wax-toolbox

Sep 08, 2018

Notes

Generic Python Tools

CHAPTER 1

QuickStart

```
import wax_toolbox
```

CHAPTER 2

Timeserie Analytics

Examples:

```
In [1]: from wax_toolbox.tsanalytics import analyse_datetimeindex

In [2]: idx_gap  # let's look at that pd.DatetimeIndex
Out[2]:
DatetimeIndex(['2016-03-01 02:00:00+01:00', '2016-03-01 03:00:00+01:00',
               '2016-03-01 04:00:00+01:00', '2016-03-01 07:00:00+01:00',
               '2016-03-01 08:00:00+01:00', '2016-03-01 09:00:00+01:00',
               '2016-03-01 14:00:00+01:00', '2016-03-01 15:00:00+01:00',
               '2016-03-01 16:00:00+01:00', '2016-03-01 17:00:00+01:00',
               ...
               '2016-03-30 15:00:00+02:00', '2016-03-30 16:00:00+02:00',
               '2016-03-30 17:00:00+02:00', '2016-03-30 18:00:00+02:00',
               '2016-03-30 19:00:00+02:00', '2016-03-30 20:00:00+02:00',
               '2016-03-30 21:00:00+02:00', '2016-03-30 22:00:00+02:00',
               '2016-03-30 23:00:00+02:00', '2016-03-31 00:00:00+02:00'],
              dtype='datetime64[ns, CET]', length=712, freq=None)

In [3]: tsinfo = analyse_datetimeindex(idx_gap, start=start, end=end)

In [4]: tsinfo
Out[4]:
freq: <Hour>
sorted: True
continuous: [(Timestamp('2016-03-01 02:00:00+0100', tz='CET'), Timestamp('2016-03-01␣
↪04:00:00+0100', tz='CET')), (Timestamp('2016-03-01 07:00:00+0100', tz='CET'),␣
↪Timestamp('2016-03-01 09:00:00+0100', tz='CET')), (Timestamp('2016-03-01␣
↪14:00:00+0100', tz='CET'), Timestamp('2016-03-31 00:00:00+0200', tz='CET'))]
gaps: [(Timestamp('2016-03-01 00:00:00+0100', tz='CET'), Timestamp('2016-03-01␣
↪01:00:00+0100', tz='CET')), (Timestamp('2016-03-01 05:00:00+0100', tz='CET'),␣
↪Timestamp('2016-03-01 06:00:00+0100', tz='CET')), (Timestamp('2016-03-01␣
↪10:00:00+0100', tz='CET'), Timestamp('2016-03-01 13:00:00+0100', tz='CET'))]
duplicates: []
```

(continues on next page)

(continued from previous page)

```
In [5]: print('This timeserie got a (minimal) frequency of {}'.format(tsinfo.freq))
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪timeserie got a (minimal) frequency of <Hour>

In [6]: print('This timeserie is sorted ? {}'.format(tsinfo.sorted))
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪timeserie is sorted ? True

# continuous parts:
In [7]: tsinfo.continuous
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪
[(Timestamp('2016-03-01 02:00:00+0100', tz='CET'),
  Timestamp('2016-03-01 04:00:00+0100', tz='CET')),
 (Timestamp('2016-03-01 07:00:00+0100', tz='CET'),
  Timestamp('2016-03-01 09:00:00+0100', tz='CET')),
 (Timestamp('2016-03-01 14:00:00+0100', tz='CET'),
  Timestamp('2016-03-31 00:00:00+0200', tz='CET'))]

# gaps parts:
In [8]: tsinfo.gaps
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪
[(Timestamp('2016-03-01 00:00:00+0100', tz='CET'),
  Timestamp('2016-03-01 01:00:00+0100', tz='CET')),
 (Timestamp('2016-03-01 05:00:00+0100', tz='CET'),
  Timestamp('2016-03-01 06:00:00+0100', tz='CET')),
 (Timestamp('2016-03-01 10:00:00+0100', tz='CET'),
  Timestamp('2016-03-01 13:00:00+0100', tz='CET'))]

# duplicates if any:
In [9]: tsinfo.duplicates
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪[]
```

A module with timeseries analysis tools.

class wax_toolbox.tsanalytics.**TSAnalytics**(*freq*, *sorted*, *continuous*, *gaps*, *duplicates*)

Wrapper for time serie analysis results.

Parameters

- **freq** (*str*) – frequency
- **sorted** (*bool*) – whether timeindex is sorted.
- **continuous** (*list of datetime tuples*) – continuous segments.
- **gaps** (*list of datetime tuples*) – gaps segments.
- **duplicates** (*list of datetime*) – duplicated index.

wax_toolbox.tsanalytics.**analyse_datetimeindex**(*idx*, *start=None*, *end=None*, *freq=None*)

Check if the given index is of type DatetimeIndex & is aware. Returns the implied frequency, a sorted flag, the list of continuous segment, the list of gap segments and the list of duplicated indices. Continuous and gaps segments are expressed as [start:end] (both side inclusive). If the index is not sorted, it will be sorted before checking for continuity. Specifying start and end check for gaps at beginning and end of the index. Specifying freq enforces control of gaps according to frequency.

Parameters

- **idx** (*pd.DatetimeIndex*) – datetimeindex aware to be analysed
- **start** (*datetime expression*) – from when to start the analysis. Defaults to None, which means from the lower bound of idx.
- **start** (*datetime expression*) – from when to end the analysis. Defaults to None, which means from the upper bound of idx.
- **freq** (*str*) – analyise on this frequency. Defaults to None, which means the idx actual frequency.

Returns *(TSAnalytics namedtuple)* – freq, sorted, continuous, gaps, duplicates

wax_toolbox.tsanalytics.**detect_frequency**(*idx*)

Return the most plausible frequency of pd.DatetimeIndex (even when gaps in it). It calculates the delta between element of the index (idx[1:] - idx[:1]), gets the 'mode' of the delta (most frequent delta) and transforms it into a frequency ('H','15T',...)

Parameters **idx** (*pd.DatetimeIndex*) – datetime index to analyse.

Returns frequency (str)

Note: A solution exists in pandas:

```
from pandas.tseries.frequencies import _TimedeltaFrequencyInferer
inferer = _TimedeltaFrequencyInferer(idx)
freq = inferer.get_freq()
```

But for timeseries with nonconstant frequencies (like for 'publication_date' of forecast timeseries), then the inferer.get_freq() return None.

In those cases, we are going to return the smallest frequency possible.

---

wax_toolbox.tsanalytics.**get_tz_info**(*tzname*, *limit_year=2000*)

Get DST informations.

Parameters

- **tzname** (*str*) – a timezone.
- **limit_year** (*int*) – filter the DST transitions datetimes older than this given year.

Returns

(tuple) –

2-elements tuple containing:

- tz (pytz.timezone): the converted string into timezone object.
- df (pd.DataFrame): dataframe containing DST informations.

```
In [1]: from wax_toolbox.tsanalytics import get_tz_info

In [2]: tz, df = get_tz_info('CET')

In [3]: tz
Out[3]: <DstTzInfo 'CET' CET+1:00:00 STD>

In [4]: df.head(10)
```

(continues on next page)

(continued from previous page)

```
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[4]:
   dstoffset            timestamp
61  01:00:00  2000-03-26 01:00:00
62  00:00:00  2000-10-29 01:00:00
63  01:00:00  2001-03-25 01:00:00
64  00:00:00  2001-10-28 01:00:00
65  01:00:00  2002-03-31 01:00:00
66  00:00:00  2002-10-27 01:00:00
67  01:00:00  2003-03-30 01:00:00
68  00:00:00  2003-10-26 01:00:00
69  01:00:00  2004-03-28 01:00:00
70  00:00:00  2004-10-31 01:00:00
```

wax_toolbox.tsanalytics.**tz_convert_multiindex**(*ts*, *to_tz='UTC'*)

Convert all aware indexes of multiIndex timeserie. It also checks first if the indexes are effectively aware.

Parameters

- **ts** (*pd.Series with pd.DatetimeIndex*) – timeserie with multiindex.
- **to_tz** (*str*) – timezone to be converted into.

Returns (pd.Series) with timezone converted.

wax_toolbox.tsanalytics.**tz_fix**(*df*, *time_col*, *from_tz='Europe/Brussels'*, *split_by=None*, *dropval_on_fail=False*)

Try to fix the timezone of a datetime column.

Parameters

- **df** (*pd.DataFrame*) – dataframe to process.
- **time_col** (*str*) – name of the column to be processed.
- **from_tz** (*str*) – initial timezone of the naive time_col. Defaults to 'Europe/Brussels'
- **split_by** (*str*) – Name of the column to split by. It is necessary when the dataframe go several series. Defaults to None.
- **dropval_on_fail** (*bool*) –
  - if false, raise TzFixFail if couldn't resolve
  - if true, drop dst values if case of TzFixFail.

  Defaults to False.

wax_toolbox.tsanalytics.**tz_localize_multiindex**(*ts*, *from_tz='UTC'*)

Localize all naive indexes of multiIndex timeserie. It also checks first if the indexes are effectively naives.

Parameters

- **ts** (*pd.Series with pd.DatetimeIndex*) – timeserie with multiindex.
- **from_tz** (*str*) – timezone to be localized into.

Returns (pd.Series) with localized mutliindex.

CHAPTER 3

Profiling

class wax_toolbox.profiling.**Timer**(*label*, *at_enter=False*, *report=<built-in function print>*)

Simple timer focused on practical use.

Parameters

- **label** (*str*) – label of the timer
- **at_enter** (*bool*) – whether it should be also displayed when entering the context. Defaults to False.
- **report** (*func*) – function to use for reporting. Defaults to logger.info

Example usage:

```
In [1]: import asyncio

In [2]: import time

In [3]: from concurrent.futures import ThreadPoolExecutor

In [4]: from wax_toolbox import profiling

In [5]: async def waiter():
   ...:     await asyncio.sleep(5)
   ...:     return
   ...:

In [6]: def secondwaiter():
   ...:     time.sleep(5)
   ...:     return
   ...:

In [7]: N = 5

In [8]: with profiling.Timer("Asyncio", at_enter=True, report=print):
   ...:     loop = asyncio.get_event_loop()
```

(continues on next page)

(continued from previous page)

```
   ...:     asyncio.set_event_loop(loop)
   ...:     tasks = []
   ...:     for i in range(N):
   ...:         tasks.append(asyncio.ensure_future(waiter()))
   ...:     loop.run_until_complete(asyncio.wait(tasks))
   ...:     loop.close()
   ...:
Asyncio in progress...
Asyncio took 5.006 sec

In [9]: with profiling.Timer("ThreadPoolExecutor", at_enter=True, report=print):
   ...:     futures = []
   ...:     with ThreadPoolExecutor(max_workers=4) as e:  # have you spotted it ?
   ...:         for i in range(N):
   ...:             futures.append(e.submit(secondwaiter))
   ...:     for i in range(len(futures)):
   ...:         futures[i].result()
   ...:
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ThreadPoolExecutor in progress...
ThreadPoolExecutor took 10.011 sec
```

CHAPTER 4

Miscelleneous Fixtures

Miscelleneous fixtures contains tools for easy testing on python requests using Betamax.

4.1 Usage

In conf.py:

```
import betamax
from wax_toolbox.misc_fixtures import sanitize_token, RecorderBase

from betamax_serializers import pretty_json

CASSETTES_DIR = Path(__file__).parent / 'cassettes'
SAMPLES_DIR = Path(__file__).parent / 'samples'

# Betamax configuration:
with betamax.Betamax.configure() as config:
    config.cassette_library_dir = CASSETTES_DIR.as_posix()
    betamax.Betamax.register_serializer(pretty_json.PrettyJSONSerializer)
    config.default_cassette_options['serialize_with'] = 'prettyjson'
    config.before_record(callback=sanitize_token)
    # config.default_cassette_options['match_requests_on'].extend([
    #     'json_body'
    # ])


def pytest_addoption(parser):
    parser.addoption('--generate-samples', action='store_true', default=False,
                     help='''Generate cassettes & samples for new tests.
                     The existing ones will remain unchanged.''')
    parser.addoption('--regenerate-samples', action='store_true', default=False,
                     help='Regenerate cassettes & samples for all tests.')
```

(continues on next page)

(continued from previous page)

```
@pytest.fixture(scope='session')
def recorder(request):
    BETAMAX_MODE = 'none'  # test existing cassettes
    if request.config.getoption("--generate-samples"):  # generate samples and␣
↪cassettes for new tests
        BETAMAX_MODE = 'once'
    if request.config.getoption("--regenerate-samples"):  # regenerate samples and␣
↪cassettes for all tests
        BETAMAX_MODE = 'all'

    class Recorder(RecorderBase):
        def __init__(self, bucket_name, session=requests.Session()):
            super().__init__(bucket_name=bucket_name,
                             sample_dir=SAMPLE_DIR
                             session=session,
                             betamax_mode=BETAMAX_MODE
                             )

    yield Recorder


@pytest.fixture(scope='session', params=[
    # Old ones:
    (pd.Timestamp('2016-01-01T00:00:00', tz='CET'),
    pd.Timestamp('2016-01-03T00:00:00', tz='CET')),

    # DST change winter -> summer:
    (pd.Timestamp('2017-03-26T00:00:00', tz='CET'),
    pd.Timestamp('2017-03-27T00:00:00', tz='CET')),

    # DST change summer -> winter:
    (pd.Timestamp('2017-10-29T00:00:00', tz='CET'),
    pd.Timestamp('2017-10-30T00:00:00', tz='CET')),

    # Early 2018:
    (pd.Timestamp('2018-01-21T00:00:00', tz='UTC'),
    pd.Timestamp('2018-01-23T00:00:00', tz='CET')),

])
def period(request):
    dct = {'start': request.param[0], 'end': request.param[1]}
    return dct
```

Then in some test_api.py:

```
def test_api(period, recorder):
    client = InitClientApi()

    bucket_name = "api_endpoint_{}_{}".format(
        period['start'].strftime(dtformat),
        period['end'].strftime(dtformat),
    )

    with recorder(bucket_name=bucket_name, session=client.session) as r:
        df = client.get_total_load_forecast(**period)
        r.send_dataframe(df)
```

class wax_toolbox.misc_fixtures.**RecorderBase**(*bucket_name*, *sample_dir*, *session=<requests.sessions.Session object>*, *betamax_mode='none'*)

Context manager that will either record or replay requests call. It also proposes a send_object that will either save a DataFrame or compare with the saved DataFrame.

Classic use:

```
with record("name-of-my-bucket") as r:
    df = a_function_that_uses_requests(...)
    r.send_dataframe(df)
```

wax_toolbox.misc_fixtures.**sanitize_token**(*interaction*, *current_cassette*)

Betamax token sanitizer.

CHAPTER 5

Authors

Guillaume <jeusel.guillaume@gmail.com>

CHAPTER 6

CHANGES

6.1 v0.0.5

- Fixing tags
- Fixing tags
- Fix setup.cfg
- Small fix
- Move from src/pkg to pkg
- Update changelog

6.2 v0.0.2

- Update Timer, more simple
- Update changelog & add tag to trigger pypi
- Update travis
- update changelog
- Adding Timer in __init__
- Add betamax
- Add travis key & comment black
- Add betamax fixture
- Fix docs layout
- Adding authors & changelog
- Adding profiling & tsanalytics

- first commit thx to soft-cookiecutter
- First commit thx to cookiecutter

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

W

Index