hrv

Release 0.2.8

Jul 14, 2021

Contents

hrv is a simple Python module that brings the most widely used techniques to extract information about cardiac autonomic functions through RRi series and Heart Rate Variability (HRV) analyses without losing the **Power** and **Flexibility** of a native Python object.

In other words, the **hrv** module eases the manipulation, inspection, pre-processing, visualization, and analyses of HRV-related information. Additionally, it is written with idiomatic code and tries to implement the API of a built-in object, which might make it intuitive for Python users.

CHAPTER 1

Contents

1.1 First Steps

1.1.1 Installation

To install use pip:

```
pip install hrv
```

Or clone the repo:

```
git clone https://github.com/rhenanbartels/hrv.git
python setup.py install
```

1.1.2 Basic Usage

Create an RRi instance

Once you create an RRi object you will have the power of a native Python iterable object. This means, that you can loop through it using a **for loop**, get a just a part of the series using native **slicing** and much more. Let us try it:

```
from hrv.rri import RRi

rri_list = [800, 810, 815, 750, 753, 905]
rri = RRi(rri_list)

print(rri)
RRi array([800., 810., 815., 750., 753., 905.])
```

Slicing

```
print(rri[0])
800.0

print(type(rri[0]))
numpy.float64

print(rri[::2])
RRi array([800., 815., 753.])
```

Logical Indexing

```
from hrv.rri import RRi

rri = RRi([800, 810, 815, 750, 753, 905])
rri_ge = rri[rri >= 800]

rri_ge
RRi array([800., 810., 815., 905.])
```

Loop

```
for rri_value in rri:
    print(rri_value)

800.0
810.0
815.0
750.0
753.0
905.0
```

Note: When time information is not provided, time array will be created using the cumulative sum of successive RRi. After cumulative sum, the time array is subtracted from the value at *t[0]* to make it start from *0s*

RRi object and time information

```
from hrv.rri import RRi

rri_list = [800, 810, 815, 750, 753, 905]
rri = RRi(rri_list)

print(rri.time)
array([0.   , 0.81 , 1.625, 2.375, 3.128, 4.033]) # Cumsum of rri values minus t[0]

rri = RRi(rri_list, time=[0, 1, 2, 3, 4, 5])
print(rri.time)
[0. 1. 2. 3. 4. 5.]
```

Note: Some validations are made in the time list/array provided to the RRi class, for instance:

- RRi and time list/array must have the same length;
- Time list/array can not have negative values;
- Time list/array must be monotonic increasing.

Basic math operations

With RRi objects you can make math operatins just like a numpy array:

```
rri
RRi array([800., 810., 815., 750., 753., 905.])

rri * 10
RRi array([8000., 8100., 8150., 7500., 7530., 9050.])

rri + 200
RRi array([1000., 1010., 1015.,  950.,  953., 1105.])
```

Works with Numpy functions

```
import numpy as np

rri = RRi([800, 810, 815, 750, 753, 905])

sum_rri = np.sum(rri)
print(sum_rri)
4833.0

mean_rri = np.mean(rri)
print(mean_rri)
805.5

std_rri = np.std(rri)
print(std_rri)
51.44171459039833
```

1.2 Read RRi files

1.2.1 Read .txt files

Text files contains a single column with all RRi values. Example of RRi text file

```
800
810
815
750
```

```
from hrv.io import read_from_text

rri = read_from_text('path/to/file.txt')

print(rri)
RRi array([800., 810., 815., 750.])
```

1.2.2 Read Polar® (.hrm) files

The .hrm files contain the RRi acquired with Polar®

A complete guide for .hrm files can be found here

```
from hrv.io import read_from_hrm

rri = read_from_hrm('path/to/file.hrm')

print(rri)
RRi array([800., 810., 815., 750.])
```

1.2.3 Read .csv files

Example of csv file:

```
800,
810,
815,
750,
```

```
from hrv.io import read_from_csv

rri = read_from_csv('path/to/file.csv')

print(rri)
RRi array([800., 810., 815., 750.])
```

Note: When using **read_from_csv** you can also provide a column containing time information. Let's check it.

```
800,1
810,2
815,3
750,4
```

```
rri = read_from_csv('path/to/file.csv', time_col_index=1)

print(rri)
RRi array([800., 810., 815., 750.])

print(rri.time)
array([0., 1., 2., 3., 4.])
```

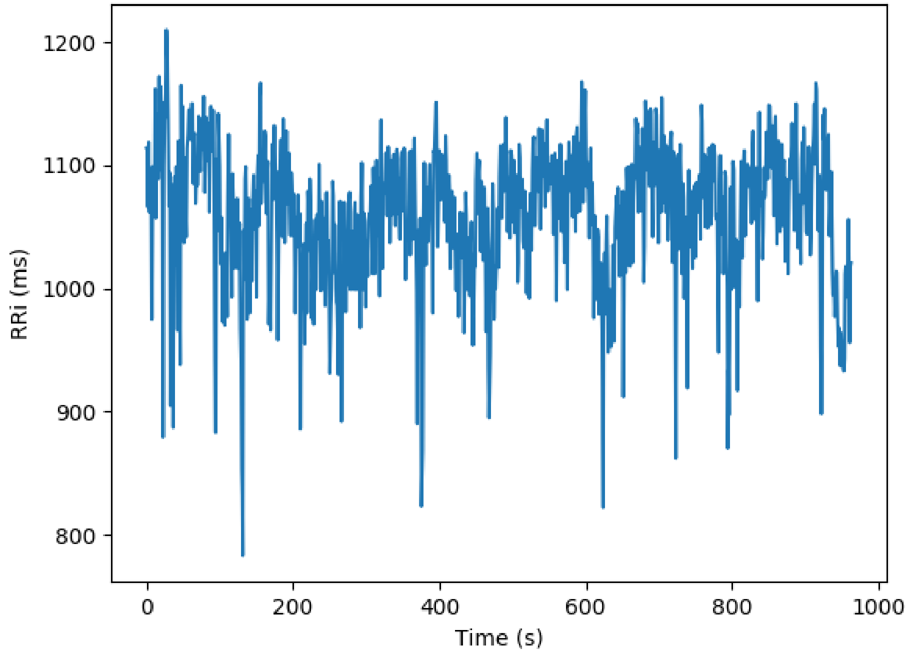
1.2.4 RRi Sample Data

The hrv module comes with some sample data. It contains:

- RRi collected during rest
- RRi collected during exercise
- RRi containing ectopic beats

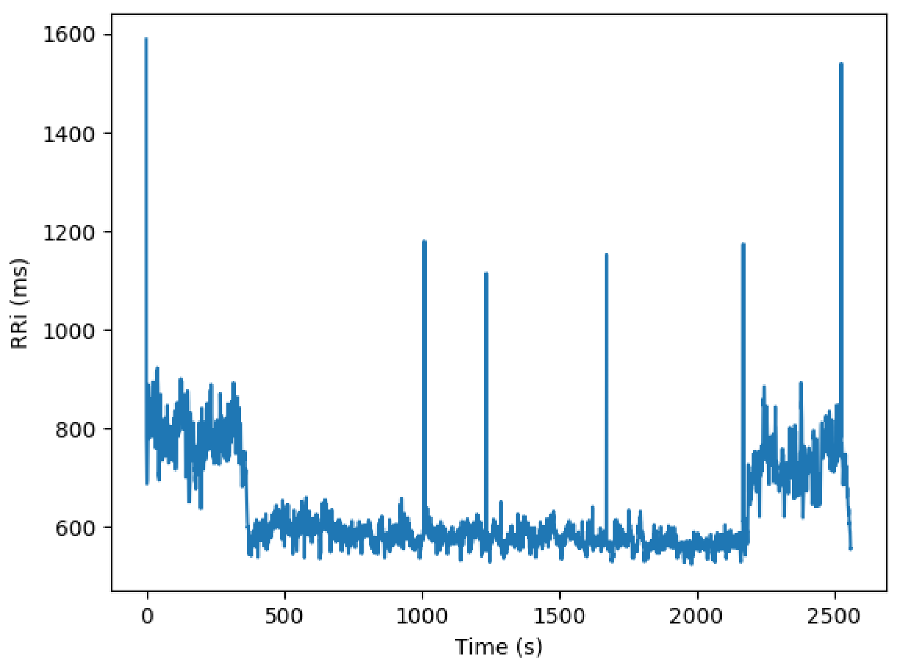
Rest RRi

```
from hrv.sampledata import load_rest_rri

rri = load_rest_rri()
rri.plot()
```



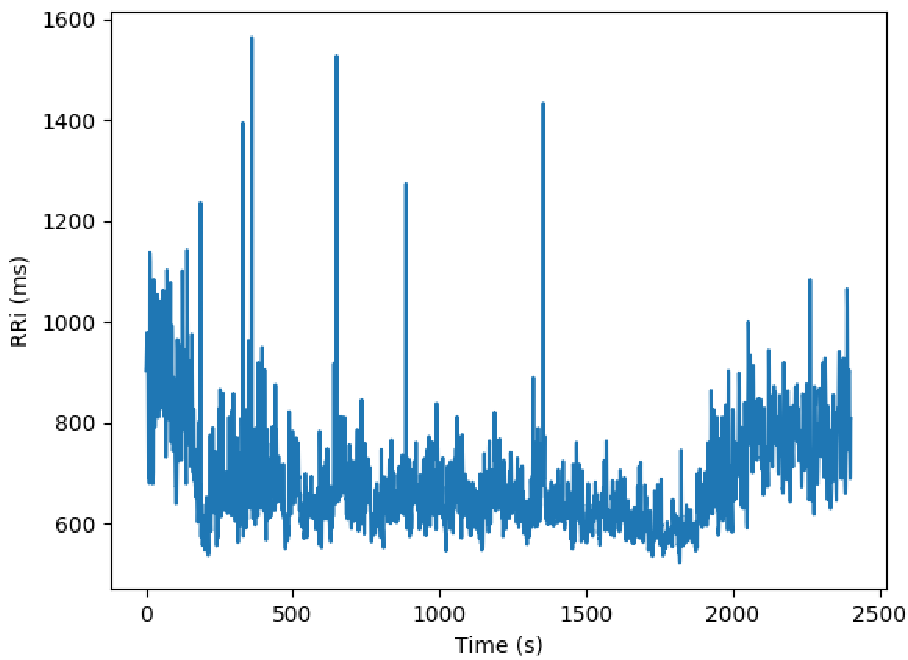
Exercise RRi

```
from hrv.sampledata import load_exercise_rri

rri = load_exercise_rri()
rri.plot()
```



Noisy RRi

```
from hrv.sampledata import load_noisy_rri

rri = load_noisy_rri()
rri.plot()
```



1.3 RRi statistics

1.3.1 Basic Statistics

The RRi object implements some basic statistics information about its values:

- mean
- median
- standard deviation
- variance
- minimum
- maximum
- amplitude

Some examples:

```
from hrv.rri import RRi
```

(continues on next page)

(continued from previous page)

```
rri = RRi([800, 810, 815, 750, 753, 905])

# mean
rri.mean()
805.5

# median
rri.median()
805.0
```

You can also have a complete overview of its statistical charactheristic

```
desc = rri.describe()
desc

----------------------------------------
                     rri           hr
----------------------------------------
min               750.00        66.30
max               905.00        80.00
mean              805.50        74.78
var              2646.25        20.85
std                51.44         4.57
median            805.00        74.54
amplitude         155.00        13.70

print(desc['std'])
{'rri': 51.44171459039833, 'hr': 4.5662272355549725}
```

1.3.2 RRi Basic Information

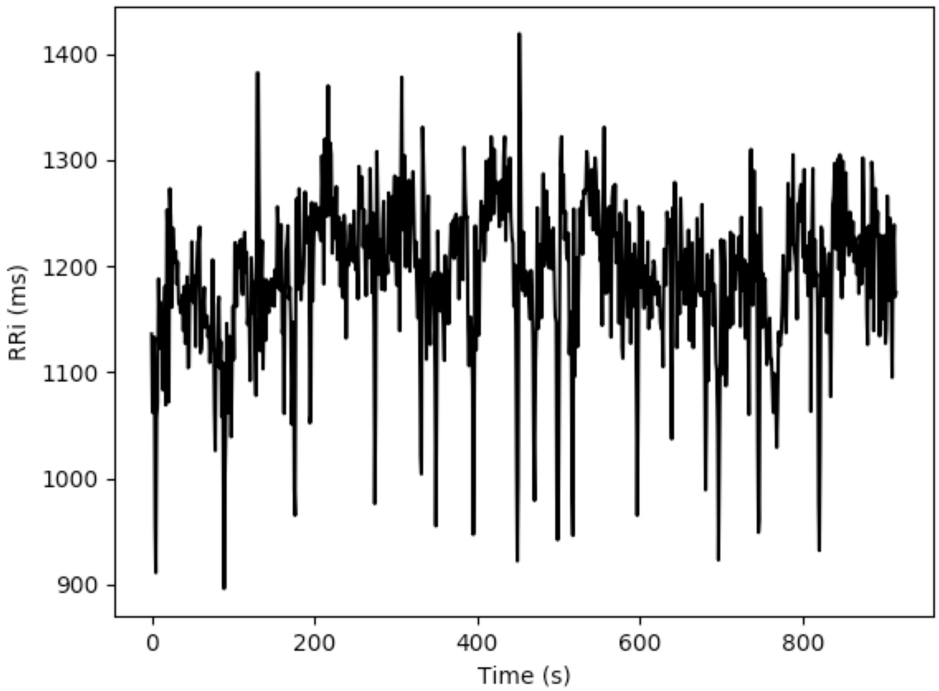
```
rri = RRi([800, 810, 815, 750, 753, 905])
rri.info()

N Points: 6
Duration: 4.03s
Interpolated: False
Detrended: False
Memory Usage: 0.05Kb
```

1.4 RRi Visualization

The RRi class brings a very easy way to visualize your series:

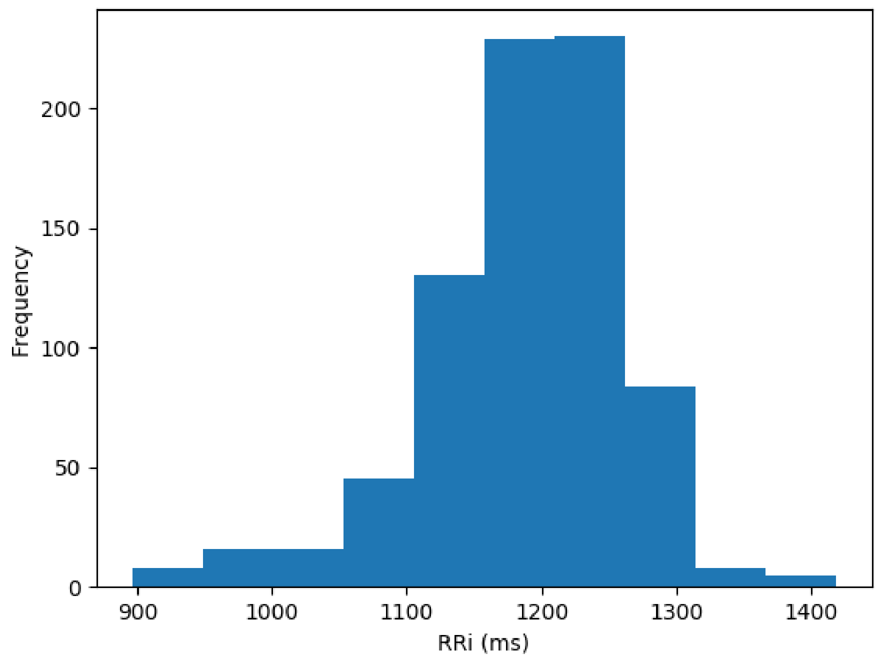
1.4.1 Plot RRi Series

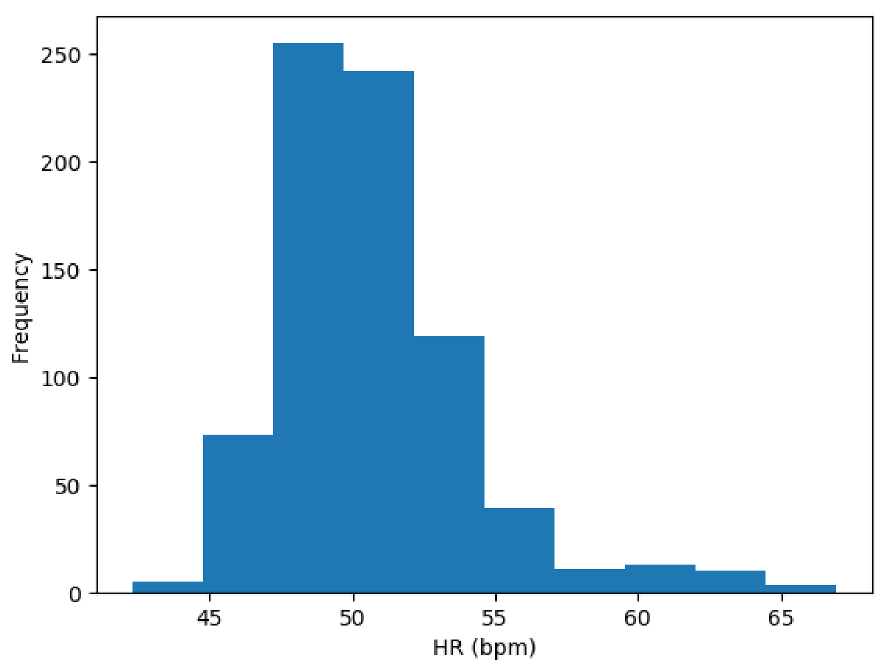
```
from hrv.io import read_from_text

rri = read_from_text('path/to/file.txt')
fig, ax = rri.plot(color='k')
```



1.4.2 RRi histogram and Heart Rate Histogram

```
rri.hist()

rri.hist(hr=True)
```



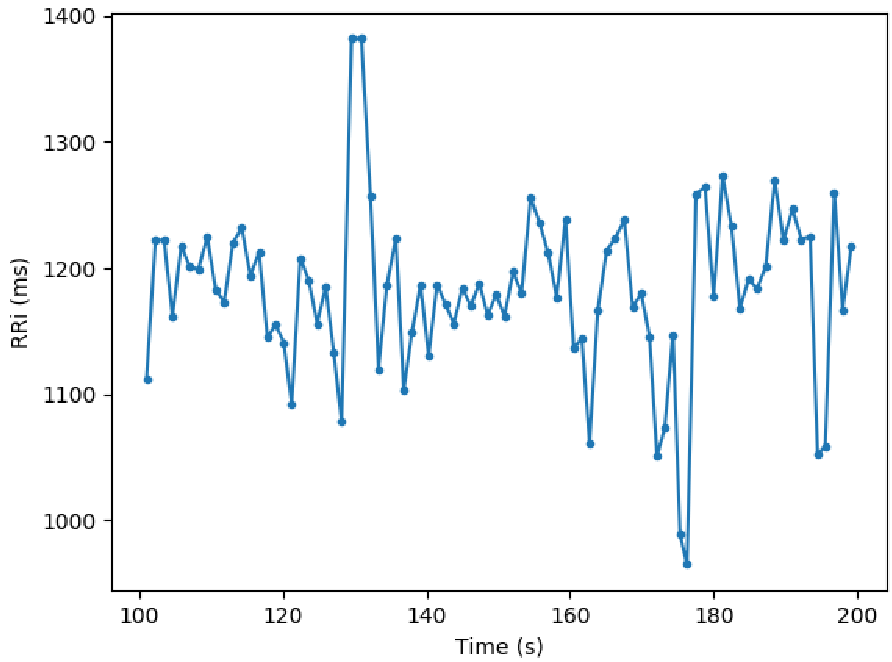


1.5 Time Slicing

It is also possible to slice RRi series with time range information (in **seconds**).

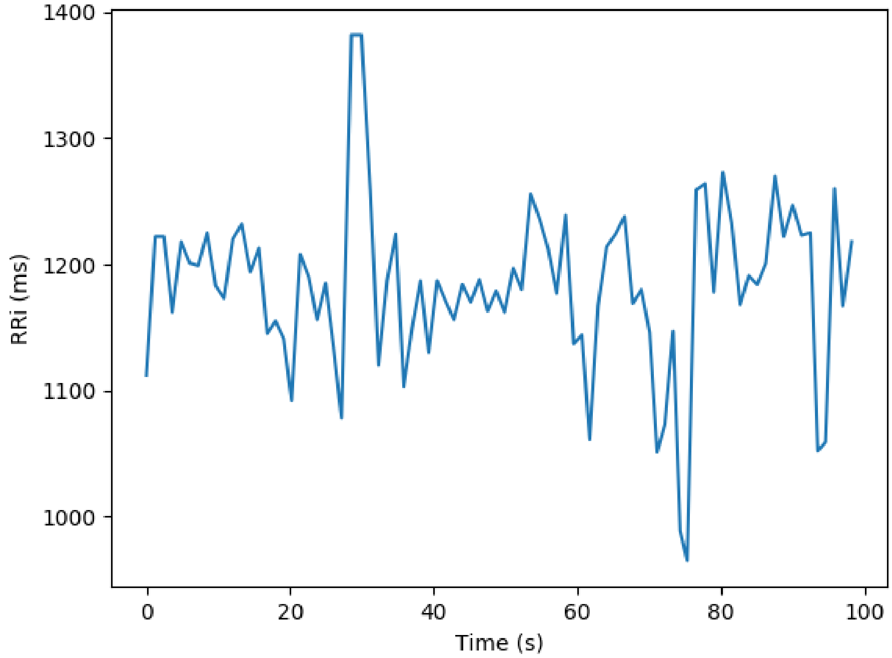
In the following example, we are taking a slice that starts at *100s ‘ and ends at ‘200s*.

```
from hrv.io import read_from_text

rri = read_from_text('path/to/file.txt')
rri_range = rri.time_range(start=100, end=200)

fig, ax = rri_range.plot(marker='.')
```



Time offset can be reset from the RRi series range:

```
rri_range.reset_time(inplace=True)
```

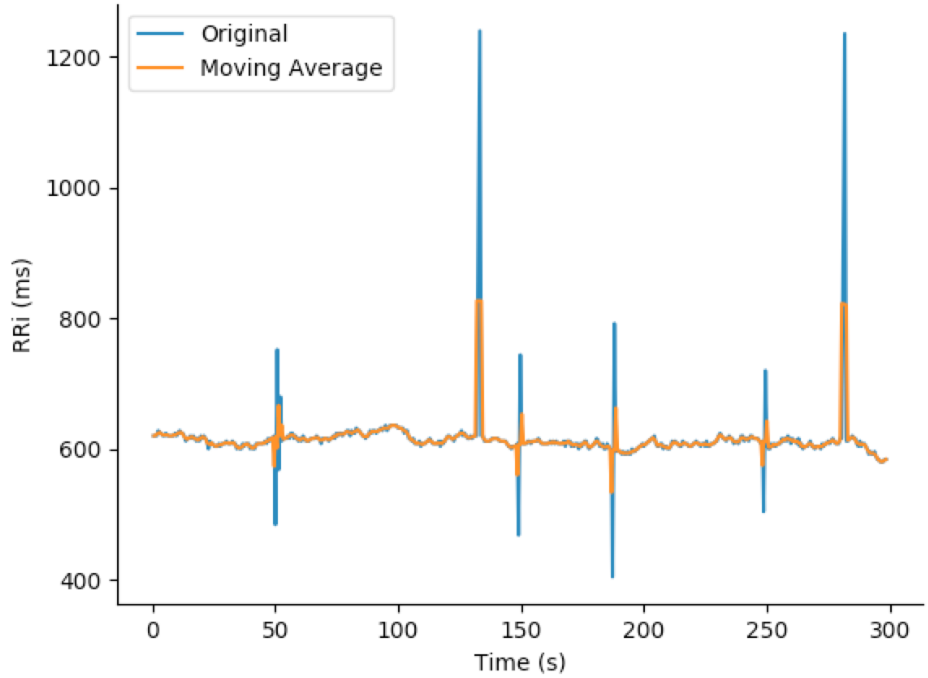


1.6 Pre-Processing

1.6.1 Filters

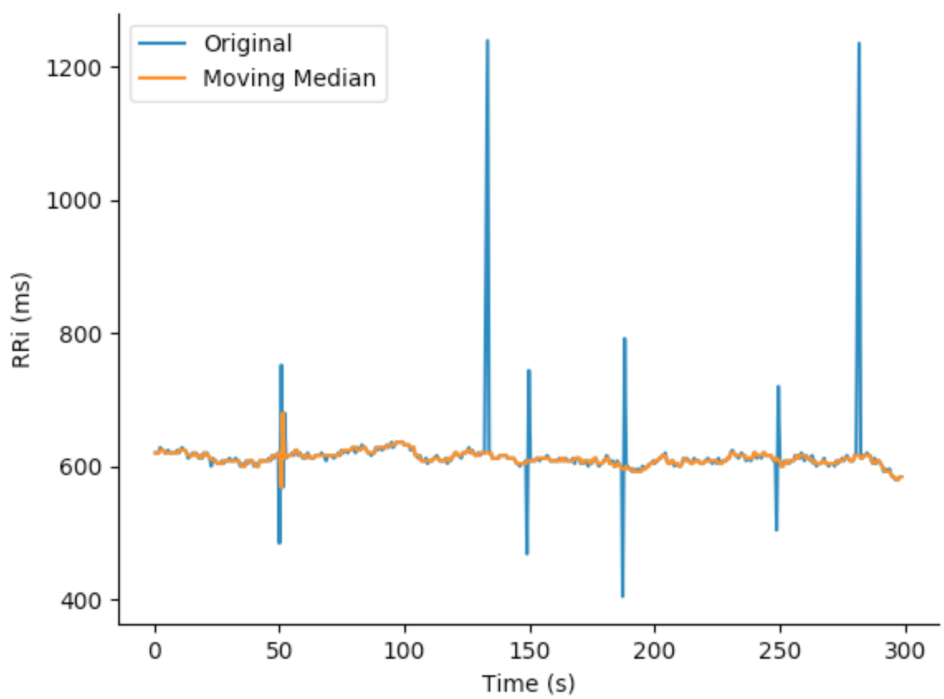
Moving Average

```
from hrv.filters import moving_average
filt_rri = moving_average(rri, order=3)

fig, ax = rri.plot()
filt_rri.plot(ax=ax)
```



Moving Median

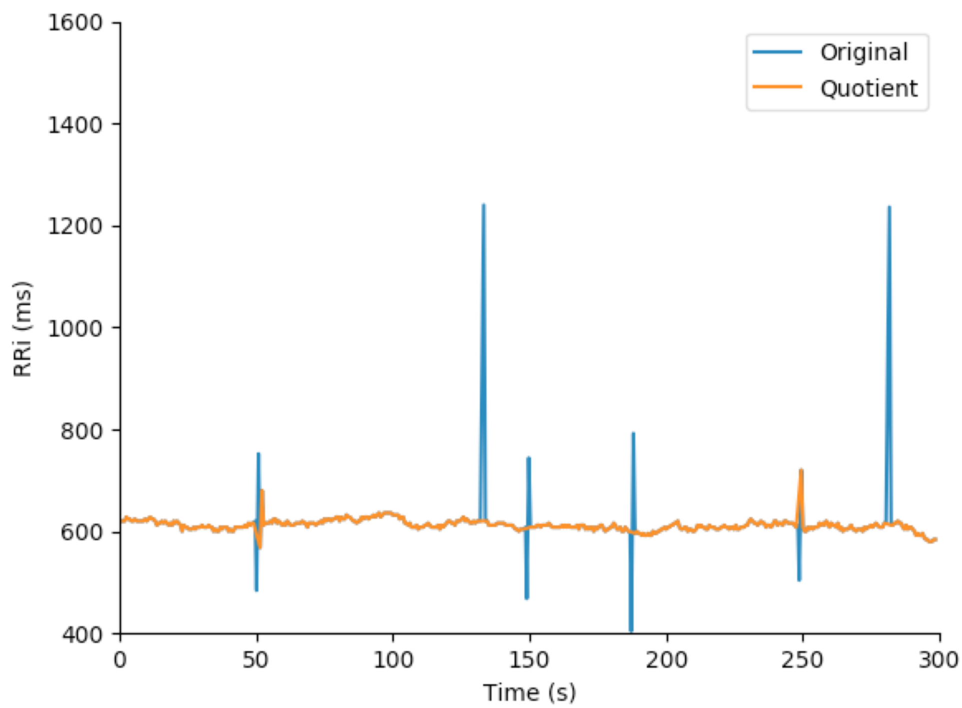
```
from hrv.filters import moving_median
filt_rri = moving_median(rri, order=3)

fig, ax = rri.plot()
filt_rri.plot(ax=ax)
```



Quotient

Read more

```
from hrv.filters import quotient
filt_rri = quotient(rri)

fig, ax = rri.plot()
filt_rri.plot(ax=ax)
```



Threshold Filter

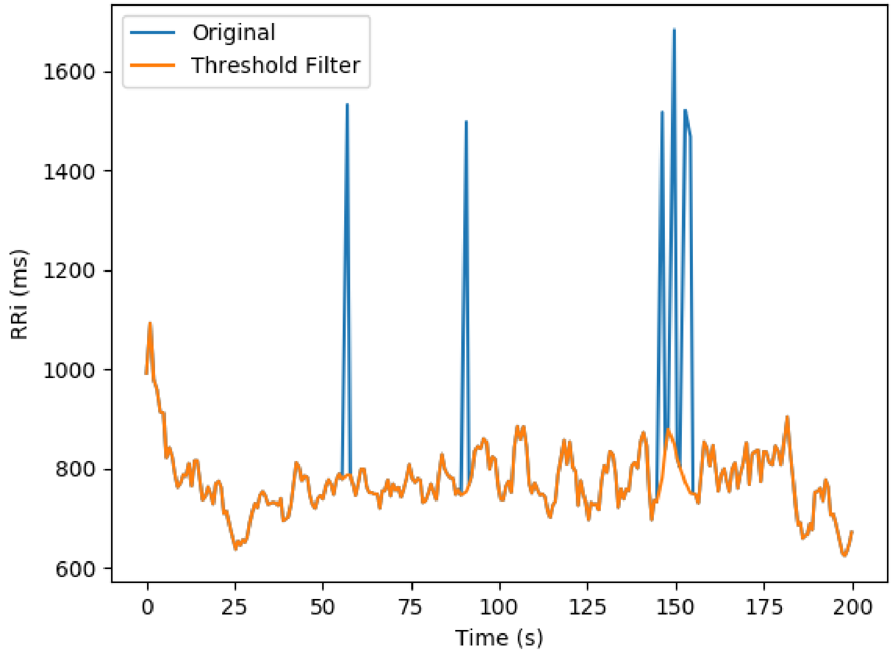
This filter is inspired by the threshold-based artifact correction algorithm offered by kubios ® . To elect outliers in the tachogram series, each RRi is compared to the median value of local RRi (default N=5). All the RRi which the difference is greater than the local median value plus a threshold is replaced by cubic interpolated RRi.

The threshold filter has five pre-defined strength values:

- Very Low: 450ms
- Low: 350ms
- Medium: 250ms
- Strong: 150ms
- Very Strong: 50ms

It also accepts custom threshold values (in milliseconds). The following snippet shows the ectopic RRi removal:

```
from hrv.filters import threshold_filter
filt_rri = threshold_filter(rri, threshold='medium', local_median_size=5)

fig, ax = rri.plot()
filt_rri.plot(ax=ax)
```



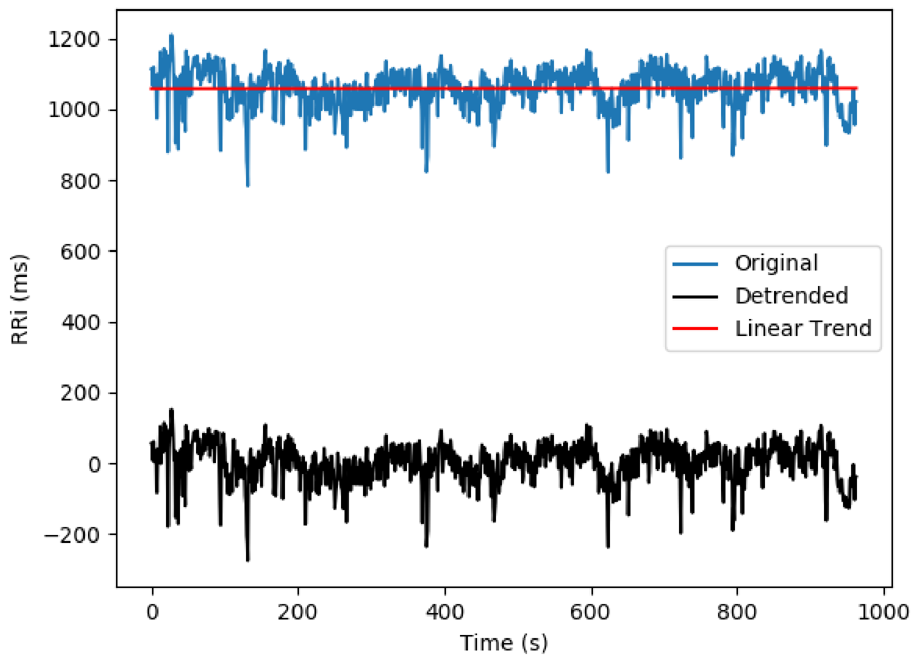
1.6.2 Detrending

The **hrv** module also offers functions to remove the non-stationary trends from the RRi series. It allows the removal of slow linear or more complex trends using the following approaches:

Polynomial models

Given a degree a polynomial filter is applied to the RRi series and subtracted from the tachogram

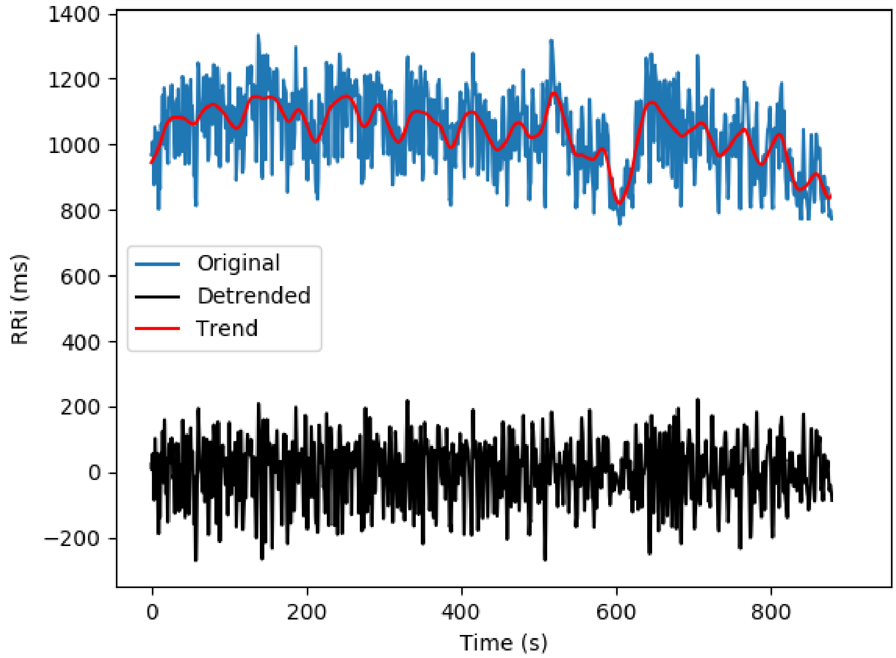
```
from hrv.detrend import polynomial_detrend

rri_detrended = polynomial_detrend(rri, degree=1)

fig, ax = rri.plot()
rri_detrended.plot(ax, color='k')
```



Smoothness priors

Developed by Tarvainen *et al*, allow the removal of complex trends. Visit here for more information. It worth noticing that the detrended RRi with the Smoothness priors approach is also interpolated and resampled using frequency equals to `` `fs` ``.

```
from hrv.detrend import smoothness_priors

rri_detrended = smoothness_priors(rri, l=500, fs=4.0)

fig, ax = rri.plot()
rri_detrended.plot(ax, color='k')
```

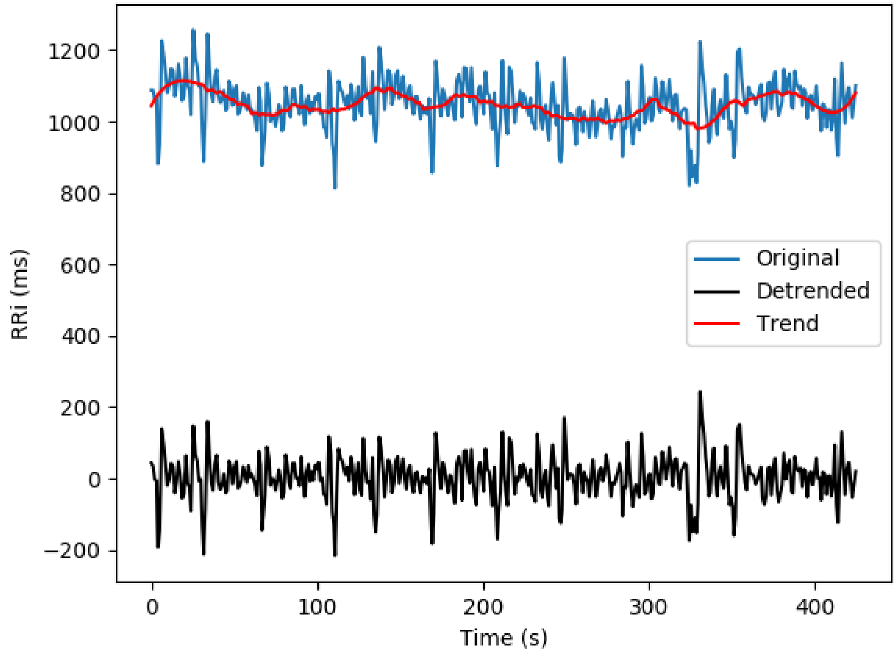


Note: this approach depends on a numpy matrix inversion and due to floating-point precision it might present round-off errors in the trend calculation

Savitzky-Golay

Uses the lowpass filter known as Savitzky-Golay filter to smooth the RRi series and remove slow components from the tachogram

```
from hrv.detrend import sg_detrend

rri_detrended = sg_detrend(rri, window_size=51, polyorder=3)

fig, ax = rri.plot()
rri_detrended.plot(ax, color='k')
```



1.7 Analysis

1.7.1 Time Domain Analysis

```
from hrv.classical import time_domain
from hrv.io import read_from_text

rri = read_from_text('path/to/file.txt')
results = time_domain(rri)
print(results)

{'mhr': 66.528130159638053,
 'mrri': 912.50302419354841,
 'nn50': 337,
 'pnn50': 33.971774193548384,
 'rmssd': 72.849900286450023,
 'sdnn': 96.990569261440797,
 'sdsd': 46.233829821038042}
```

1.7.2 Frequency Domain Analysis

```
from hrv.classical import frequency_domain
from hrv.io import read_from_text

rri = read_from_text('path/to/file.txt')
results = frequency_domain(
```

(continues on next page)

(continued from previous page)

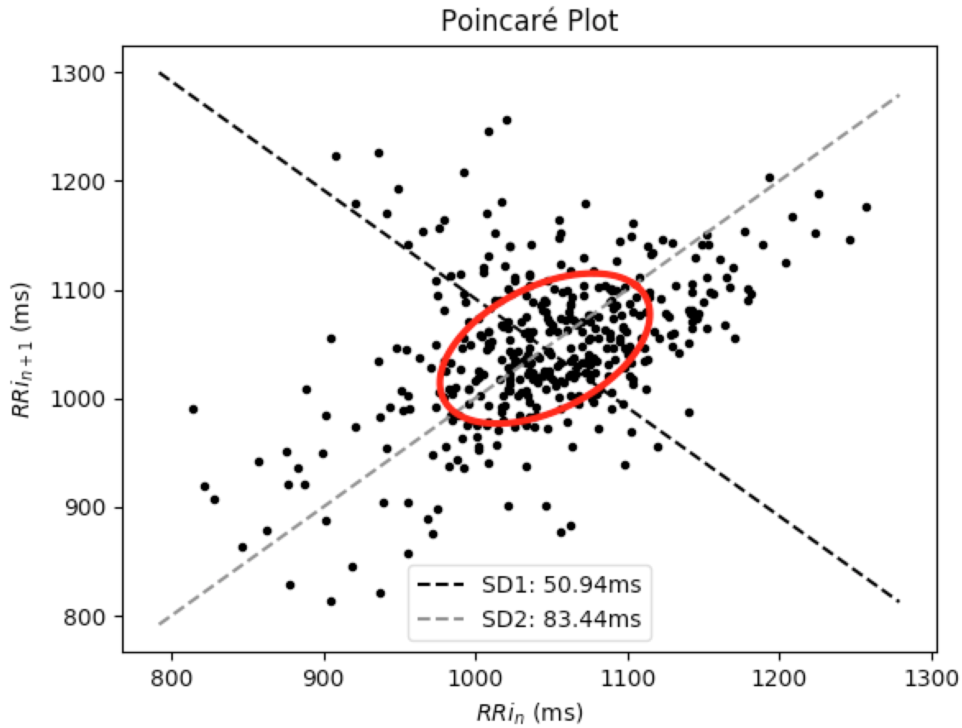
```
    rri=rri,
    fs=4.0,
    method='welch',
    interp_method='cubic',
    detrend='linear'
)
print(results)

{'hf': 1874.6342520920668,
 'hfnu': 27.692517001462079,
 'lf': 4894.8271587038234,
 'lf_hf': 2.6110838171452708,
 'lfnu': 72.307482998537921,
 'total_power': 7396.0879278950533,
 'vlf': 626.62651709916258}
```

1.7.3 Non-linear Analysis

```
from hrv.classical import non_linear
from hrv.io import read_from_text

rri = read_from_text('path/to/file.txt')
results = non_linear(rri)
print(results)

{'sd1': 51.538501037146382,
 'sd2': 127.11460955437322}
```

It is also possible to depict the Poincaré Plot, from which SD1 and SD2 are derived:

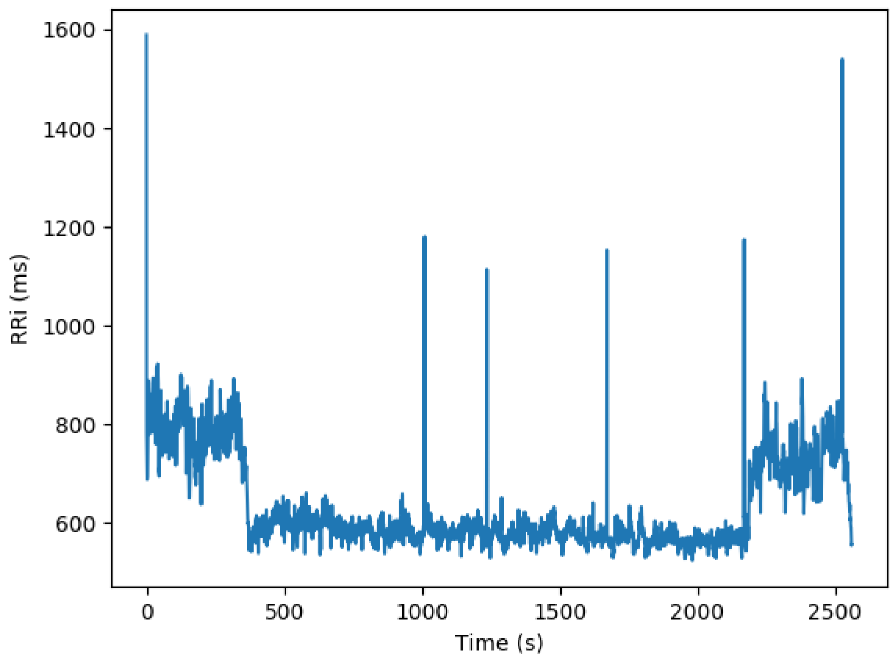
```
rri.poincare_plot()
```



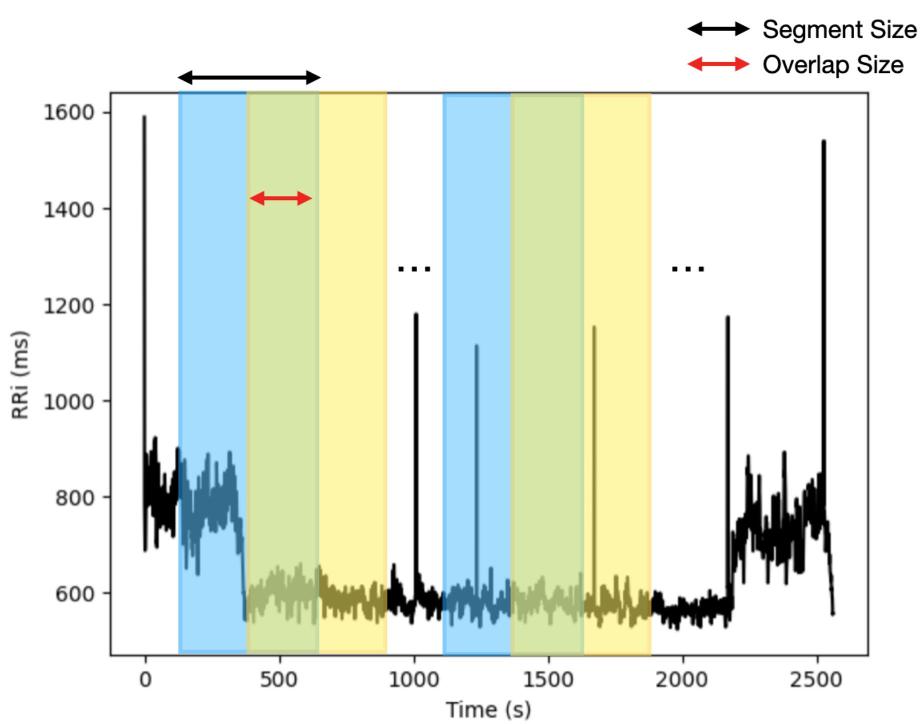
1.8 Non-stationary RRi series

In some situations like physical exercise, the RRi series might present a non-stationary behavior. In cases like these, classical approaches are not recommended once the statistical properties of the signal vary over time.

The following figure depicts the RRi series recorded on a subject riding a bicycle. Without running analysis and only visually inspecting the time series, is possible to tell that the average and the standard deviation of the RRi are not constant as a function of time.



In order to extract useful information about the dynamics of non-stationary RRi series, the following methods applies the classical metrics in shorter running adjacent segments, as illustrated in the following image:



For example, for a segment size of **30s** (S) and **15s** (O) overlap a signal with **300s** (D) will have P segments:

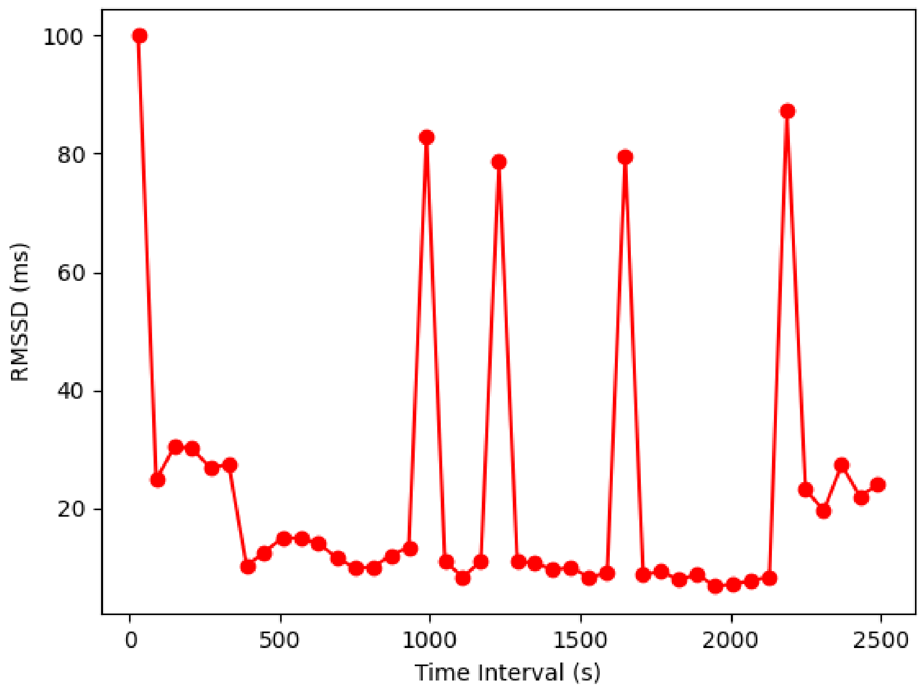
P = int((D - S) / (S - O)) + 1

P = int((300 - 30) / (30 - 15)) + 1 = 19 segments

1.8.1 Time Varying

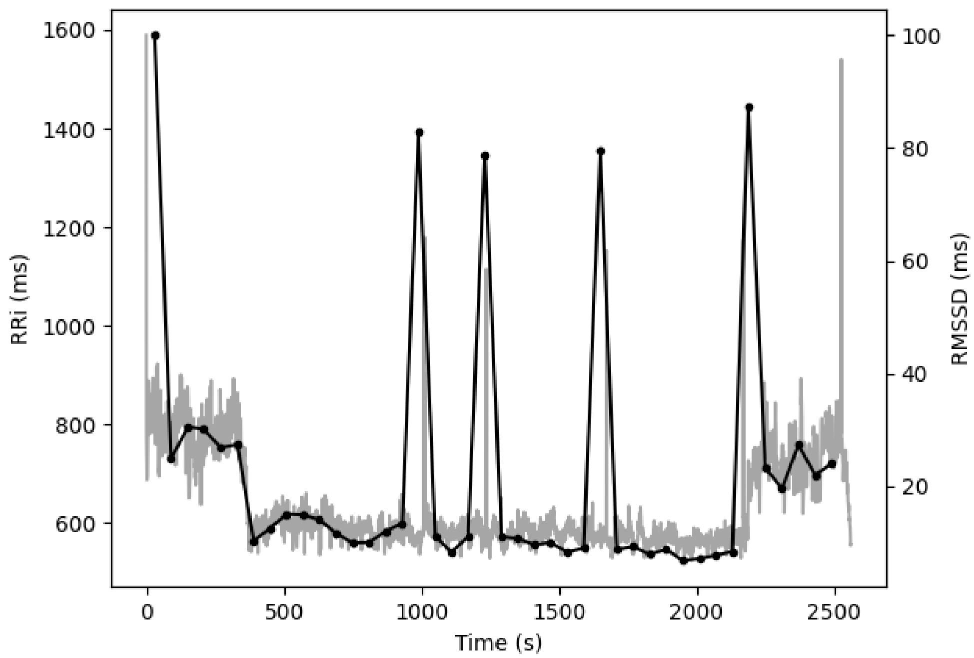
Time domain indices applied to shorter segments

```
from hrv.sampledata import load_exercise_rri
from hrv.nonstationary import time_varying

rri = load_exercise_rri()
results = time_varying(rri, seg_size=30, overlap=0)
results.plot(index="rmssd", marker="o", color="r")
```



Plot the results from **time varying** together with its respective RRi series

```
from hrv.sampledata import load_exercise_rri
from hrv.nonstationary import time_varying

rri = load_exercise_rri()
results = time_varying(rri, seg_size=30, overlap=0)
results.plot_together(index="rmssd", marker="o", color="k")
```



1.8.2 Short Time Fourier Transform

To be implemented.

1.9 Contribution start guide

The preferred way to start contributing for the project is creating a virtualenv (you can do by using virtualenv, virtualenvwrapper, pyenv or whatever tool you'd like). Only **Python 3.x** are supported

1.9.1 Preparing the eviroment

Create the virtualenv:

```
mkvirtualenv hrv
```

Install all dependencies:

```
pip install -r requirements.txt
```

Install development dependencies:

```
pip install -r dev-requirements.txt
```

1.9.2 Running the tests

In order to run the tests, activate the virtualenv and execute pytest:

```
workon <virtualenv>
pytest -v
# or
make test
```

1.9.3 Coding and Docstring styles

Generally, we try to use Python common styles conventions as described in PEP 8 and PEP 257, which are also followed by the numpy project.

Example

```
def moving_average(rri, order=3):
    """
    Low-pass filter. Replace each RRi value by the average of its N/2
    neighbors. The first and the last N/2 RRi values are not filtered

    Parameters
    ----------
    rri : array_like
    sequence containing the RRi series
    order : int, optional
    Strength of the filter. Number of adjacent RRi values used to calculate
    the average value to replace the current RRi. Defaults to 3.

    .. math::
    considering movinge average of order equal to 3:
    RRi[j] = sum(RRi[j-2] + RRi[j-1] + RRi[j+1] + RRi[j+2]) / 3

    Returns
    -------
    results : RRi array
    instance of the RRi class containing the filtered RRi values

    See Also
    -------
    moving_median, threshold_filter, quotient

    Examples
    --------
    >>> from hrv.filters import moving_average
    >>> from hrv.sampledata import load_noisy_rri
    >>> noisy_rri = load_noisy_rri()
    >>> moving_average(noisy_rri)
    RRi array([904., 918., 941.66666667, ..., 732.66666667, 772.33333, 808.])
    """
```

We also encourage the use of code linters, such isort , black and autoflake.

```
autoflake --in-place --recursive --remove-unused-variables --remove-all-unused-
↪imports .
sort -rc .
black .
```