$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$



## Structure

# pandas

$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$



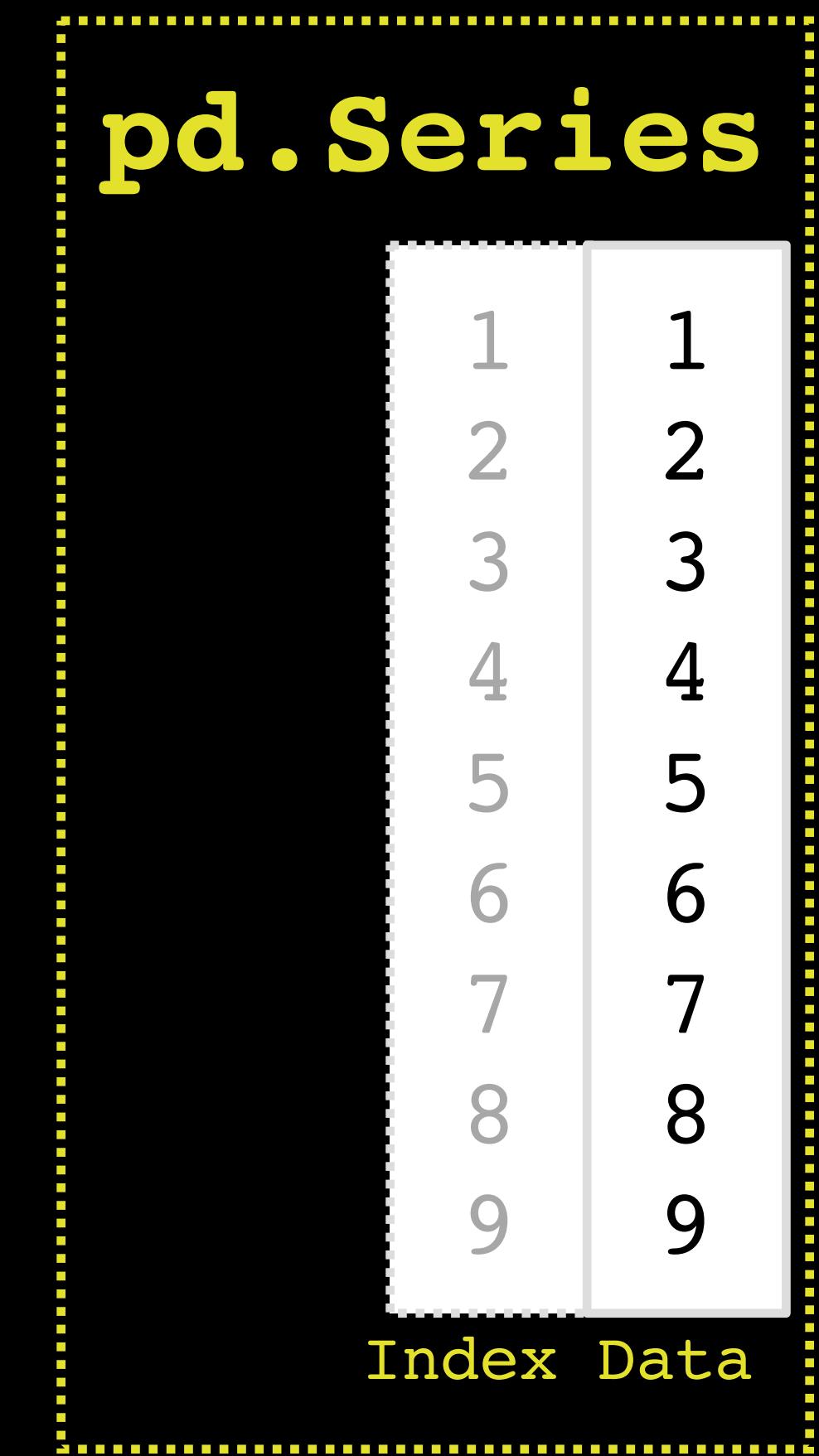
## Structure

|   |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |

Data



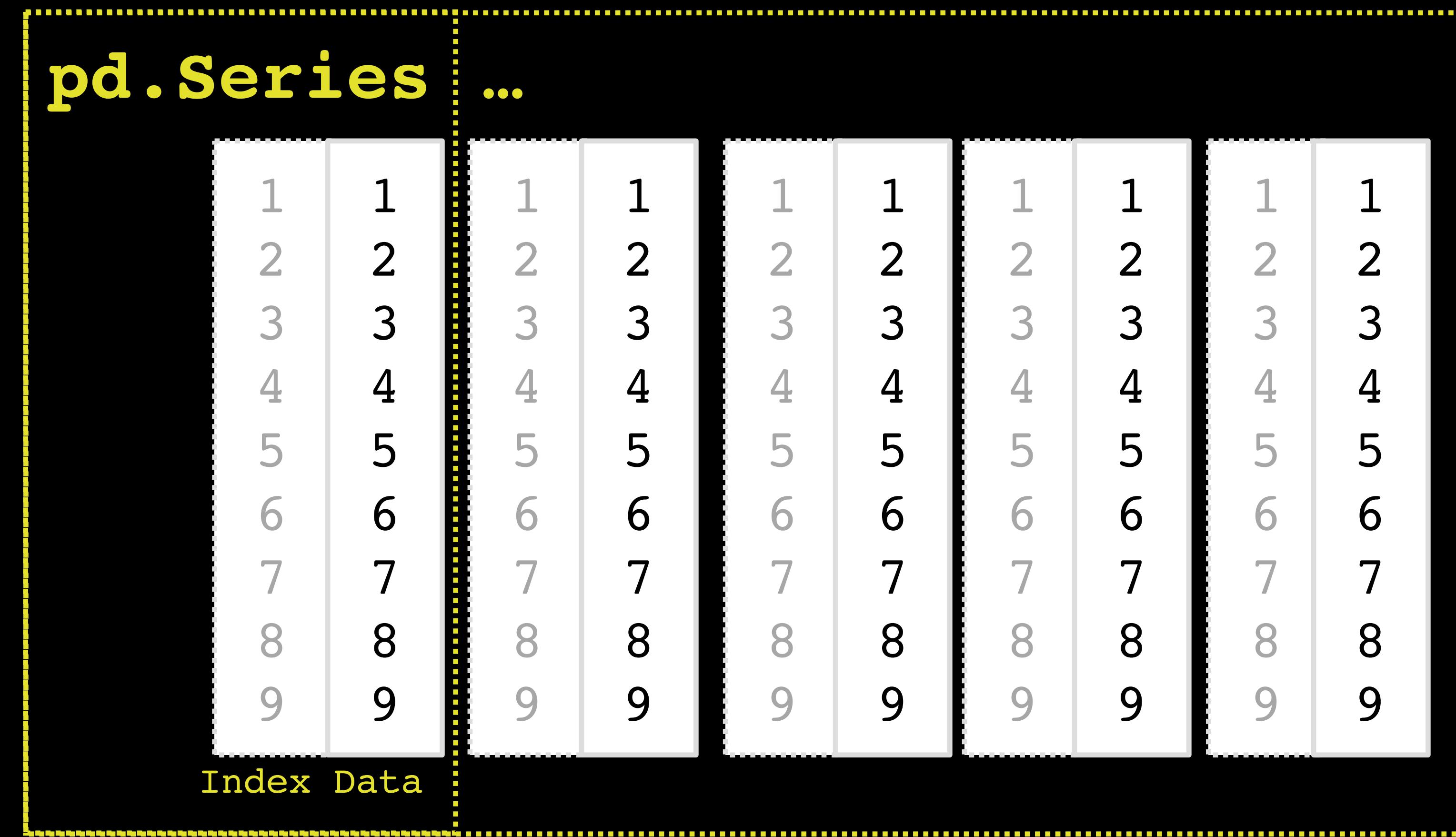
## Structure





# Structure

# pd.DataFrame





## Structure: DataSeries

- one dimensional, labeled series, may contain any data type
- the label of the series is usually called **index**
- **index** automatically created if not given
- **One** data type,  
datatype can be set or transformed dynamically in a pythonic fashion  
e. g. explicitly set

```
In [41]: series = pd.Series(np.random.rand(n))
series
```

```
Out[41]: 0    0.183418
1    0.836998
2    0.045299
3    0.894486
4    0.433939
dtype: float64
```

simple series, auto data type auto, index auto

```
In [42]: series = pd.Series(np.random.randint(1, 5, n))
series
```

```
Out[42]: 0    4
1    3
2    3
3    3
4    4
dtype: int64
```

simple series, auto data type auto, index auto

```
In [43]: series = pd.Series(np.random.randint(1, 5, n), dtype=np.float64)
series
```

```
Out[43]: 0    2.0
1    3.0
2    4.0
3    1.0
4    4.0
dtype: float64
```

simple series, auto data type set, index auto

```
In [48]: series = pd.Series(np.random.randint(1, 5, n), dtype=np.float64, index=[n*x for x in range(n)])
series
```

```
Out[48]: 0    3.0
5    3.0
10   1.0
15   2.0
20   4.0
dtype: float64
```

simple series, auto data type set, *numerical* index given

```
In [79]: series = pd.Series(np.random.randint(1, 100, n), dtype=np.float64, index=list('ABCDEFGHIJKLMNPQRSTUVWXYZ')[:n])
series
```

```
Out[79]: A    71.0
         B    60.0
         C    75.0
         D    18.0
         E    77.0
dtype: float64
```

simple series, auto **data type set**, *text-label* index given

```
In [80]: series['A']
```

```
Out[80]: 71.0
```

access via **index / label**

```
In [81]: series[0]
```

```
Out[81]: 71.0
```

access via **index / position**

```
In [82]: series[['A', 'D']]
```

```
Out[82]: A    71.0
         D    18.0
dtype: float64
```

access **multiple** via **index / label**

```
In [83]: series[1:3]
```

```
Out[83]: B    60.0
         C    75.0
dtype: float64
```

access **multiple** via **index / position range**

```
In [84]: series[[1, 3]]
```

```
Out[84]: B    60.0
         D    18.0
dtype: float64
```

access **multiple** via **index / multiple positions**

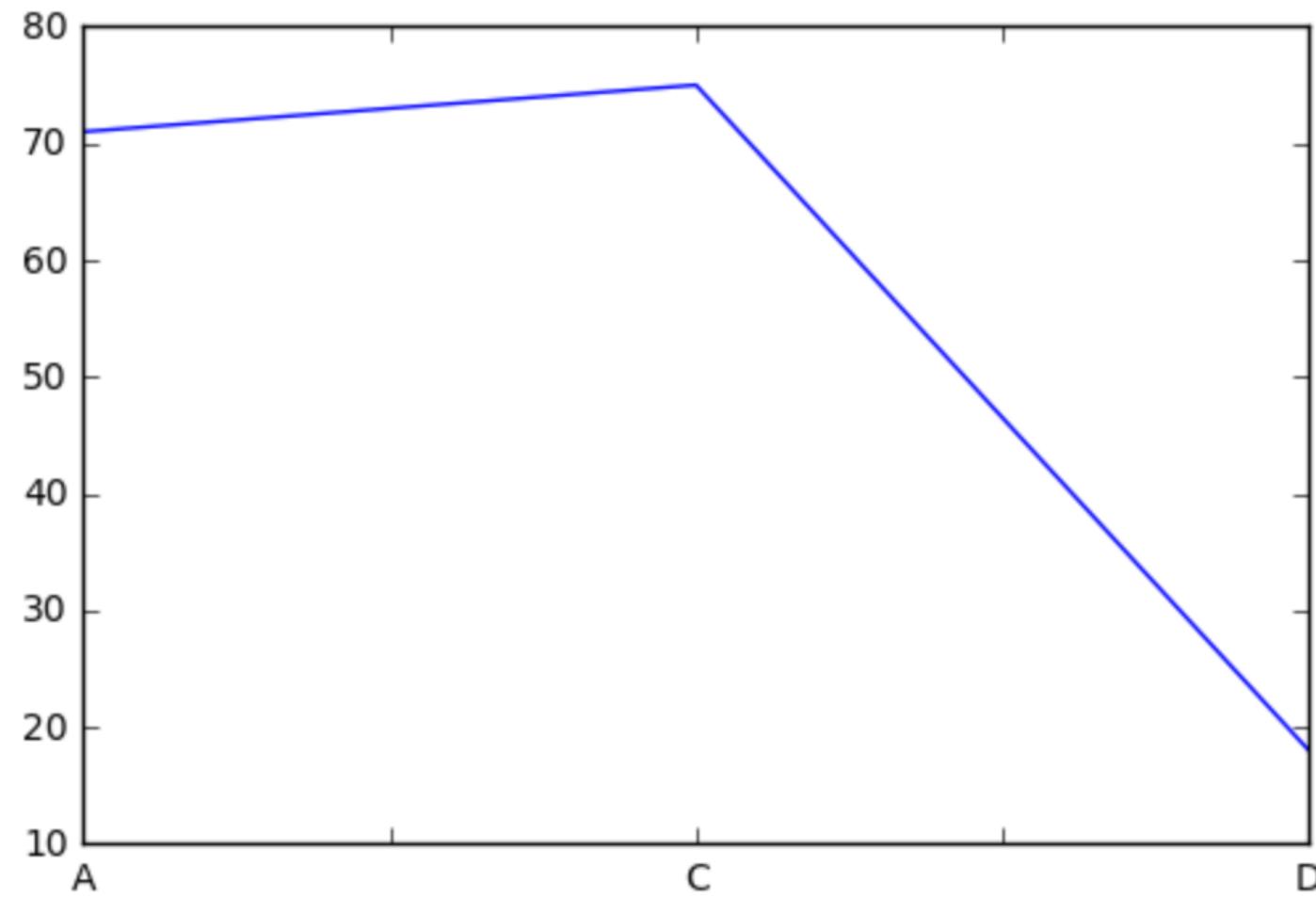
```
In [85]: series[lambda x: x%2 == 0]
```

```
Out[85]: B    60.0
         D    18.0
dtype: float64
```

access via **boolean index / lambda function**

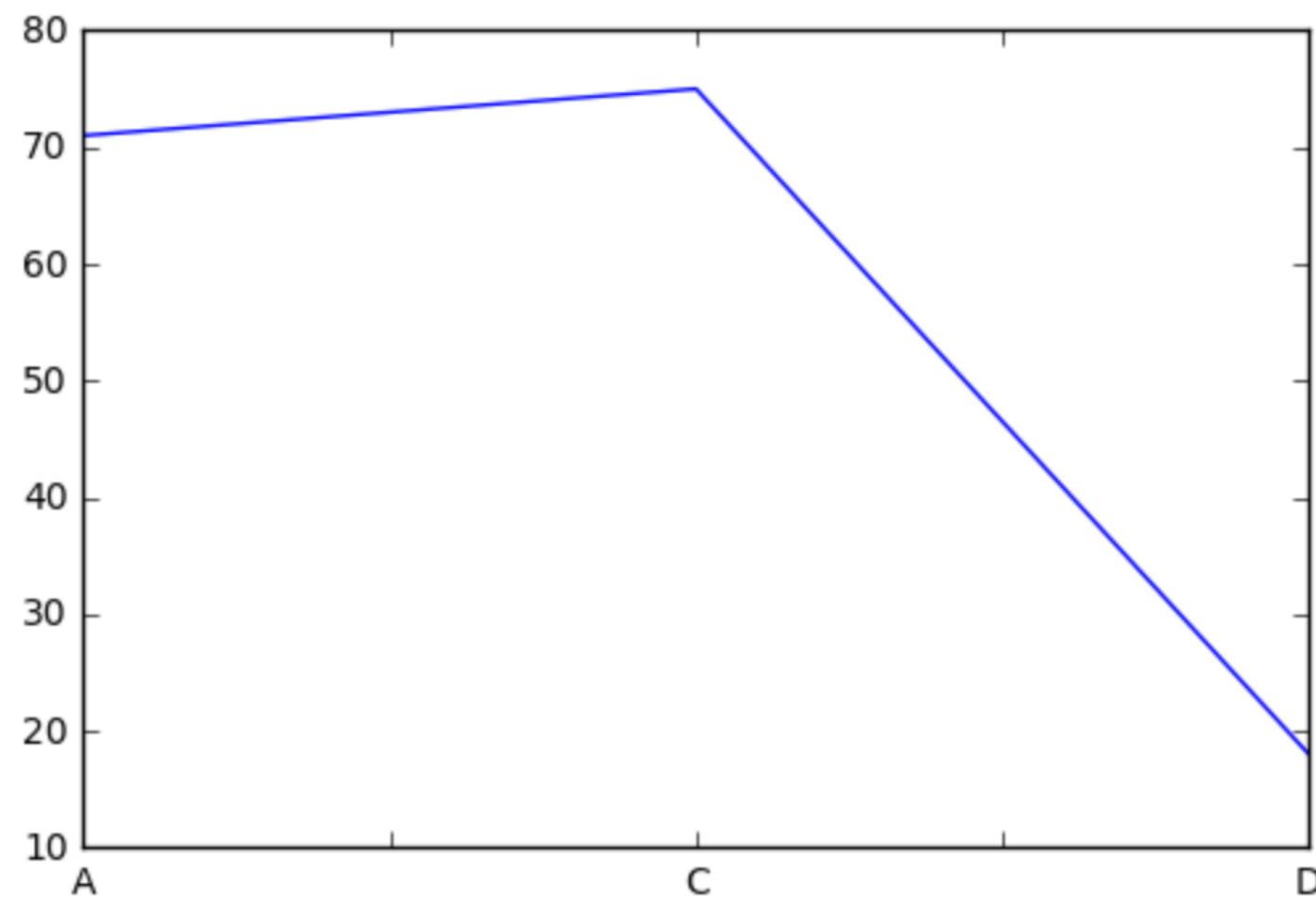
```
Out[79]: A    71.0  
          B    60.0  
          C    75.0  
          D    18.0  
          E    77.0  
         dtype: float64
```

```
In [90]: ax = series.loc[['A', 'C', 'D']].plot()
```



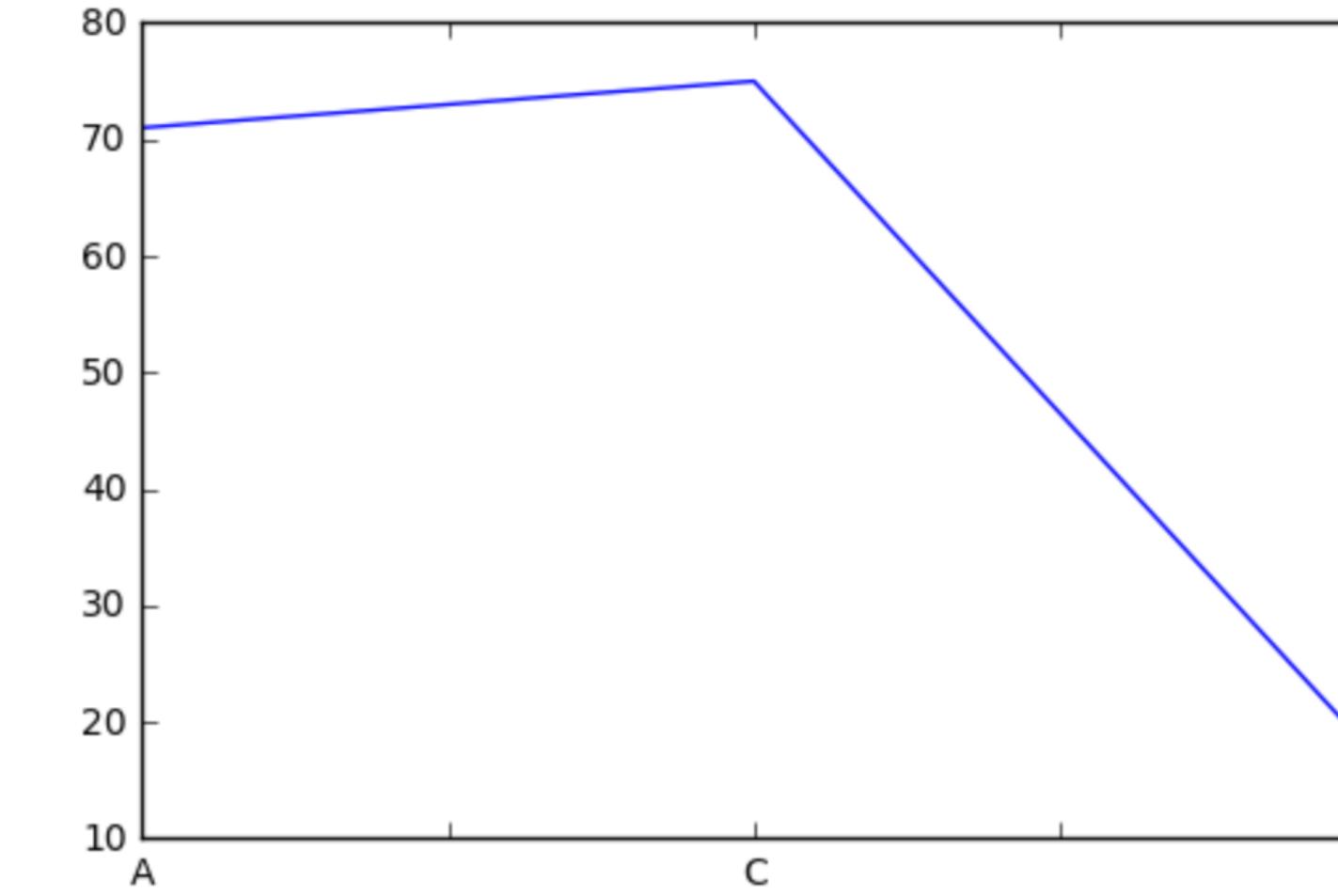
.loc()  
index label

```
In [91]: ax = series.iloc[[0, 2, 3]].plot()
```

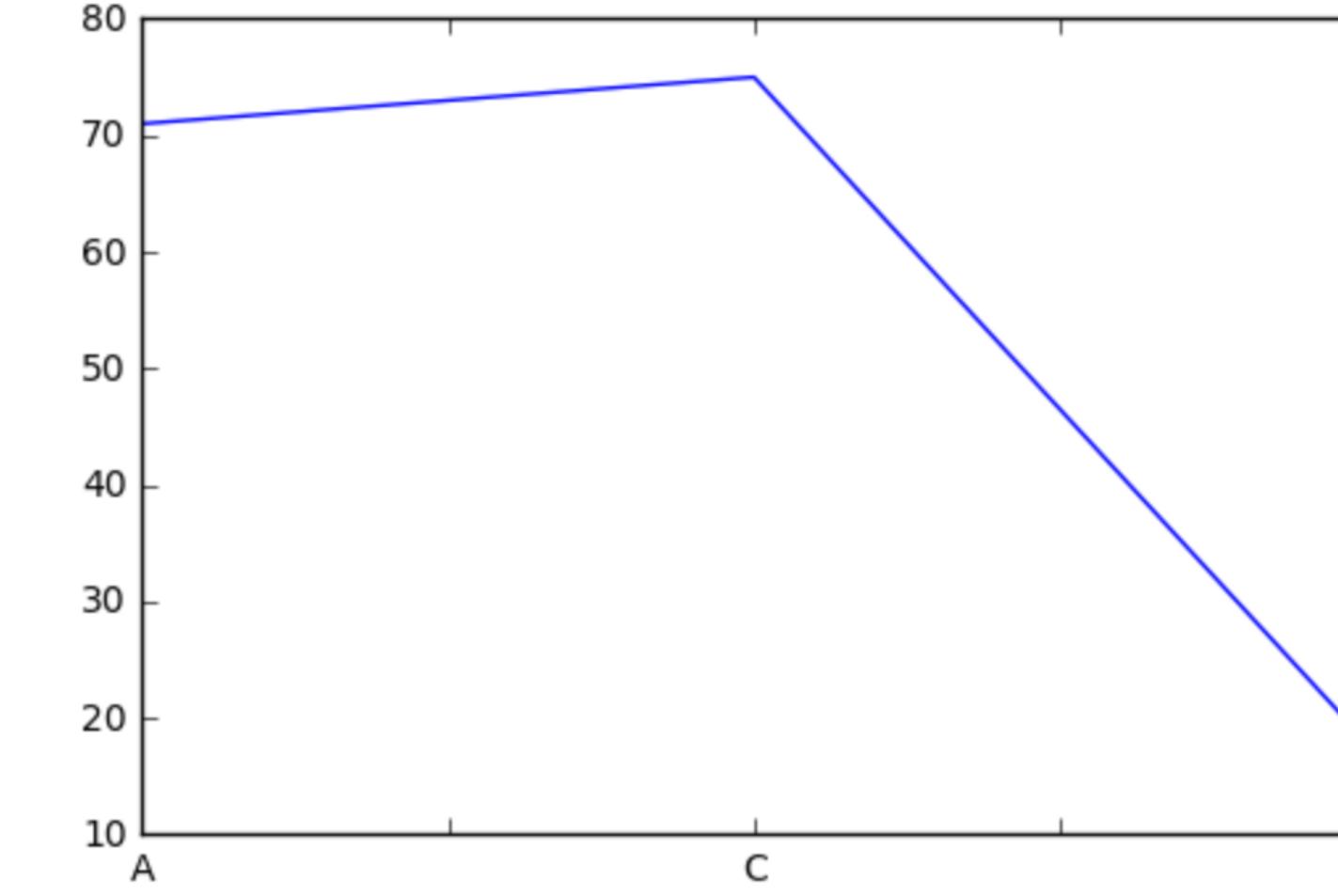


.iloc()  
index position

```
In [94]: ax = series.ix[['A', 'C', 'D']].plot()
```



```
In [95]: ax = series.ix[[0, 2, 3]].plot()
```



.ix()  
index *guessing*  
label/position fallback

```
In [100]: series.sample(2)
```

```
Out[100]: B    60.0  
          A    71.0  
          Name: some data, dtype: float64
```

.sample()  
sampling data set

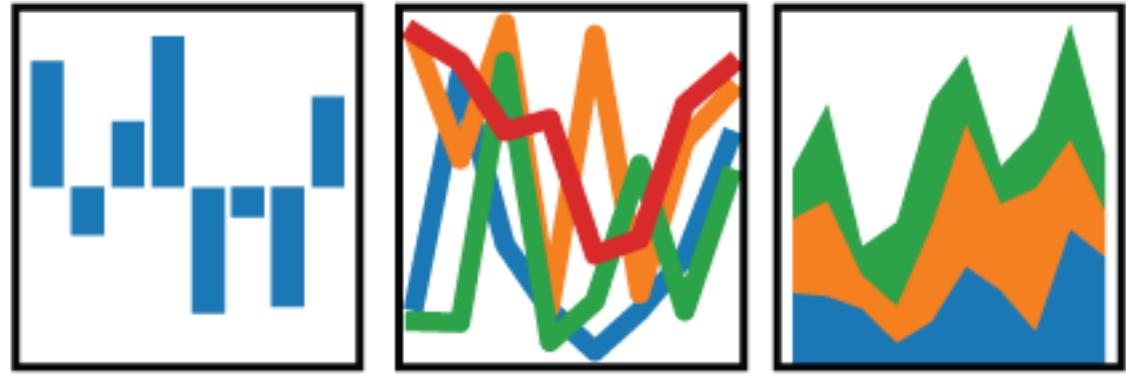
```
In [97]: series.name = "some data"  
series
```

```
Out[97]: A    71.0  
          B    60.0  
          C    75.0  
          D    18.0  
          E    77.0  
          Name: some data, dtype: float64
```

.name  
(column) names

# pandas

$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$



## Selecting Data

- Slicing
- Boolean indexing

`series[x], series[[x, y]]`

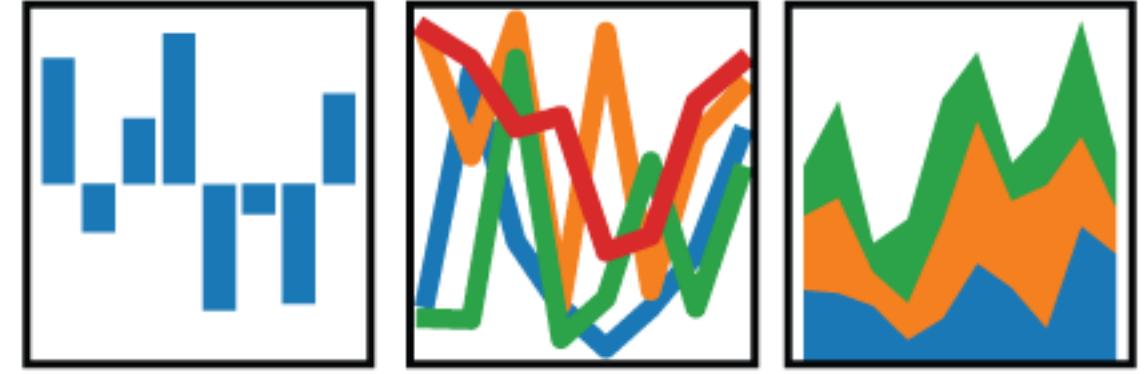
`series[2], series[[2, 3]], series[2:3]`

`series.ix() / .iloc() / .loc()`

`series.sample()`

pandas

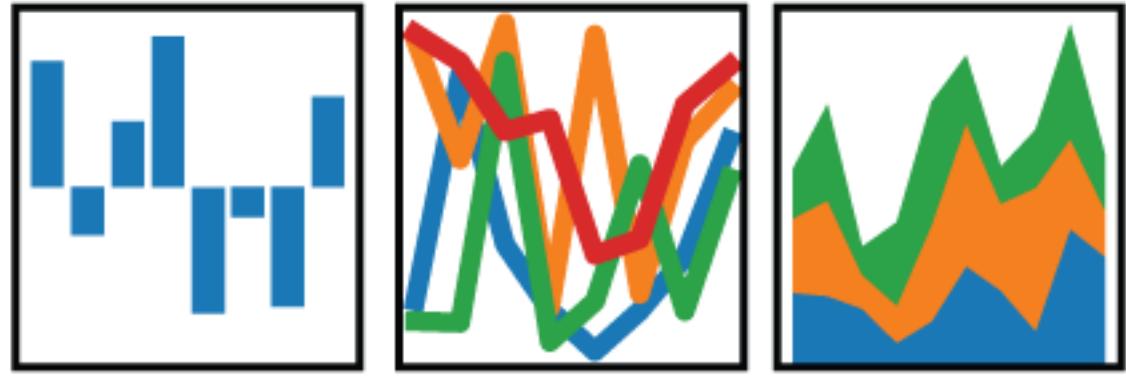
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



## Structure: DataFrame

- Two-dimensional, labeled data structure of e. g.
  - `DataSeries`
  - 2-D `numpy.ndarray`
  - other `DataFrames`
- `index` automatically created if not given

pandas  
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

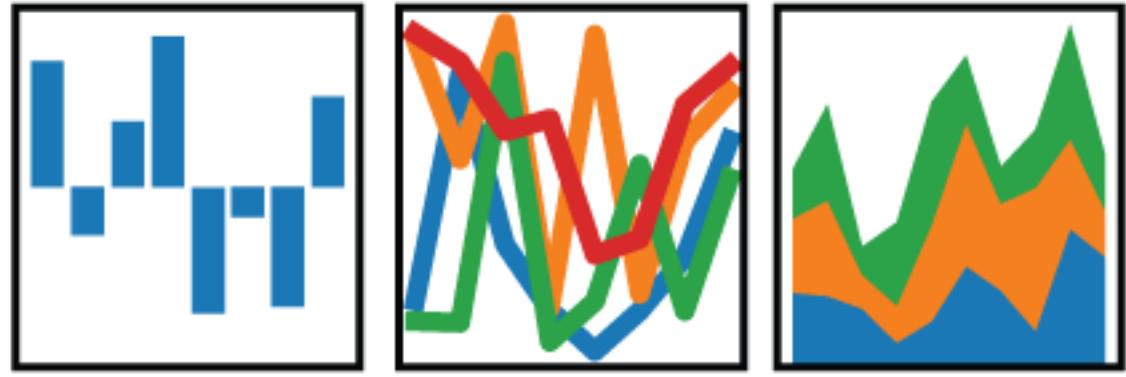


## Structure: Index

- Index
  - automatically created if not given
  - can be reset or replaced
  - types: position, timestamp, time range, labels,...
  - one or more dimensions
  - may contain a value more than once (NOT UNIQUE!)

# pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



## Examples

- work with series / calculation
- create and add a new series
- how to deal with null (NaN) values
- method calls directly from Series/ DataFrames

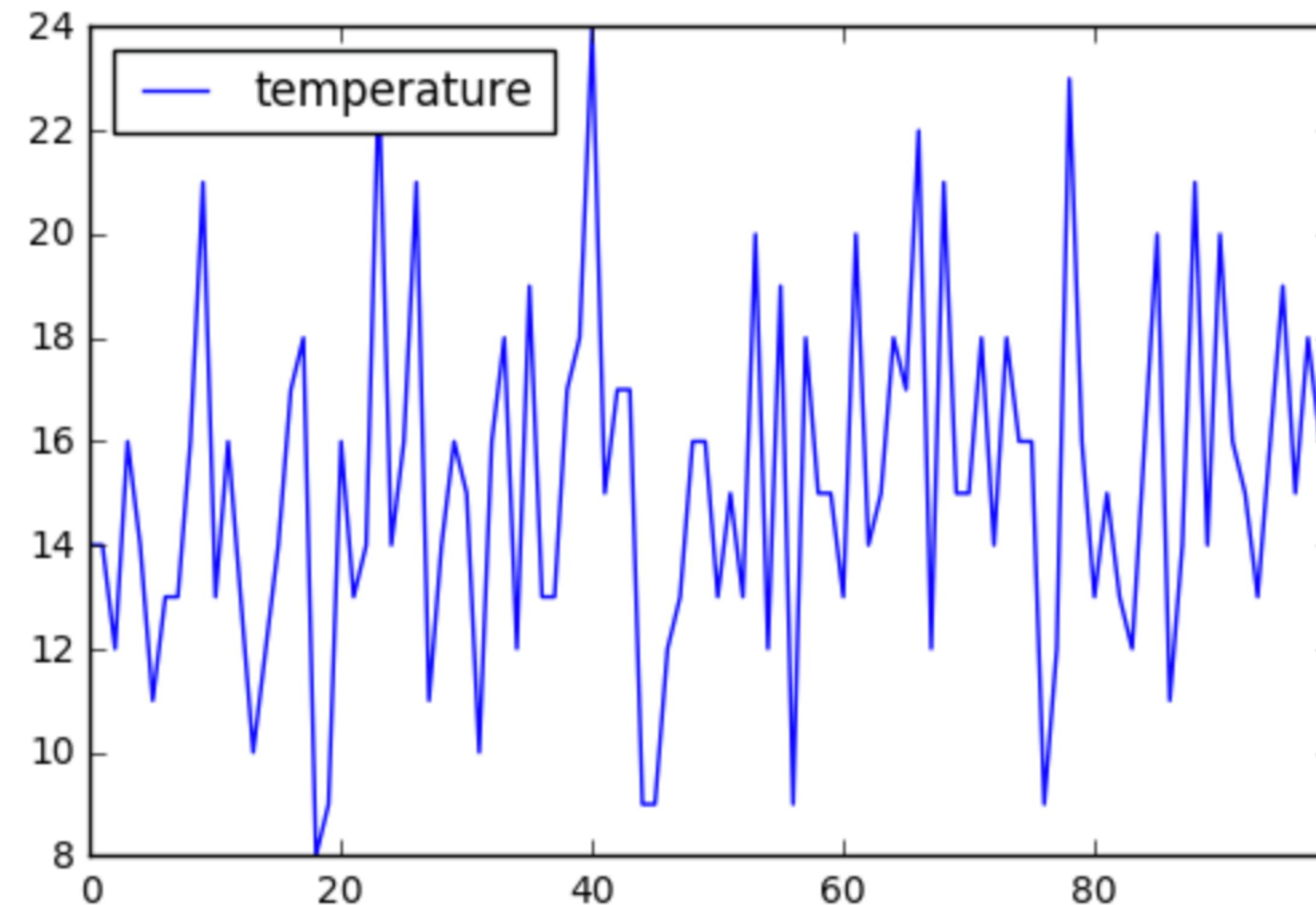
```
In [16]: df.columns = ['timestamp', 'temperature']
df.head(3)
```

```
Out[16]:
```

|   | timestamp           | temperature |
|---|---------------------|-------------|
| 0 | 2014-09-26T03:50:00 | 14.0        |
| 1 | 2014-08-10T05:00:00 | 14.0        |
| 2 | 2014-08-21T22:50:00 | 12.0        |

```
In [17]: df[:100].plot()
```

```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x10c39a0b8>
```



```
In [13]: def to_fahrenheit(celsius):
    return (celsius * 9./5.) + 32.
```

```
In [14]: df['temperature'].map(to_fahrenheit)[:5]
```

```
Out[14]: 0    57.2
1    57.2
2    53.6
3    60.8
4    57.2
Name: temperature, dtype: float64
```

```
In [15]: df['temperature F'] = df['temperature'].map(to_fahrenheit)
df.head(5)
```

Out[15]:

|   | timestamp           | temperature | temperature F |
|---|---------------------|-------------|---------------|
| 0 | 2014-09-26T03:50:00 | 14.0        | 57.2          |
| 1 | 2014-08-10T05:00:00 | 14.0        | 57.2          |
| 2 | 2014-08-21T22:50:00 | 12.0        | 53.6          |
| 3 | 2014-08-17T13:20:00 | 16.0        | 60.8          |
| 4 | 2014-08-06T01:20:00 | 14.0        | 57.2          |

```
In [16]: df['temperature F'] = df['temperature'].apply(lambda x: (x * 9./5.) + 32.)
df.head()
```

Out[16]:

|   | timestamp           | temperature | temperature F |
|---|---------------------|-------------|---------------|
| 0 | 2014-09-26T03:50:00 | 14.0        | 57.2          |
| 1 | 2014-08-10T05:00:00 | 14.0        | 57.2          |
| 2 | 2014-08-21T22:50:00 | 12.0        | 53.6          |
| 3 | 2014-08-17T13:20:00 | 16.0        | 60.8          |
| 4 | 2014-08-06T01:20:00 | 14.0        | 57.2          |

```
In [17]: df['ruleoftumb'] = df['temperature F'] / df['temperature']
df.head()
```

Out[17]:

|   | timestamp           | temperature | temperature F | ruleoftumb |
|---|---------------------|-------------|---------------|------------|
| 0 | 2014-09-26T03:50:00 | 14.0        | 57.2          | 4.085714   |
| 1 | 2014-08-10T05:00:00 | 14.0        | 57.2          | 4.085714   |
| 2 | 2014-08-21T22:50:00 | 12.0        | 53.6          | 4.466667   |
| 3 | 2014-08-17T13:20:00 | 16.0        | 60.8          | 3.800000   |
| 4 | 2014-08-06T01:20:00 | 14.0        | 57.2          | 4.085714   |

```
In [18]: df['ruleoftumb'].describe()
```

Out[18]: count 4356.000000  
mean 3.988146  
std 0.709771  
min 2.985185  
25% 3.577778  
50% 3.933333  
75% 4.261538  
max 17.800000  
Name: ruleoftumb, dtype: float64

```
In [19]: df.rename(columns={'ruleoftumb': 'bad_rule'}, inplace=True)  
df.head()
```

Out[19]:

|   | timestamp           | temperature | temperature F | bad_rule |
|---|---------------------|-------------|---------------|----------|
| 0 | 2014-09-26T03:50:00 | 14.0        | 57.2          | 4.085714 |
| 1 | 2014-08-10T05:00:00 | 14.0        | 57.2          | 4.085714 |
| 2 | 2014-08-21T22:50:00 | 12.0        | 53.6          | 4.466667 |
| 3 | 2014-08-17T13:20:00 | 16.0        | 60.8          | 3.800000 |
| 4 | 2014-08-06T01:20:00 | 14.0        | 57.2          | 4.085714 |

```
In [20]: df.drop('bad_rule', axis=1, inplace=True)  
df.head()
```

Out[20]:

|   | timestamp           | temperature | temperature F |
|---|---------------------|-------------|---------------|
| 0 | 2014-09-26T03:50:00 | 14.0        | 57.2          |
| 1 | 2014-08-10T05:00:00 | 14.0        | 57.2          |
| 2 | 2014-08-21T22:50:00 | 12.0        | 53.6          |
| 3 | 2014-08-17T13:20:00 | 16.0        | 60.8          |
| 4 | 2014-08-06T01:20:00 | 14.0        | 57.2          |



## Modifying Series/DataFrames

- Methods applied to Series or DataFrames **do not change** them, but **return** the result as Series or DataFrames
- With parameter **inplace** the result can be deployed directly into Series / DataFrames

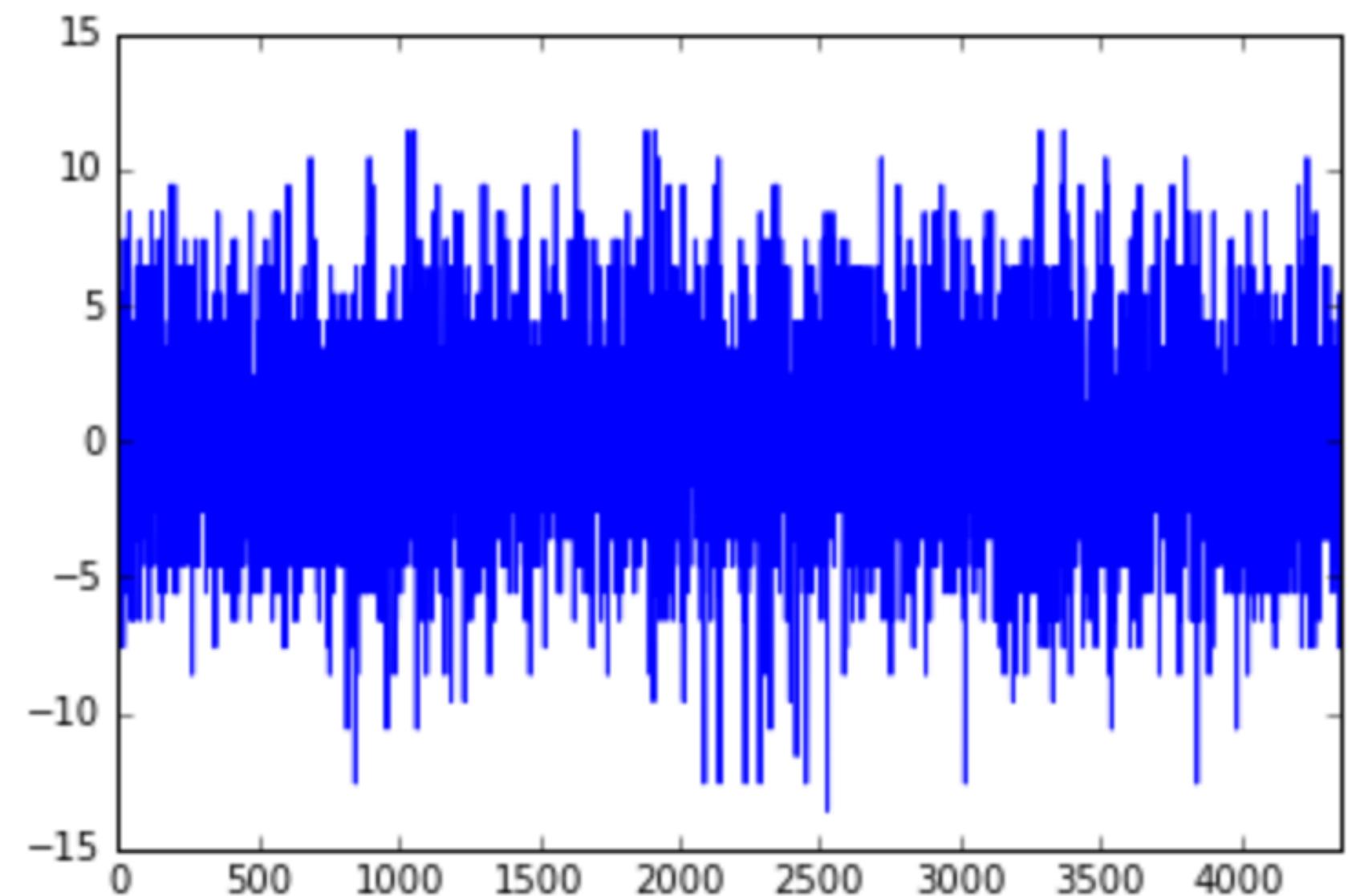
```
In [22]: df['deviation'] = df['temperature'] - df['temperature'].mean()  
df.head()
```

Out[22]:

|   | timestamp           | temperature | temperature F | deviation |
|---|---------------------|-------------|---------------|-----------|
| 0 | 2014-09-26T03:50:00 | 14.0        | 57.2          | -1.59045  |
| 1 | 2014-08-10T05:00:00 | 14.0        | 57.2          | -1.59045  |
| 2 | 2014-08-21T22:50:00 | 12.0        | 53.6          | -3.59045  |
| 3 | 2014-08-17T13:20:00 | 16.0        | 60.8          | 0.40955   |
| 4 | 2014-08-06T01:20:00 | 14.0        | 57.2          | -1.59045  |

```
In [23]: df['deviation'].plot()
```

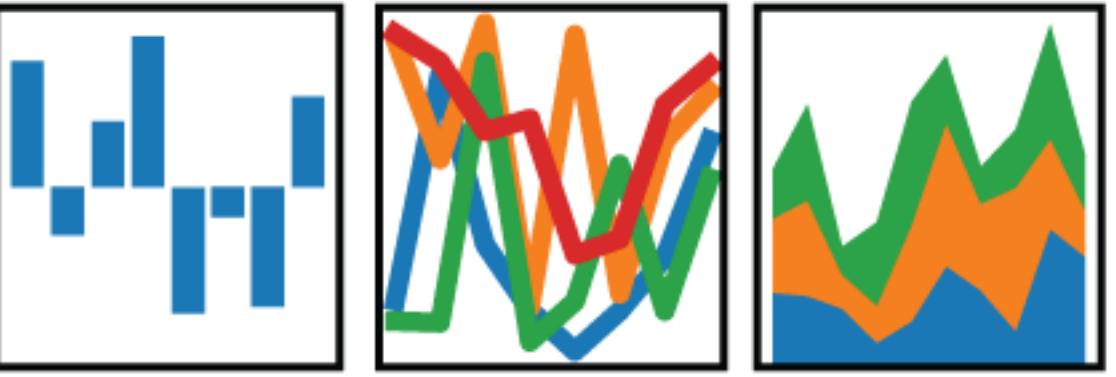
Out[23]: <matplotlib.axes.\_subplots.AxesSubplot at 0x114a07c88>



```
In [21]: df.groupby('temperature').count()
```

Out[21]:

|             | timestamp | temperature F |
|-------------|-----------|---------------|
| temperature |           |               |
| <b>2.0</b>  | 1         | 1             |
| <b>3.0</b>  | 8         | 8             |
| <b>4.0</b>  | 1         | 1             |
| <b>5.0</b>  | 6         | 6             |
| <b>6.0</b>  | 7         | 7             |
| <b>7.0</b>  | 25        | 25            |
| <b>8.0</b>  | 29        | 29            |
| <b>9.0</b>  | 69        | 69            |
| <b>10.0</b> | 143       | 143           |
| <b>11.0</b> | 233       | 233           |
| <b>12.0</b> | 242       | 242           |
| <b>13.0</b> | 455       | 455           |
| <b>14.0</b> | 552       | 552           |
| <b>15.0</b> | 464       | 464           |
| <b>16.0</b> | 504       | 504           |
| <b>17.0</b> | 368       | 368           |
| <b>18.0</b> | 371       | 371           |
| <b>19.0</b> | 0         | 0             |



## NaN Values & Replacing

- NaN is representation of **null** values
- `series.describe()` ignore NaN
- NaNs:
  - remove **drop()**
  - replace with default
    - forward- or backwards-fill, interpolate
- Series can be removed from DF with **drop()**



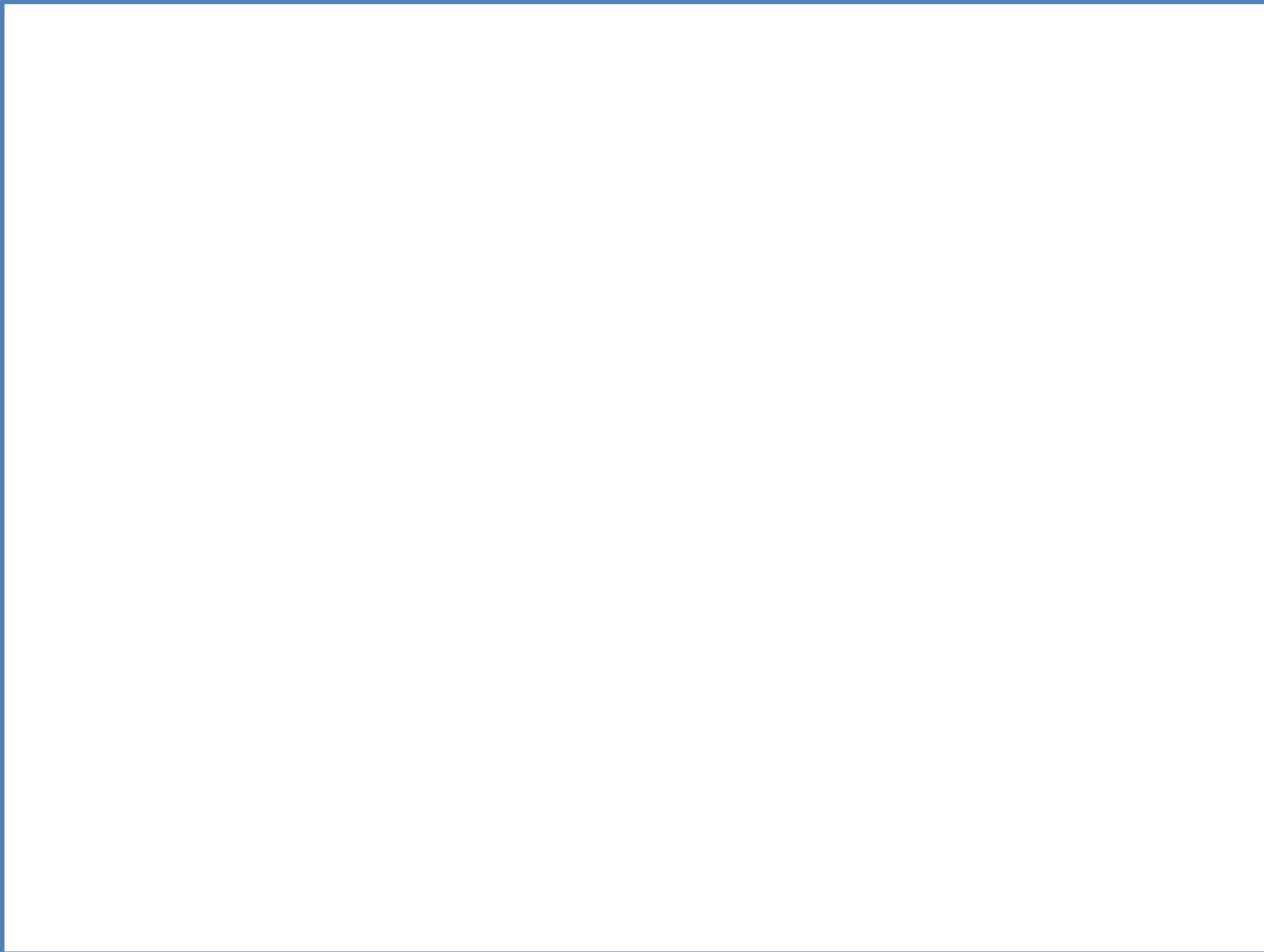
## Data Aggregation

- `describe()`
- `groupby()`
- `groupby([]) & unstack()`
- `mean(), sum(), median(),...`



## End Part 1

- **DataSeries & DataFrame**
- **I/O**
- **Data analysis & aggregation**
- **Indexes**
- **Visualisation**
- **Interacting with the data**



Year

K Ö N I G S W E G

|    |    |    |    |
|----|----|----|----|
|    |    |    |    |
| 31 | 28 | 31 | 30 |
| 31 | 30 | 31 | 31 |
| 30 | 31 | 30 | 31 |

Year

12 months

|    |                    |       |    |
|----|--------------------|-------|----|
|    | February<br>90% of | March |    |
| 31 | 28                 | 31    | 30 |
| 31 | 30                 | 31    | 31 |
| 30 | 31                 | 30    | 31 |

Year  
12 months

Roman year used to start in March and had 10 months



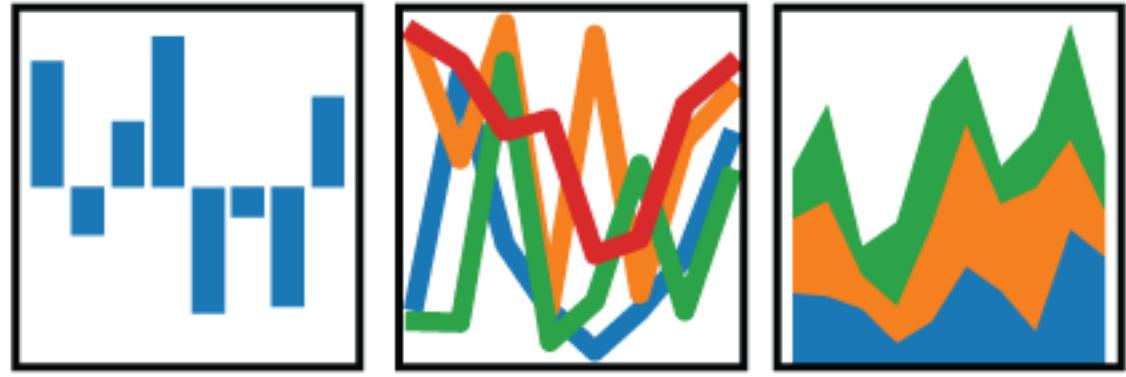
2 months there was "no" month



solar | topical year

# pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



## TimeSeries

- TimeSeriesIndex
- **pd.to\_datetime()** ! US date friendly
- Data Aggregation examples

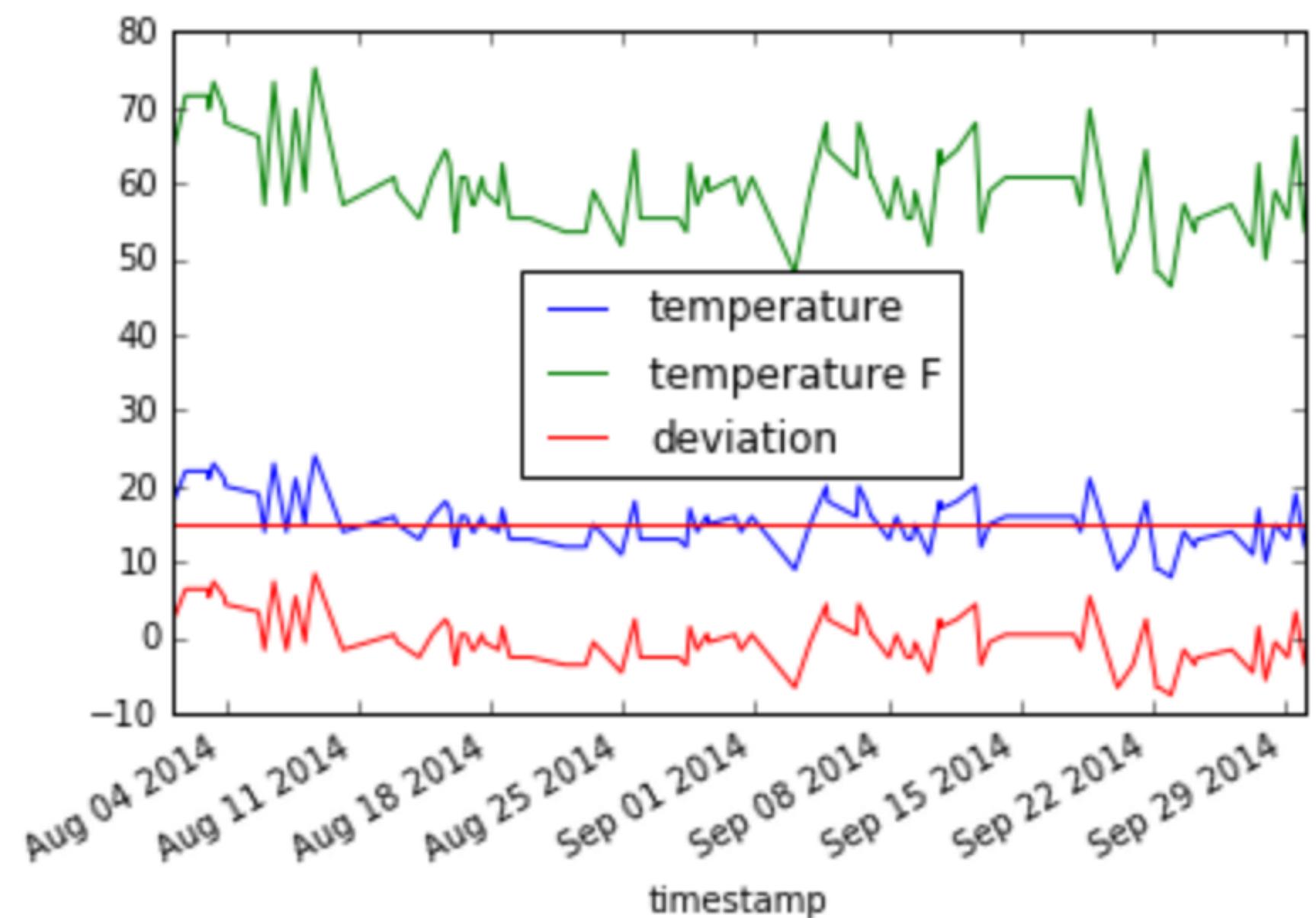
```
In [24]: # create TimeSeries Index
```

```
df.index = pd.to_datetime(df['timestamp'])
```

```
In [25]: ax = df[:100].plot()
```

```
ax.axhline(df[:100]['temperature'].median(), color='r', linestyle='--')
```

```
Out[25]: <matplotlib.lines.Line2D at 0x114dd6978>
```



```
In [26]: df.head()
```

Out[26]:

|                     | timestamp           | temperature | temperature F | deviation |
|---------------------|---------------------|-------------|---------------|-----------|
| timestamp           |                     |             |               |           |
| 2014-09-26 03:50:00 | 2014-09-26T03:50:00 | 14.0        | 57.2          | -1.59045  |
| 2014-08-10 05:00:00 | 2014-08-10T05:00:00 | 14.0        | 57.2          | -1.59045  |
| 2014-08-21 22:50:00 | 2014-08-21T22:50:00 | 12.0        | 53.6          | -3.59045  |
| 2014-08-17 13:20:00 | 2014-08-17T13:20:00 | 16.0        | 60.8          | 0.40955   |
| 2014-08-06 01:20:00 | 2014-08-06T01:20:00 | 14.0        | 57.2          | -1.59045  |

```
In [27]: df.index.view
```

Out[27]: <bound method Index.view of DatetimeIndex(['2014-09-26 03:50:00', '2014-08-10 05:00:00',  
'2014-08-21 22:50:00', '2014-08-17 13:20:00',  
'2014-08-06 01:20:00', '2014-09-27 06:50:00',  
'2014-08-25 21:50:00', '2014-08-14 05:20:00',  
'2014-09-14 05:20:00', '2014-08-03 02:50:00',  
...  
'2014-08-22 10:00:00', '2014-09-10 16:20:00',  
'2014-08-14 15:50:00', '2014-09-05 04:00:00',  
'2014-09-29 02:50:00', '2014-08-21 01:50:00',  
'2014-09-13 10:00:00', '2014-08-16 23:20:00',  
'2014-09-28 10:20:00', '2014-09-25 18:20:00'],  
dtype='datetime64[ns]', name='timestamp', length=4357, freq=None)>

```
In [61]: df.groupby(df.index.date).count()
```

Out[61]:

|            | timestamp | temperature | temperature F | deviation | weekday | weekend |
|------------|-----------|-------------|---------------|-----------|---------|---------|
| 2014-08-01 | 66        | 66          | 66            | 66        | 66      | 66      |
| 2014-08-02 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-03 | 70        | 70          | 70            | 70        | 70      | 70      |
| 2014-08-04 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-05 | 68        | 68          | 68            | 68        | 68      | 68      |
| 2014-08-06 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-07 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-08 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-09 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-10 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-11 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-12 | 68        | 68          | 68            | 68        | 68      | 68      |
| 2014-08-13 | 71        | 71          | 71            | 71        | 71      | 71      |
| 2014-08-14 | 71        | 71          | 71            | 71        | 71      | 71      |
| 2014-08-15 | 71        | 71          | 71            | 71        | 71      | 71      |
| 2014-08-16 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-17 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-18 | 72        | 72          | 72            | 72        | 72      | 72      |
| 2014-08-19 | 72        | 72          | 72            | 72        | 72      | 72      |

```
In [31]: df['timestamp'].describe()  
# get info, text = object
```

```
Out[31]: count          4357  
unique         4357  
top    2014-08-04T21:50:00  
freq             1  
Name: timestamp, dtype: object
```

```
In [32]: df['temperature'].describe()  
# get basic stats
```

```
Out[32]: count    4354.000000  
mean        15.590951  
std         3.596220  
min         2.000000  
25%        13.000000  
50%        15.000000  
75%        18.000000  
max         27.000000  
Name: temperature, dtype: float64
```

```
In [93]: df.describe(percentiles=[.1, .5, .6, .7])  
# get basic stats
```

Out[93]:

|              | temperature | temperature F | deviation     | weekday     |
|--------------|-------------|---------------|---------------|-------------|
| <b>count</b> | 4354.000000 | 4354.000000   | 4.354000e+03  | 4354.000000 |
| <b>mean</b>  | 15.590951   | 60.063712     | 5.548565e-16  | 3.024575    |
| <b>std</b>   | 3.596220    | 6.473197      | 3.596220e+00  | 2.029326    |
| <b>min</b>   | 2.000000    | 35.600000     | -1.359095e+01 | 0.000000    |
| <b>10%</b>   | 11.000000   | 51.800000     | -4.590951e+00 | 0.000000    |
| <b>50%</b>   | 15.000000   | 59.000000     | -5.909508e-01 | 3.000000    |
| <b>60%</b>   | 16.000000   | 60.800000     | 4.090492e-01  | 4.000000    |
| <b>70%</b>   | 17.000000   | 62.600000     | 1.409049e+00  | 4.000000    |
| <b>max</b>   | 27.000000   | 80.600000     | 1.140905e+01  | 6.000000    |

```
In [92]: df.describe(percentiles=[.1, .5, .6, .7], include=[np.float64])  
# get basic stats
```

Out[92]:

|              | temperature | temperature F | deviation     |
|--------------|-------------|---------------|---------------|
| <b>count</b> | 4354.000000 | 4354.000000   | 4.354000e+03  |
| <b>mean</b>  | 15.590951   | 60.063712     | 5.548565e-16  |
| <b>std</b>   | 3.596220    | 6.473197      | 3.596220e+00  |
| <b>min</b>   | 2.000000    | 35.600000     | -1.359095e+01 |
| <b>10%</b>   | 11.000000   | 51.800000     | -4.590951e+00 |
| <b>50%</b>   | 15.000000   | 59.000000     | -5.909508e-01 |
| <b>60%</b>   | 16.000000   | 60.800000     | 4.090492e-01  |
| <b>70%</b>   | 17.000000   | 62.600000     | 1.409049e+00  |
| <b>max</b>   | 27.000000   | 80.600000     | 1.140905e+01  |

```
In [45]: df.resample('D').max().head()
```

Out[45]:

|            | timestamp           | temperature | temperature F | deviation | weekday | weekend |
|------------|---------------------|-------------|---------------|-----------|---------|---------|
| timestamp  |                     |             |               |           |         |         |
| 2014-08-01 | 2014-08-01T23:50:00 | 25.0        | 77.0          | 9.409049  | 4       | False   |
| 2014-08-02 | 2014-08-02T23:50:00 | 27.0        | 80.6          | 11.409049 | 5       | True    |
| 2014-08-03 | 2014-08-03T23:50:00 | 25.0        | 77.0          | 9.409049  | 6       | True    |
| 2014-08-04 | 2014-08-04T23:50:00 | 24.0        | 75.2          | 8.409049  | 0       | False   |
| 2014-08-05 | 2014-08-05T23:50:00 | 23.0        | 73.4          | 7.409049  | 1       | False   |

```
In [46]: df.resample('M').mean().head()
```

Out[46]:

|            | temperature | temperature F | deviation | weekday  | weekend  |
|------------|-------------|---------------|-----------|----------|----------|
| timestamp  |             |               |           |          |          |
| 2014-08-31 | 16.436652   | 61.585973     | 0.845701  | 3.201810 | 0.324434 |
| 2014-09-30 | 14.719216   | 58.494590     | -0.871734 | 2.841884 | 0.267724 |

```
In [35]: df[df['temperature'].isnull()]
```

Out[35]:

|                     | timestamp           | temperature | temperature F | deviation |
|---------------------|---------------------|-------------|---------------|-----------|
| timestamp           |                     |             |               |           |
| 2014-09-09 03:50:00 | 2014-09-09T03:50:00 | NaN         | NaN           | NaN       |
| 2014-09-15 10:00:00 | 2014-09-15T10:00:00 | NaN         | NaN           | NaN       |
| 2014-08-27 05:00:00 | 2014-08-27T05:00:00 | NaN         | NaN           | NaN       |

```
In [37]: df['temperature'].isnull()[2350:2357]
```

Out[37]: timestamp

```
2014-08-24 10:20:00    False
2014-09-14 02:20:00    False
2014-09-29 05:00:00    False
2014-08-10 12:00:00    False
2014-08-27 05:00:00    True
2014-09-28 12:50:00    False
2014-08-05 01:00:00    False
Name: temperature, dtype: bool
```

```
In [38]: df['temperature'].isnull().any()
```

Out[38]: True

```
In [38]: df.dropna()  
df['temperature'].isnull().any()
```

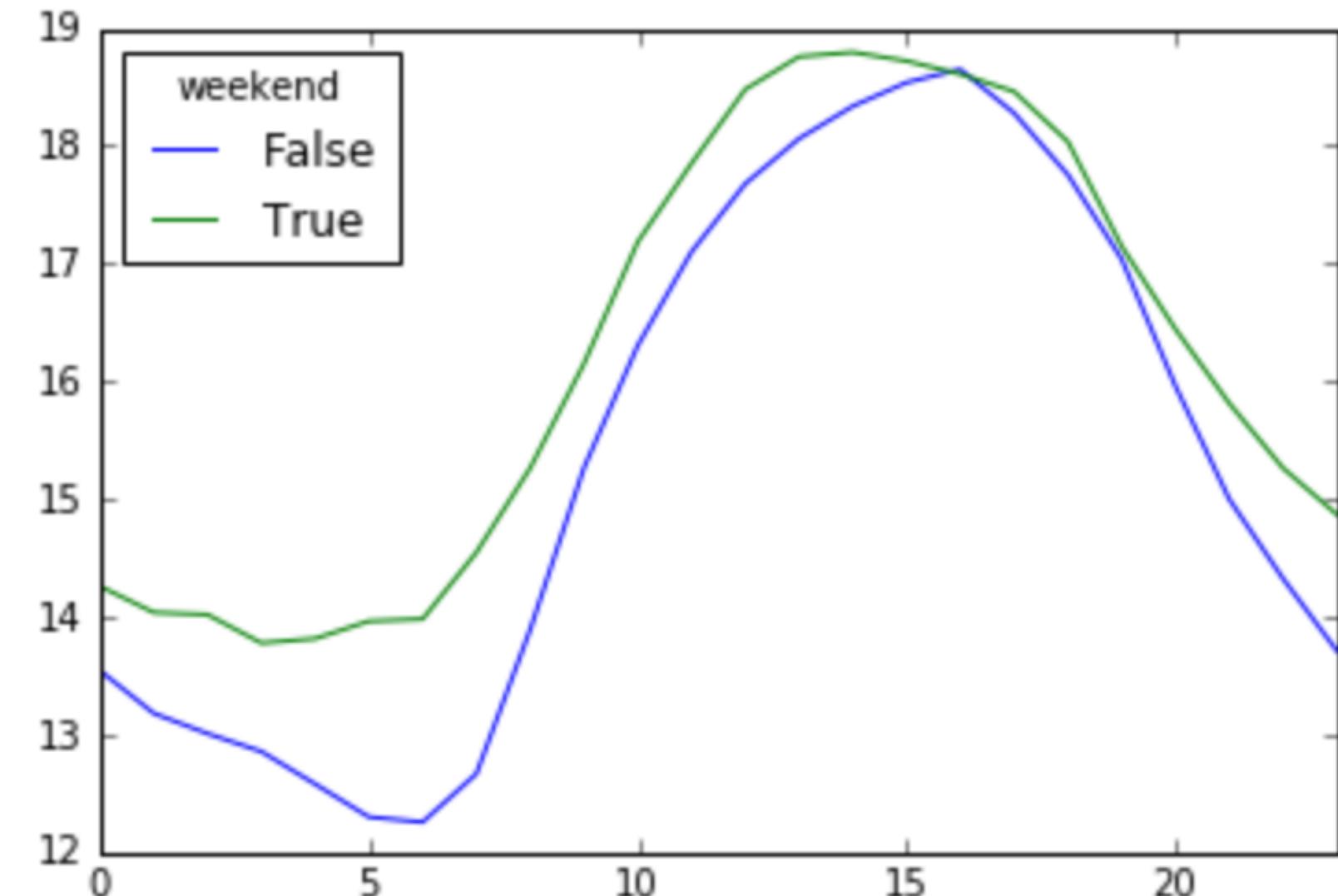
Out[38]: True

```
In [39]: df.dropna(inplace=True)  
df['temperature'].isnull().any()
```

Out[39]: False

```
In [40]: df['weekday'] = df.index.weekday  
df['weekend'] = df['weekday'].isin({5, 6})  
df.groupby(['weekend', df.index.hour])['temperature'].mean().unstack(level=0).plot()
```

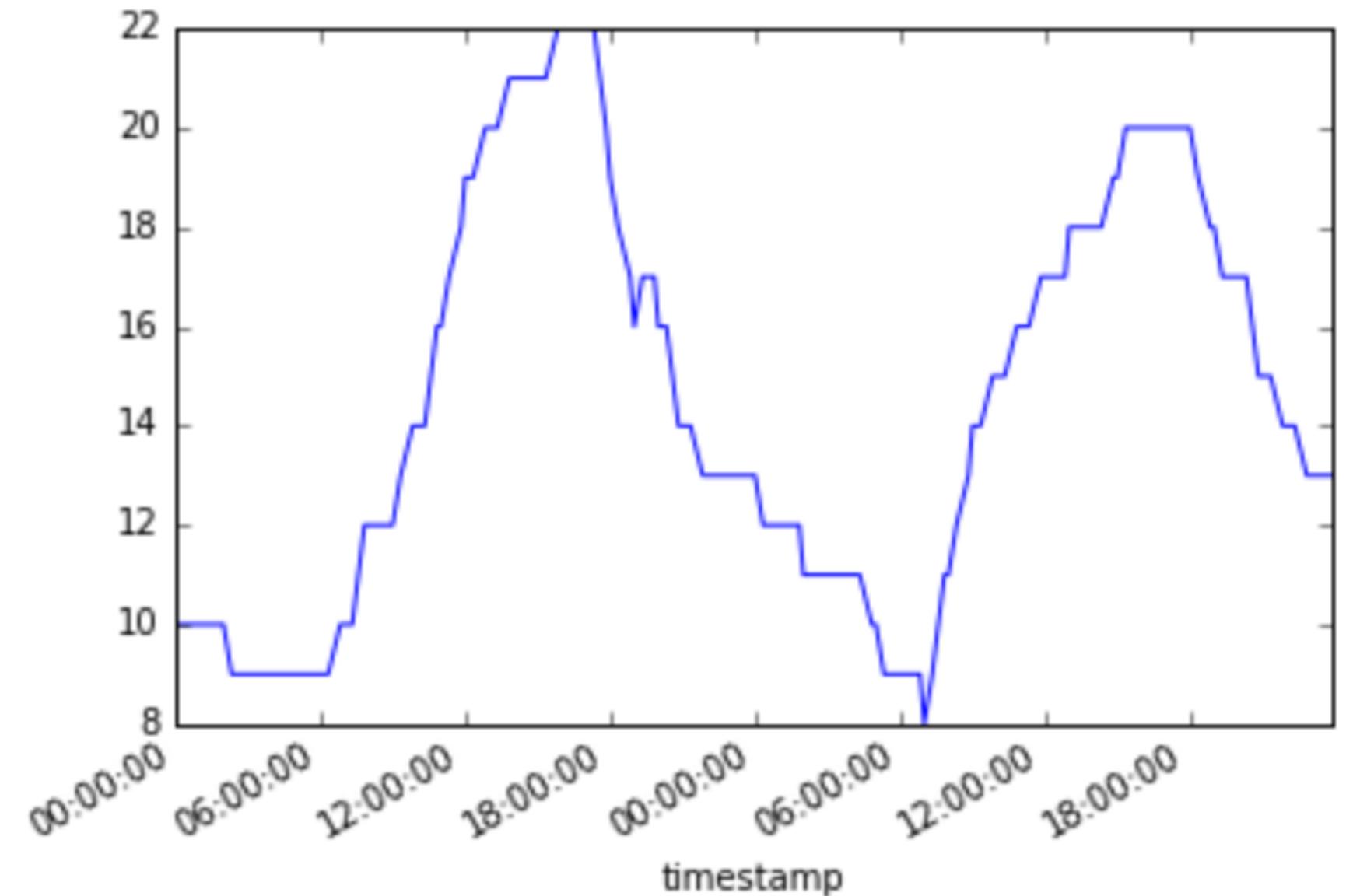
```
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x114b72fd0>
```



```
In [41]: # selecting ranges
```

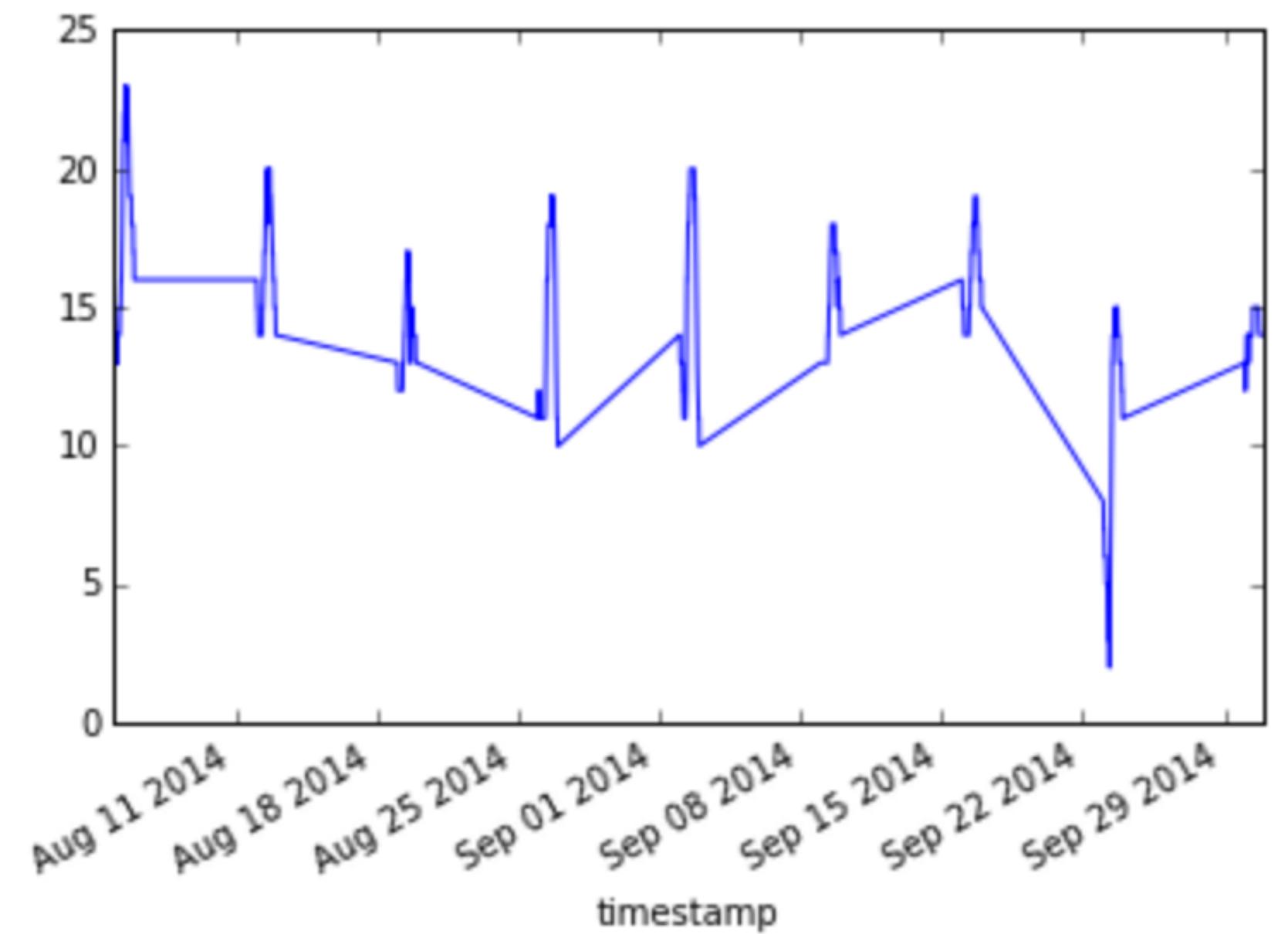
```
df["2014-08-27":"2014-08-28"]['temperature'].plot()
```

```
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x114a55780>
```



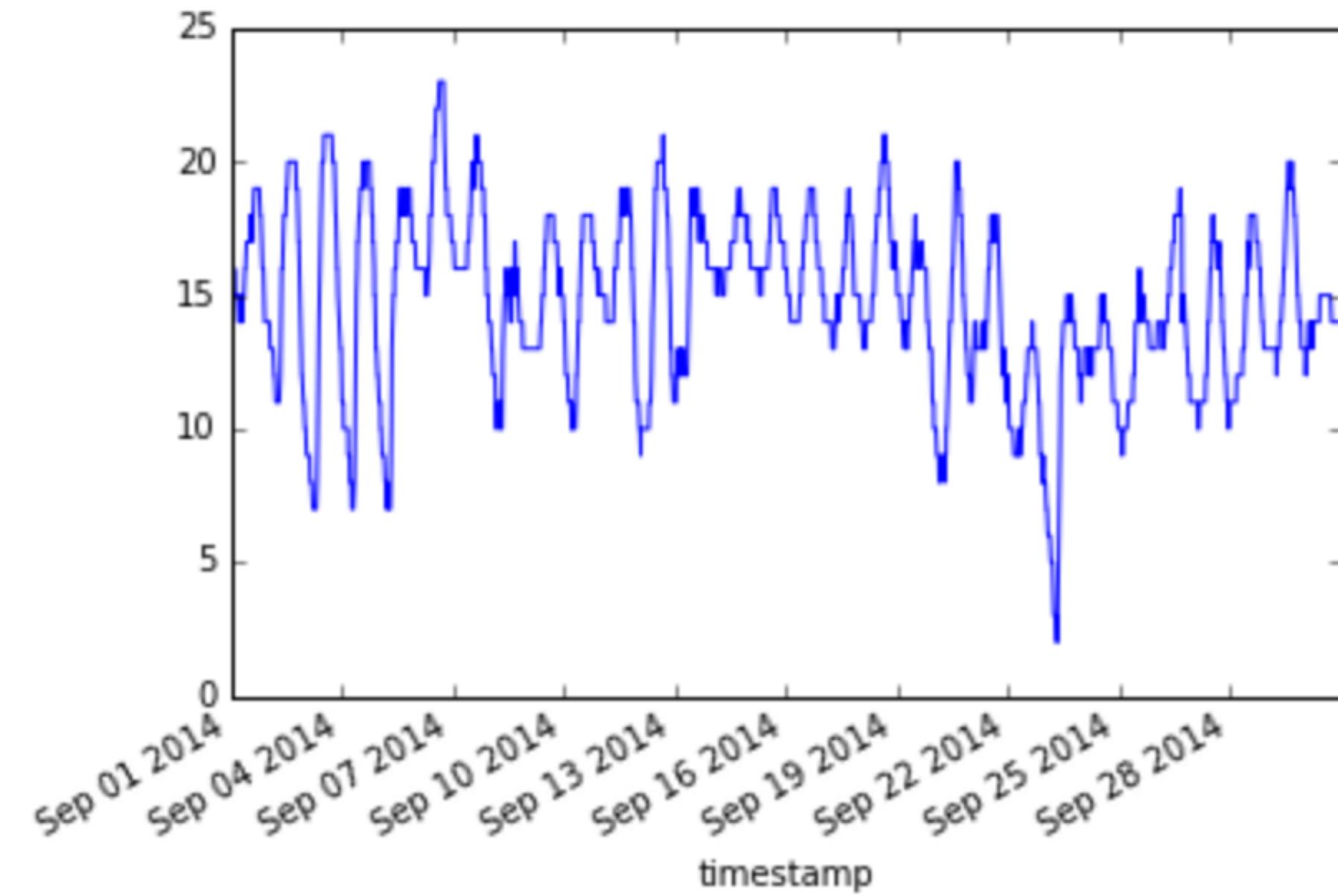
```
In [42]: df[df.index.weekday == True]['temperature'].plot()
```

```
Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x115e01828>
```



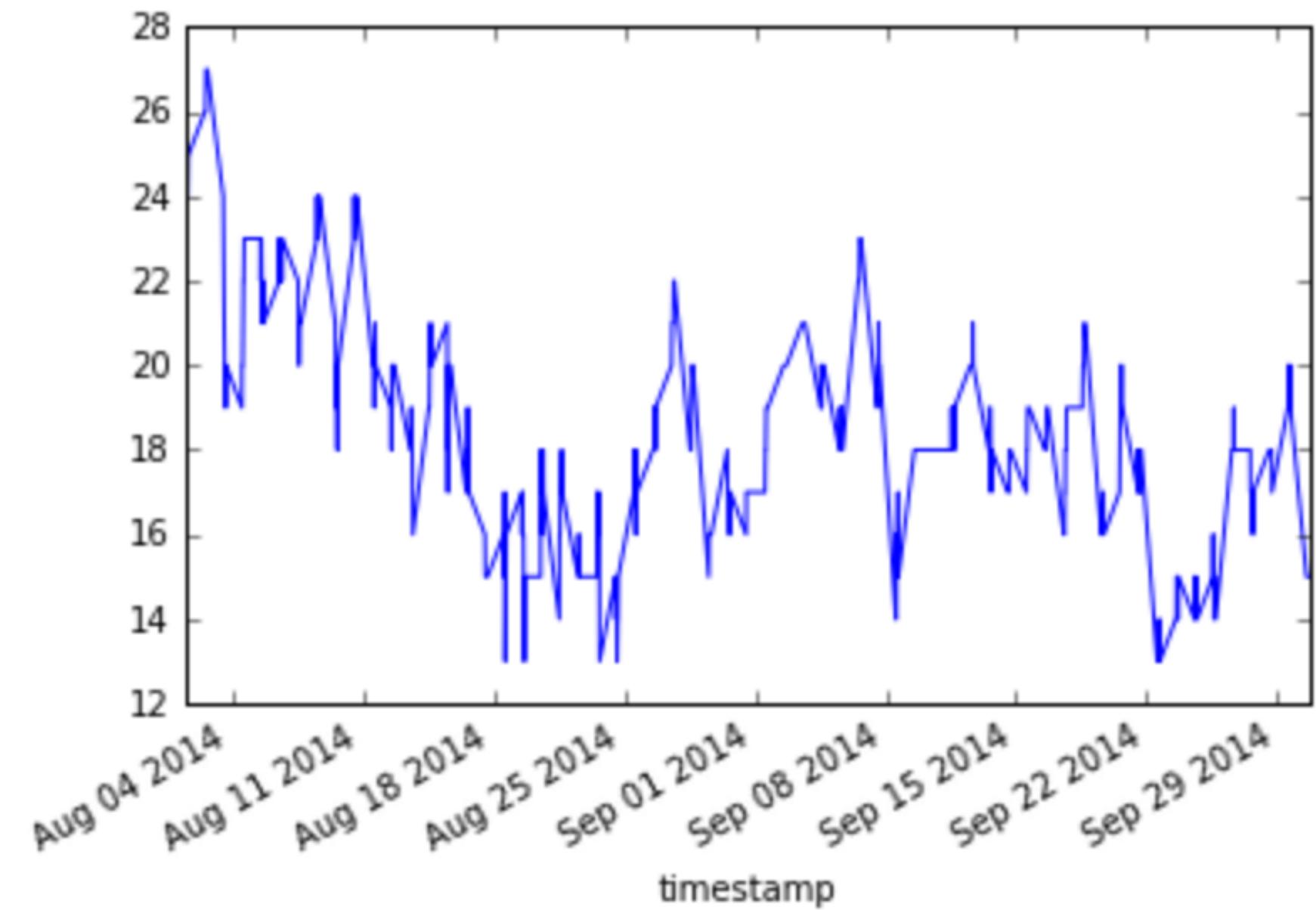
```
In [43]: df['2014-09']['temperature'].plot()
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1148d8eb8>
```



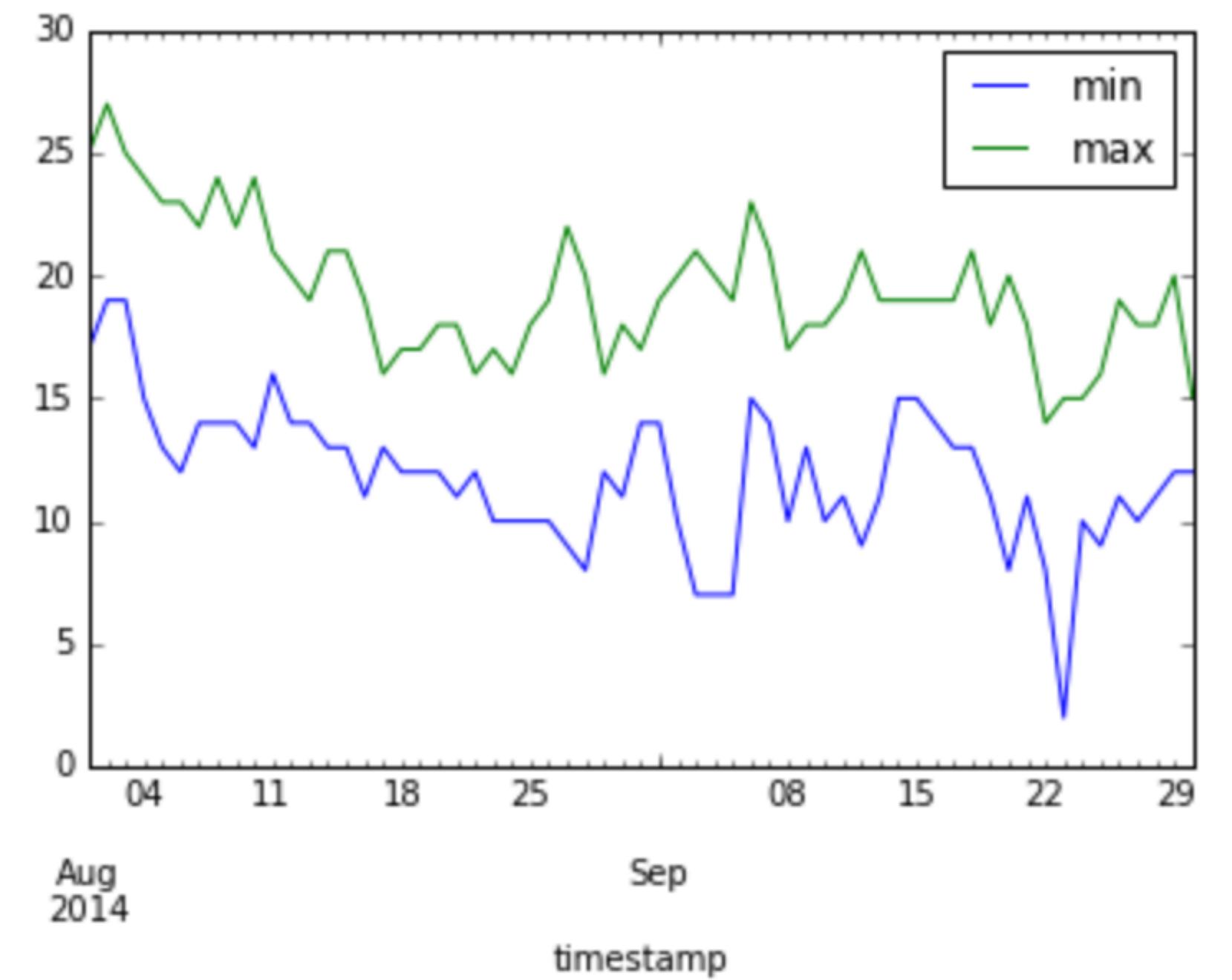
```
In [44]: df[(df.index.hour > 12) & (df.index.hour <=16)]['temperature'].plot()
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1148a3e10>
```



```
In [53]: df['temperature'].resample('D').agg(['min', 'max']).plot()
```

```
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x1167c0f60>
```



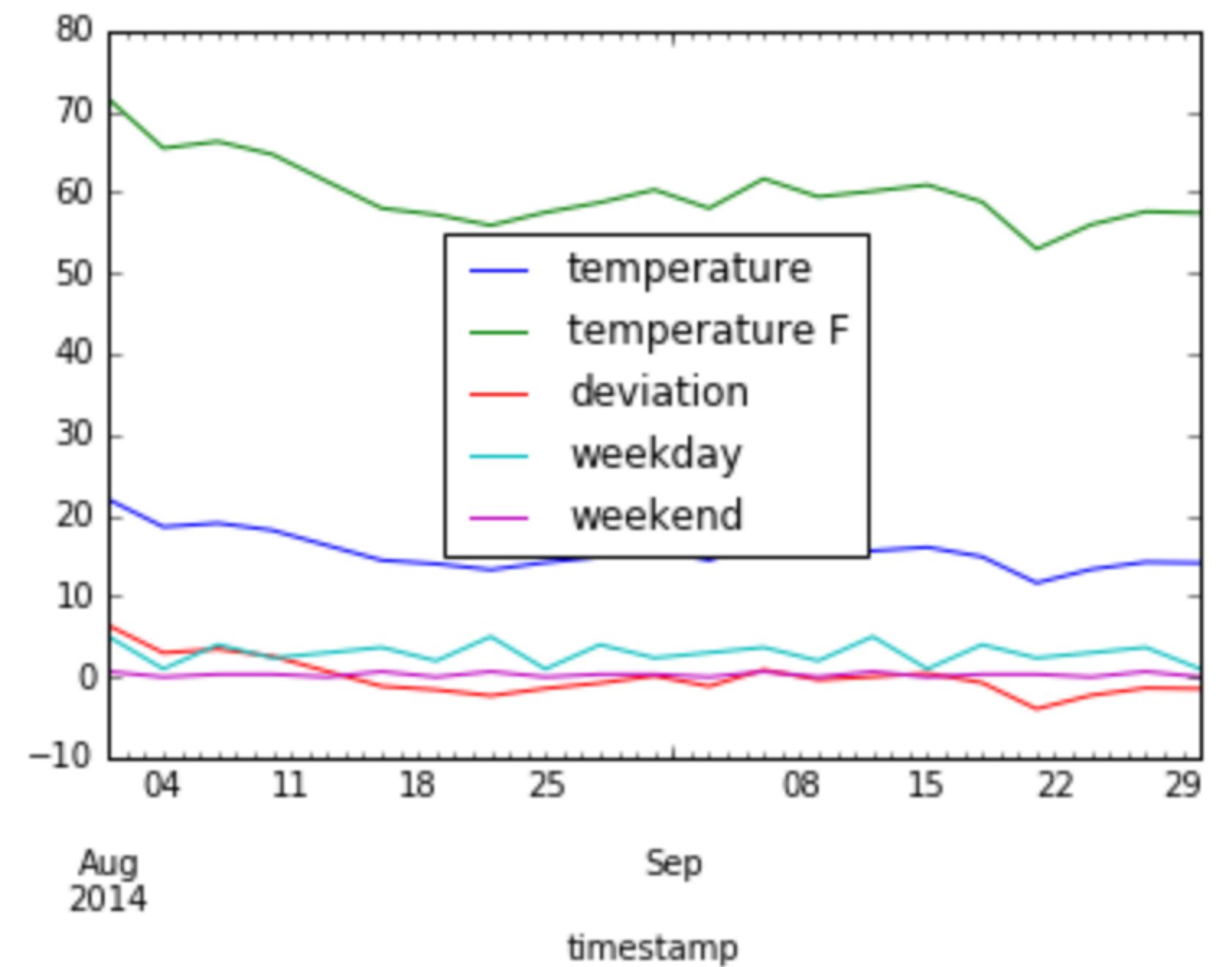


# Resampling

- H hourly frequency
- T minutely frequency
- S secondly frequency
- L milliseconds
- U microseconds
- N nanoseconds
  
- D calendar day frequency
- W weekly frequency
- M month end frequency
- Q quarter end frequency
- A year end frequency
  
- B business day frequency
- C custom business day frequency (experimental)
- BM business month end frequency
- CBM custom business month end frequency
- MS month start frequency
- BMS business month start frequency
- CBMS custom business month start frequency
- BQ business quarter endfrequency
- QS quarter start frequency
- BQS business quarter start frequency
- BA business year end frequency
- AS year start frequency
- BAS business year start frequency
- BH business hour frequency

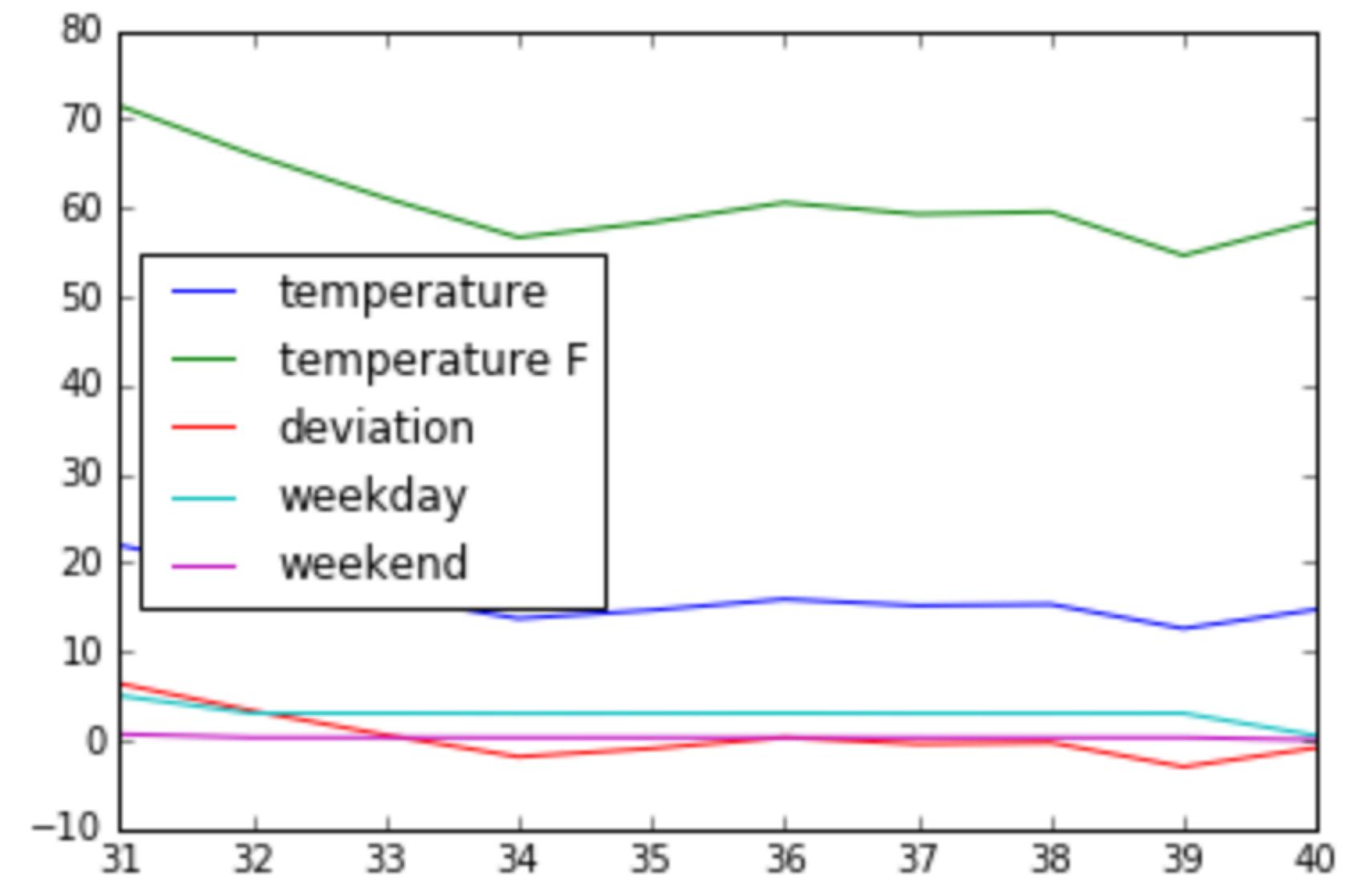
```
In [51]: df.resample('3D').mean().plot()
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x11480b128>
```



```
In [47]: df.groupby(df.index.week).mean().plot()
```

```
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x11481e7b8>
```



```
In [72]: import random  
index = pd.date_range('1/1/2016', periods=1200, freq='S')  
series = pd.Series([random.randint(0,100) for p in range(1200)], index=index)  
series
```

```
Out[72]: 2016-01-01 00:00:00    85  
2016-01-01 00:00:01    54  
2016-01-01 00:00:02    81  
2016-01-01 00:00:03     1  
2016-01-01 00:00:04   83  
2016-01-01 00:00:05   80  
2016-01-01 00:00:06   69  
2016-01-01 00:00:07   86  
2016-01-01 00:00:08     1  
2016-01-01 00:00:09   83  
2016-01-01 00:00:10   95  
2016-01-01 00:00:11   51  
2016-01-01 00:00:12   80  
2016-01-01 00:00:13     7  
2016-01-01 00:00:14   27  
2016-01-01 00:00:15     6  
2016-01-01 00:00:16   33  
2016-01-01 00:00:17   32  
2016-01-01 00:00:18     4  
2016-01-01 00:00:19   82  
2016-01-01 00:00:20   39  
2016-01-01 00:00:21   12  
2016-01-01 00:00:22   11  
2016-01-01 00:00:23   92  
2016-01-01 00:00:24   10
```

```
In [90]: resampled = series.resample('5T', label='right', closed='right').mean()  
resampled
```

```
Out[90]: 2016-01-01 00:00:00    85.000000  
2016-01-01 00:05:00    48.530000  
2016-01-01 00:10:00    48.290000  
2016-01-01 00:15:00    53.116667  
2016-01-01 00:20:00    47.953177  
Freq: 5T, dtype: float64
```

```
In [84]: upsampled = resampled.resample('90S').mean()[:6]
upsampled
```

```
Out[84]: 2016-01-01 00:00:00      85.0
2016-01-01 00:01:30      NaN
2016-01-01 00:03:00      NaN
2016-01-01 00:04:30    14559.0
2016-01-01 00:06:00      NaN
2016-01-01 00:07:30      NaN
Freq: 90S, dtype: float64
```

```
In [85]: upsampled = resampled.resample('90S').pad()[:6]
upsampled
```

```
Out[85]: 2016-01-01 00:00:00      85
2016-01-01 00:01:30      85
2016-01-01 00:03:00      85
2016-01-01 00:04:30      85
2016-01-01 00:06:00    14559
2016-01-01 00:07:30    14559
Freq: 90S, dtype: int64
```

```
In [86]: upsampled = resampled.resample('90S').bfill()[:6]
upsampled
```

```
Out[86]: 2016-01-01 00:00:00      85
2016-01-01 00:01:30    14559
2016-01-01 00:03:00    14559
2016-01-01 00:04:30    14559
2016-01-01 00:06:00    14487
2016-01-01 00:07:30    14487
Freq: 90S, dtype: int64
```

```
In [89]: def myresmapler(*args, **kwargs):
    return random.randint(1, 100)

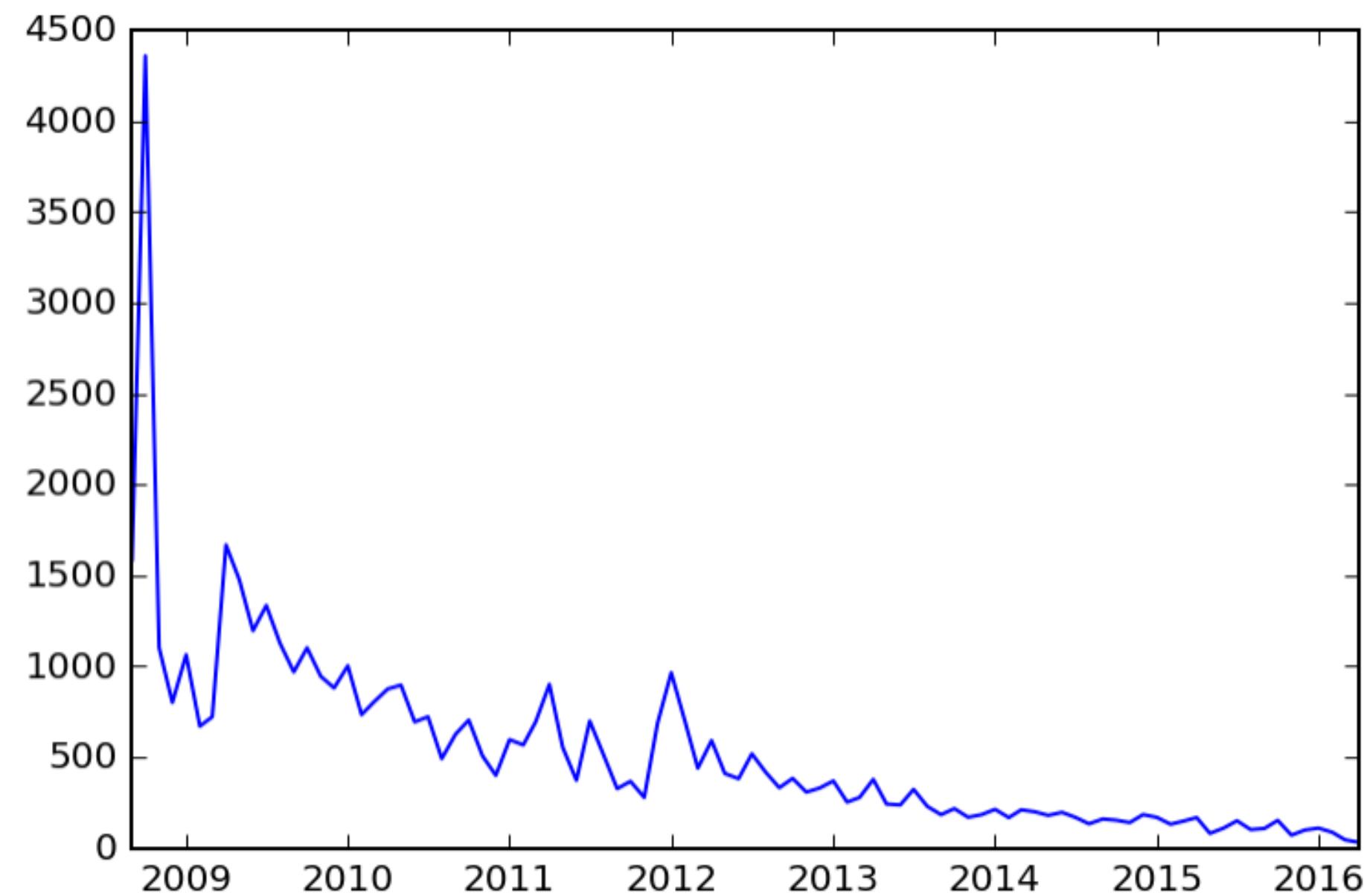
upsampled = resampled.resample('90S').apply(myresmapler)
upsampled
```

```
Out[89]: 2016-01-01 00:00:00      35
2016-01-01 00:01:30      94
2016-01-01 00:03:00      46
2016-01-01 00:04:30      55
2016-01-01 00:06:00      63
2016-01-01 00:07:30     99
2016-01-01 00:09:00      73
2016-01-01 00:10:30      54
2016-01-01 00:12:00      17
2016-01-01 00:13:30      10
2016-01-01 00:15:00      27
2016-01-01 00:16:30      13
2016-01-01 00:18:00      76
2016-01-01 00:19:30      67
Freq: 90S, dtype: int64
```



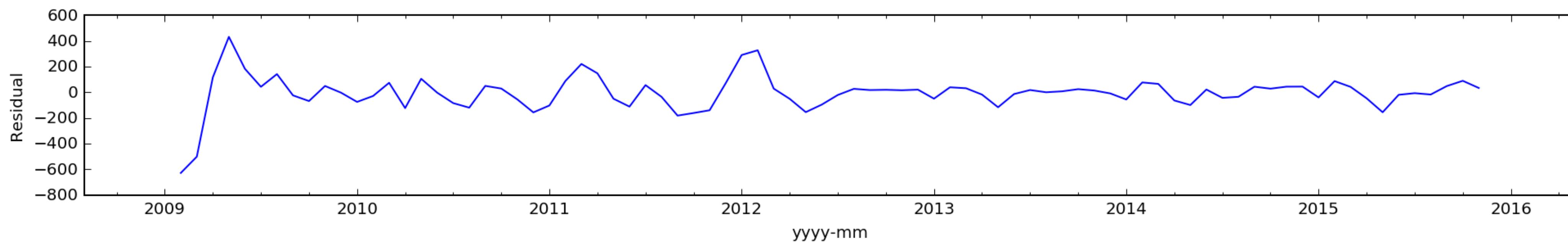
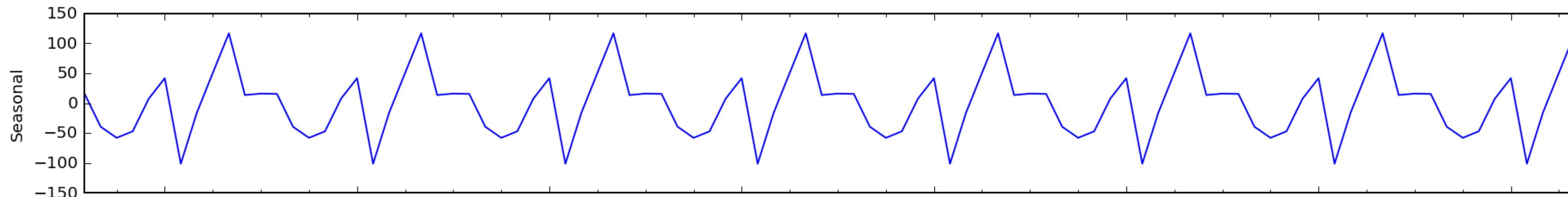
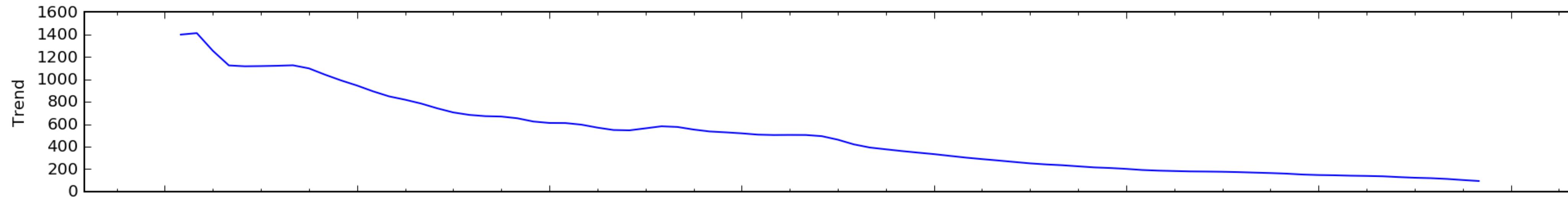
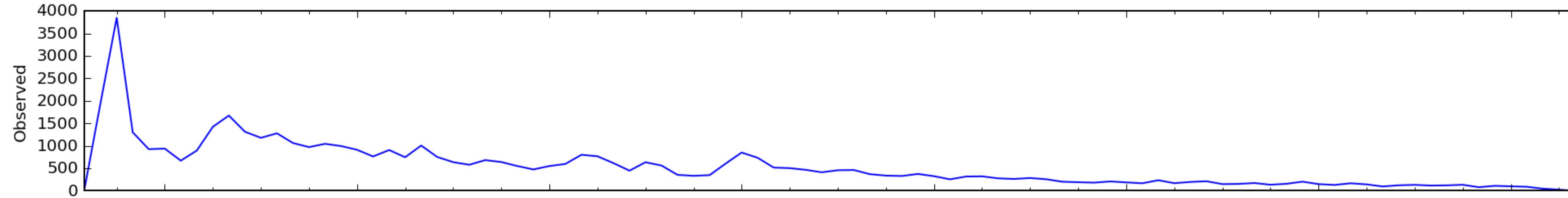
## Bonus: statsmodels

is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests



Some sales data of a single product

```
n [44]: dtap = pd.DataFrame(mdf.groupby(mdf.index)[ 'activity' ].sum())
# deal with missing values. see issue
dtap.activity.interpolate(inplace=True)
res = sm.tsa.seasonal_decompose(dtap.activity)
resplot = res.plot()
resplot.set_size_inches(15,15)
```



## Attributions

### Panda Picture

By Ailuropoda at en.wikipedia (Transferred from en.wikipedia) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) or CC BY-SA 2.5-2.0-1.0 (<http://creativecommons.org/licenses/by-sa/2.5-2.0-1.0>)], from Wikimedia Commons

# Alexander C. S. Hendorf

ah@koenigsweg.com

 @hendorf

Code-Examples

<https://github.com/Koenigsweg/data-timeseries-analysis-with-pandas>