ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Python for IoT Data Analytics

Time Series Processing with Pandas

EIT4SEMM PHD Program, July 2023

Prof. Marco Di Felice

Department of Computer Science and Engineering, University of Bologna

marco.difelice3@unibo.it

https://realpython.com/python-modules-packages/#python-packages

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modules

Modular programming: breaking a large coding task into separate, smaller, more manageable subtasks or **modules.**

- **Simplicity.** Focus on one relatively small portion of the problem.
- **Maintainability.** Modifications to a single portion will not produce an impact on other parts of the program.
- **Reusability:** Functionality defined in a single portion can be easily reused.
- **Scoping:** Modules typically define a separate *namespace*, which helps avoid collisions between identifiers in different areas of a program.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modules

A **Python module** is a file containing definitions and statements.

- Every module has a name equal to the file name.

  `mymodule` → `mymodule.py`

- Definitions from a module can be **imported** into other modules through the **`import`** statement (code reuse).
- Every module uses its **own namespace** (no variable definition conflicts with other Python script importing that modules).
- To speed up loading modules, Python caches the **compiled version** of each module in the `__pycache__` directory.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modules

mymod.py

```
def isEven(num):
    if (num % 2 == 0):
        return True
    else:
        return False
```

client.py

```
import mymod as mmod

If (mmod.isEven(5) == True):
    print("%d is even" %(num))
else:
    print("%d is odd" %(num))
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modules

In the previous example, the Python interpreter **searches for a module** named `mymod` in the following folders:

- The directory from which the input script is run
- The list of directories contained in the `PYTHONPATH` variable, if set
- An installation-dependent list of directories given by `sys.path` variable

```
import sys
print(sys.path)
```

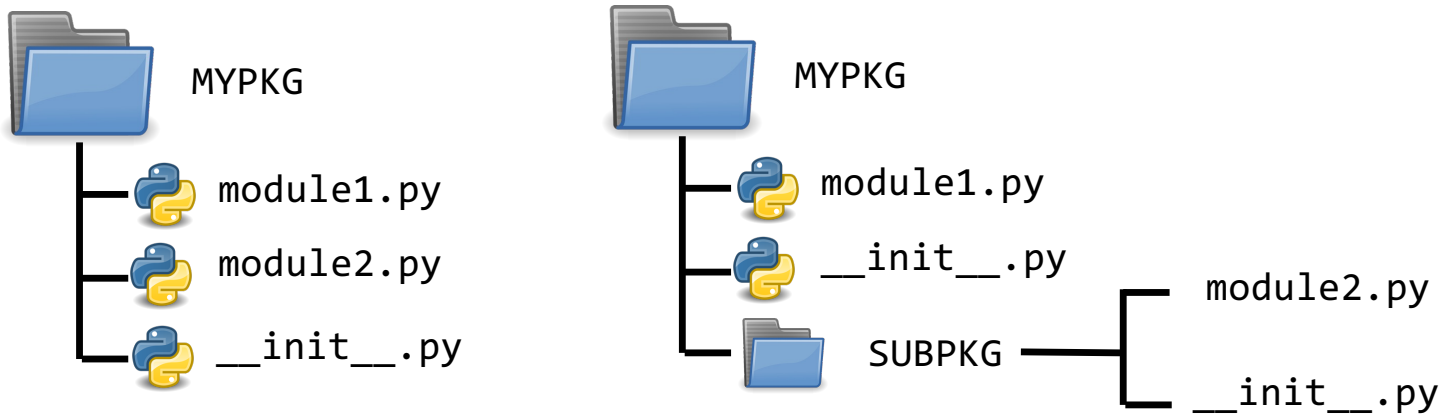
```
['/Users/marcodifelice/Documents/Coding_workspaces/Jupyter/',
'/Users/marcodifelice/miniforge3/envs/mlp/lib/python38.zip',
'/Users/marcodifelice/miniforge3/envs/mlp/lib/python3.8']
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Package

A **Python package** is a way of structuring modules related to the same domain/functionality.

- The package consists of a directory containing one or more modules.
- The directory must contain a file named `__init__.py`.
- This file contains the initialization code (can be left empty).



Built-in Modules

- File-system operations
  - Modules: `std` library, `os, pathlib`
- CLI & process management
  - Modules: `sys`,`argparse`, `invoke`
- Time operations
  - Modules: `time`, **`datetime`**
- Math operations
  - Modules: `math, random`
- Network operations
  - Modules: `http, request`
- Data structures operations
  - Modules: `pickle, collections`

https://docs.python.org/3/library/datetime.html

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Datatime Module

`Datatime module`: it provides functions for date/time manipulation.

- Allows time-related math operations (e.g. time difference)
- Supported classes:
  - `datetime.`**`date:`** `year, month, day`
  - `datetime.`**`time:`** `hour, minute, second, microsecond, tz`
  - `datetime.`**`datetime:`** combination of date + time
  - `datetime.`**`timedelta:`** differences between two datatimes
  - `datetime.`**`timezone:`** fixed offset from UTC
  - Get current time: `datetime.`**`date.today(),`**
    `datatime.`**`datetime.today`**`()`

Datatime Module

```
import datetime

print("Today is: ",datetime.date.today())
print("Today is: ",datetime.datetime.today())
print("Year: ",datetime.datetime.today().year)
```

```
$Today is: 2022-11-21
Today is: 2022-11-21 13:12:26.457889
Year: 2022
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Datatime Module

```
import datetime

tS = datetime.date(year = 2022, month = 11, day = 23)
tE = datetime.date(year = 2022, month = 11, day = 19)
tD = tS - tE
print("Date difference =", tD, type(tD))

tS = datetime.datetime(year = 2022, month = 11, day = 12, hour = 8, minute =
0, second = 40)
tE = datetime.datetime(year = 2022, month = 10, day = 7, hour = 7, minute = 0,
second = 10)
tD = tS - tE
print("Datetime difference =", tD, type(tD))
```

```
$Date difference = 4 days, 0:00:00 <class 'datetime.timedelta'>
Datetime difference = 36 days, 1:00:30 <class 'datetime.timedelta'>
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Datatime Module

- Convert string to a datetime object through the `datetime.strptime` method. It needs two parameters: (1) the string to be converted; (2) the format code (see examples).

%Y	Year (4 digit)	**%H**	Hour (24)
%m	Month (2 digit)	**%M**	Minute (2 digit)
%d	Day (2 digit)	**%S**	Second (00:59)

```
import datetime

value = "2023/07/12 08-12-23"
d = datetime.datetime.strptime(value,"%Y/%m/%d %H-%M-%S")
print(d)
```

```
2023-07-12 08:12:23
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Pandas contains **extensive** features for working with **time series** data

- It provides: (1) new types for time-related data; (2) time-specific methods (e.g. time slicing, lagging, resampling).
- It takes as input Python `datetime` object(s)
- Optimized and time-efficient implementation (based on Numpy)
- No additional packages required

Complete description of Python TS features available at:

https://pandas.pydata.org/docs/user_guide/timeseries.html

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Testing the Pandas efficiency for TS operations …

Dataset: `tetuan.csv`

Operation: extracting all temperature samples from Feb 2017 to July 2017

IMPLEMENTATION 1: Using Pandas `DatatimeIndex` (see next slides)

↓

Avg computation time (1000 repetions): **0.21 ms**

IMPLEMENTATION 2: Using Python `datetime` (prev slides)

↓

Avg computation time (1000 repetions): **24.34 ms**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - Shifting
  - Plotting

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas – TS types

- `Timestamp`

  Basic type of time series data that associates values with points in time

  Works similar to `datetime.datetime` Python type

```
Timestamp('2012-05-01 08:30:00')
```

- `Period`

  Timespans like days, months, quarters or years

```
Period('2011-01', 'M')
```

Pandas – TS types

- `Timestamp`

  In the simplest case, a time series can be represented as a generic Pandas **Series** indexed by a sequence of `Timestamps` (`DatatimeIndex`)

`DatatimeIndex`

index	value
2023-07-10	23.44
2023-07-11	23.45
2023-07-12	22.67
2023-07-13	21.89

`Timestamp` → Automatically converted from a `datetime` object

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas – TS types

- `Timestamp`

  In the simplest case, a time series can be represented as a Pandas Series indexed by a sequence of `Timestamps`

```
import datetime
import pandas as pd

dates  = [datetime.datetime(2023, 7, 10), datetime.datetime(2023, 7, 11),
datetime.datetime(2023, 7, 12)]
values = [10.12, 20.34, 44.12]
ser = pd.Series(values, index = dates)
print(ser.index)
```

```
DatetimeIndex(['2023-07-10', '2023-07-11', '2023-07-12'],
dtype='datetime64[ns]', freq=None)
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas – TS types

- Timestamp

  `DatatimeIndex` can be generated through the `data_range` method

```
index = pd.date_range("2023-07-10","2023-07-14", freq='D')
print(index)
```

```
DatetimeIndex(['2023-07-10', '2023-07-11', '2023-07-12', '2023-07-
13', '2023-07-14'], dtype='datetime64[ns]', freq='D')
```

```
index = pd.date_range("2023-07-10","2023-07-14", freq='2D')
print(index)
```

```
DatetimeIndex(['2023-07-10', '2023-07-12', '2023-07-14'],
dtype='datetime64[ns]', freq='2D')
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas – TS types

- `Timestamp`

  Base time series frequencies (only few examples)

Alias	Description
D	Day
BD	BusinessDay
H	Hour
T	Minute
S	Second
M	Month End
MS	MonthBegin
W, W-MON, W-TUE	Weekly on given day of the week

https://pandas.pydata.org/docs/user_guide/timeseries.html#dateoffset-objects

Pandas – TS types

- Timestamp

  `pandas.to_datetime` converts a sequence of values into a `DatatimeIndex`

```
df = pd.read_csv("tetuan.csv")
df["DateTime"] = pd.to_datetime(df["DateTime"])
df = df.set_index('DateTime')
print(df.index)
```

```
DatetimeIndex(['2017-01-01 00:00:00', '2017-01-01 00:10:00', '2017-
01-01 00:20:00', '2017-01-01 00:30:00', ….])
```

In a more compact way:

```
df = pd.read_csv("tetuan.csv", parse_dates = ["DateTime"],
                 index_col = "DateTime")
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas – TS types

- Timestamp

  `pandas.to_datetime` converts a sequence of values into a `DatatimeIndex`
  The time unit can be specified through the argument.

```
tsList = [1689125087, 1689125387, 1689125687, 1689125987]
ts = pd.to_datetime(tsList, unit='s', origin='unix')
values = [0, 1, 2, 3]
ser = pd.Series(values, index = ts)
print(ser)
```

```
2023-07-12 01:24:47 0
2023-07-12 01:29:47 1
2023-07-12 01:34:47 2
2023-07-12 01:39:47 3
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - **`Period`**
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - Shifting
  - Plotting

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas – TS types

- Period

  PeriodIndex can be generated through the period_range method

```
p = pd.period_range("2022-01", "2022-05", freq = 'M')
val = [10, 20, 30, 40, 50]
ser = pd.Series(val, index = p)
print(ser.index)
```

```
PeriodIndex(['2022-01', '2022-02', '2022-03', '2022-04', '2022-05'],
dtype='period[M]')
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - **Data selection**
  - Resampling
  - Downsampling
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - Shifting
  - Plotting

Pandas for TS: Overview

❑ Data selection

Dates and strings that parse to timestamps can be passed as indexing parameters, like for generic pandas Series

Single value selection:

```
print(df.loc["2017-01-05 08:00:00"])
```

Range selection:

```
print(df.loc["2017-01-05 08:00:00":"2017-01-05 10:00:00"])
```

Partial match selection: → RULE: If the string is less accurate than the index, it will be treated as a slice, otherwise as an exact match.

```
print(df.loc["2017-01"])
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - **Resampling**
  - Downsampling
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - Shifting
  - Plotting

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

❑ Resampling

Resampling refers to the process of converting a time series from one frequency to another. All frequency conversions can be handled through the `resample` method.

```
ts = pd.date_range("2023-01-01", "2023-05-01", freq = "M")
val = [100, 100, 100, 100]
ser = pd.Series(val, index = ts)
ts2 = ser.resample("D")
print(type(ts2))
for elem in ts2:
    print(elem)
```

```
<class 'pandas.core.resample.DatetimeIndexResampler'>
(Timestamp('2023-01-31 00:00:00', freq='D'), 2023-01-31 100 Freq:
M, dtype: int64)
(Timestamp('2023-02-01 00:00:00', freq='D'), Series([], Freq: M,
dtype: int64))
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

❑ Resampling

Resampling refers to the process of converting a time series from one frequency to another. We distinguish between:

- **Downsampling**

  Aggregating frequency data to lower frequency.

  Conceptually similar to a goupby operator (with time-related groups).

- **Upsampling**

  Converting lower frequency to higher frequency.

  Used often in conjunction with *interpolation* methods.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - **Downsampling**
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - Shifting
  - Plotting

Pandas for TS: Overview

Downsampling

Aggregating frequency data to lower frequency, by slicing the series into intervals of equal length; each point can belong to one interval only.

Parameters to be configured:

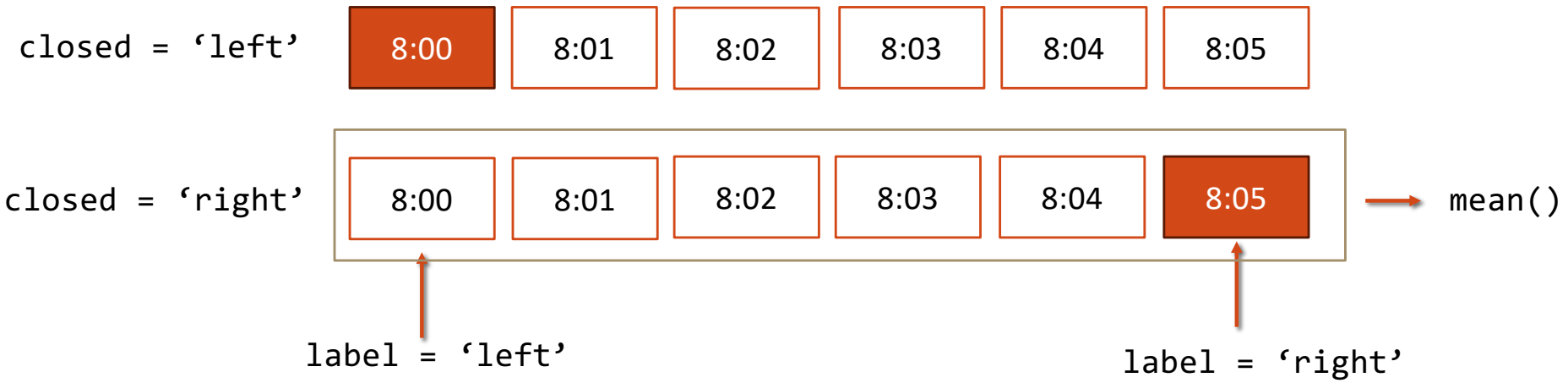
- Which side of each interval is closed (intervals are half-open)
- The operator to apply to each bin (e.g., the mean)
- How to label each bin (start of the interval or end)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Downsampling

Aggregating frequency data to lower frequency, by slicing the series into intervals of equal length; each point can belong to one interval only.



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Downsampling

Aggregating frequency data to lower frequency, by slicing the series into intervals of equal length; each point can belong to one interval only.

```
ts = pd.date_range("2023-01-01", "2024-01-01", freq = "M")
val = [i for i in range(0, 12)]
ser = pd.Series(val, index = ts)
print(ser)
serU = ser.resample("4M", closed = "left", label = "left").mean()
print(serU)
```

```
2023-01-31 1.5
2023-05-31 5.5
2023-09-30 9.5
Freq: 4M, dtype: float64
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - **Oversampling**
  - Moving Average
  - Time-zone Handling
  - Shifting
  - Plotting

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

❑ Oversampling

Aggregating frequency data to higher frequency; no aggregation is needed, however interpolation may be useful to replace the missing values.

```
ts = pd.date_range("2023-01-01", "2024-01-01", freq = "M")
val = [i for i in range(0, 12)]
ser = pd.Series(val, index = ts)
serO = ser.resample("D").interpolate(method="quadratic")
print(serO)
```

```
2023-01-31 0.000000
2023-02-01 0.037706
2023-02-02 0.075265
2023-02-03 0.112676
2023-02-04 0.149940
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

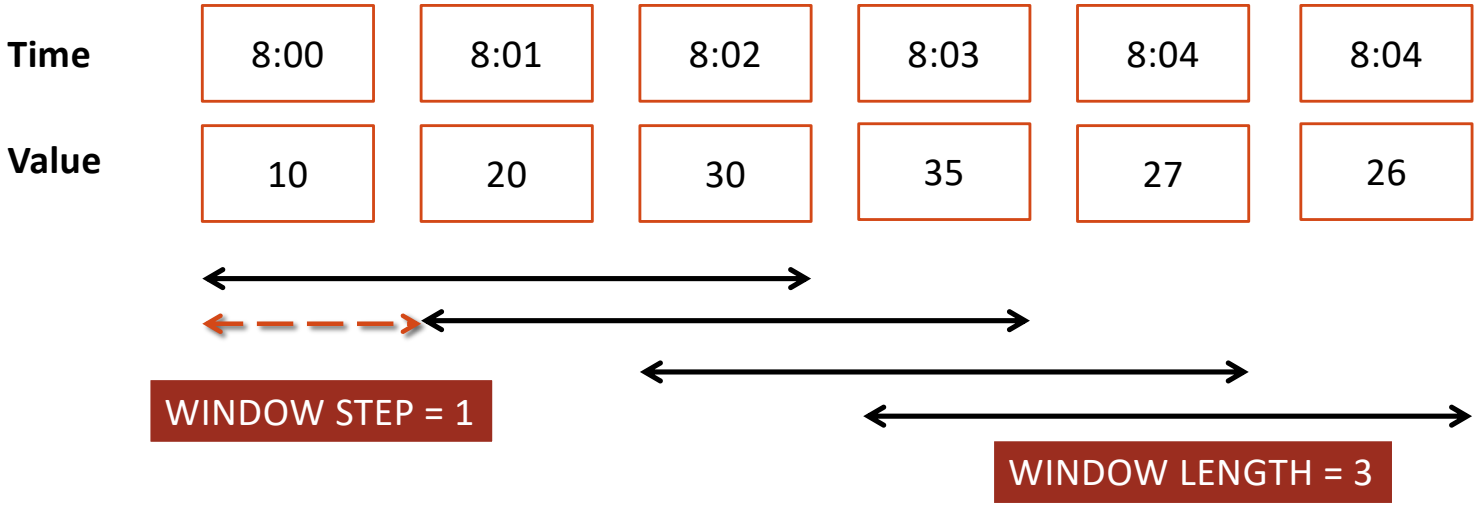
Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - Oversampling
  - **Moving Average**
  - Time-zone Handling
  - Shifting
  - Plotting

Pandas for TS: Overview

Rolling

Moving window functions for smoothing noisy data via fixed length windows.



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Rolling

Moving window functions for smoothing noisy data via fixed length windows.

```
ts = pd.date_range("2023-01-01", "2024-01-01", freq = "M")
val = [i for i in range(0, 12)]
ser = pd.Series(val, index = ts)
print(ser.rolling(4).mean())
print(ser.rolling("30D").mean())
```

```
2023-01-31 NaN
2023-02-28 NaN
2023-03-31 NaN
2023-04-30 1.5
2023-05-31 2.5
2023-06-30 3.5
2023-07-31 4.5
```

Fixed number of observations used for each window.

Time period of each window

Pandas for TS: Overview

❑ Weighted Functions

Instead of using a static window size, we can specify a constant decay factor to give more weight to more recent observations. This can be achieved through the ewm operator, which takes in input the span length.

```
df = pd.read_csv("tetuan.csv", parse_dates = ["DateTime"], index_col =
                 "DateTime")
serU = df["Temperature"].ewm(span = 500).mean()
serU2 = df["Temperature"].rolling(500).mean()
```

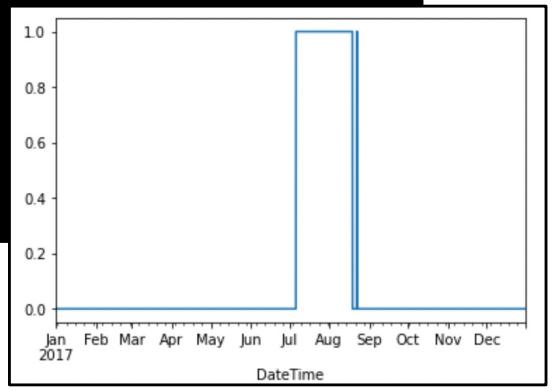
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

User-defined Moving Windows Functions

Through the apply window, users can define their custom aggregation function to be applied on each moving window. The only requirement is that the function must produce a single value (reduction process) at each invokation.

```
df = pd.read_csv("tetuan.csv", parse_dates = ["DateTime"], index_col =
"DateTime")
hotweeks = lambda x: 0 if (x.mean()<25) else 1
serU = df["Temperature"].rolling("7d").apply(hotweeks)
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - Oversampling
  - Moving Average
  - **Time-zone Handling**
  - Shifting
  - Plotting

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

❑ Time zone handling

By default, pandas objects are time zone unaware. We can get the time zone info through the `tz` property of a `TimeSeries` or `DatetimeIndex` object

```
ts = pd.date_range("2023-01-01", "2024-01-01", freq = "M")
print(ts.tz)
```

```
None
```

```
ts = pd.date_range("2023-01-01", "2024-01-01", freq = "M",
                   tz="Europe/London")
print(ts.tz)
```

```
Europe/London
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Time zone handling

The `tz_convert` method allows to convert time values into a different time zone.

```
ts = pd.date_range("2023-01-01 08:00:00", "2023-01-01 08:20:00",
tz="Europe/London", freq = "5min")
ts = ts.tz_convert("US/Eastern")
val = [i for i in range(0, len(ts))]
ser = pd.Series(val, index = ts)
print(ser)
```

```
2023-01-01 03:00:00-05:00 0
2023-01-01 03:05:00-05:00 1
2023-01-01 03:10:00-05:00 2
2023-01-01 03:15:00-05:00 3
2023-01-01 03:20:00-05:00 4
Freq: 5T, dtype: int64
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - **Shifting**
  - Plotting

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

❑ Shifting / Lagging

The `shift` method allows to shift the values in a time series back and forward in time.

```
ts = pd.date_range("2023-01-01 08:00:00", "2023-01-01 08:20:00",
tz="Europe/London", freq = "5min")
val = [i for i in range(0, len(ts))]
ser = pd.Series(val, index = ts)
print(ser.shift(2))
print(ser.shift(2, freq = "5min"))
```

```
2023-01-01 08:00:00+00:00 NaN
2023-01-01 08:05:00+00:00 NaN
2023-01-01 08:10:00+00:00 0.0
2023-01-01 08:15:00+00:00 1.0
2023-01-01 08:20:00+00:00 2.0
```

```
2023-01-01 08:10:00+00:00 0
2023-01-01 08:15:00+00:00 1
2023-01-01 08:20:00+00:00 2
2023-01-01 08:25:00+00:00 3
2023-01-01 08:30:00+00:00 4
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

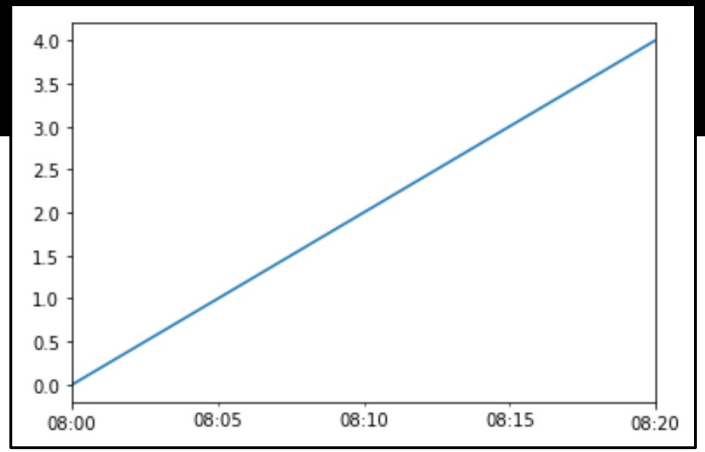
- **Built-in data structures**
  - `Timestamp`
  - `Period`
- **Built-in operations**
  - Data selection
  - Resampling
  - Downsampling
  - Oversampling
  - Moving Average
  - Time-zone Handling
  - Shifting
  - **Plotting**

Pandas for TS: Overview

Plotting

The `plot` method allows to plot a Timeseries, by using the `matplotlib` library.

```
ts = pd.date_range("2023-01-01 08:00:00", "2023-01-01 08:20:00",
tz="Europe/London", freq = "5min")
val = [i for i in range(0, len(ts))]
ser = pd.Series(val, index = ts)
ser.plot()
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Pandas for TS: Overview

Plotting

The `plot` method allows to plot a Timeseries, by using the `matplotlib` library.

Some parameters:

- `kind`: kind of the plot (bar, line, hist, box, scatter, area…)
- `title`: title of the plot
- `xlabel`: label of the xaxis
- `ylabel`: label of the yaxis
- `grid`: (boolean) use grid
- ….

Complete list of parameters available here:.

https://pandas.pydata.org/docs/reference/api/pandas.Series.plot.html