

---

# **pycast Documentation**

***Release 0.0.7-prealpha***

**Christian Schwarz**

November 18, 2012



# CONTENTS

<b>1</b>	<b>pycast.common</b>	<b>1</b>
<b>2</b>	<b>TimeSeries</b>	<b>3</b>
<b>3</b>	<b>Smoothing Methods</b>	<b>9</b>
<b>4</b>	<b>Forecasting Methods</b>	<b>11</b>
<b>5</b>	<b>Error Measures</b>	<b>13</b>
<b>6</b>	<b>Optimization Methods</b>	<b>15</b>
<b>7</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



# PYCAST.COMMON

`pycast.common.profileme.profileMe`  
alias of `_ProfileDecorator`

`pycast.common.helper.linear_interpolation` (*first, last, steps*)  
Interpolates all missing values using linear interpolation.

**Parameters**

- **first** (*Numeric*) – Start value for the interpolation.
- **last** (*Numeric*) – End Value for the interpolation
- **steps** (*Integer*) – Number of missing values that have to be calculated.

**Returns** Returns a list of floats containing only the missing values.

**Return type** List



# TIMESERIES

**class** `pycast.common.timeseries.TimeSeries` (*isNormalized=False, isSorted=False*)

A `TimeSeries` instance stores all relevant data for a real world time series.

**Warning** `TimeSeries` instances are NOT threadsafe.

**\_\_add\_\_** (*otherTimeSeries*)

Creates a new `TimeSeries` instance containing the data of `self` and `otherTimeSeries`.

**Parameters** `otherTimeSeries` (*TimeSeries*) – `TimeSeries` instance that will be merged with `self`.

**Returns** Returns a new `TimeSeries` instance containing the data entries of `self` and `otherTimeSeries`.

**Return type** `TimeSeries`

**\_\_eq\_\_** (*otherTimeSeries*)

Returns if `self` and the other `TimeSeries` are equal.

**TimeSeries are equal to each other if:**

- they contain the same number of entries
- each data entry in one `TimeSeries` is also member of the other one.

**Parameters** `otherTimeSeries` (*TimeSeries*) – `TimeSeries` instance that is compared with `self`.

**Returns** `True` if the `TimeSeries` objects are equal, `False` otherwise.

**Return type** `Boolean`

**\_\_getitem\_\_** (*index*)

Returns the item stored at the `TimeSeries` `index`-th position.

**Parameters** `index` (*Integer*) – Position of the element that should be returned. Starts at 0

**Returns** Returns a list containing [timestamp, data] lists.

**Return type** `List`

**Raise** Raises an `IndexError` if the `index` is out of range.

**\_\_init\_\_** (*isNormalized=False, isSorted=False*)

Initializes the `TimeSeries`.

**Parameters**

- **isNormalized** (*Boolean*) – Within a normalized TimeSeries, all data points have the same temporal distance to each other. When this is `True`, the memory consumption of the TimeSeries might be reduced. Also algorithms will probably run faster on normalized TimeSeries. This should only be set to `True`, if the TimeSeries is really normalized! TimeSeries normalization can be forced by executing `TimeSeries.normalize()`.
- **isSorted** (*Boolean*) – If all data points added to the time series are added in their ascending temporal order, this should set to `True`.

`__iter__()`

Returns an iterator that can be used to iterate over the data stored within the TimeSeries.

**Returns** Returns an iterator for the TimeSeries.

**Return type** Iterator

`__len__()`

Returns the number of data entries stored in the TimeSeries.

**Returns** Returns an Integer representing the number on data entries stored within the TimeSeries.

**Return type** Integer

`__setitem__(index, value)`

Sets the item at the index-th position of the TimeSeries.

**Parameters**

- **index** (*Integer*) – Index of the element that should be set.
- **value** (*List*) – A list of the form [timestamp, data]

**Raise** Raises an `IndexError` if the index is out of range.

`__str__()`

Returns a string representation of the TimeSeries.

**Returns**

Returns a string representing the TimeSeries in the format:

“TimeSeries([timestamp, data], [timestamp, data], [timestamp, data])”.

**Return type** String

`__weakref__`

list of weak references to the object (if defined)

**add\_entry** (*timestamp, data, format=None*)

Adds a new data entry to the TimeSeries.

**Parameters**

- **timestamp** – Time stamp of the data. This has either to be a float representing the UNIX epochs or a string containing a timestamp in the given format.
- **data** (*Numeric*) – Actual data value.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if set. For valid examples take a look into the `time.strptime()` documentation.

**apply** (*method*)

Applies the given ForecastingAlgorithm or SmoothingMethod from the `pystac.methods` module to the TimeSeries.



**Parameters method** (*BaseMethod*) – Method that should be used with the TimeSeries. For more information about the methods take a look into their corresponding documentation.

**classmethod** `convert_epoch_to_timestamp` (*timestamp, format*)

Converts the given float representing UNIX-epochs into an actual timestamp.

**Parameters**

- **timestamp** (*Float*) – Timestamp as UNIX-epochs.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp from UNIX epochs. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns the timestamp as defined in format.

**Return type** String

**classmethod** `convert_timestamp_to_epoch` (*timestamp, format*)

Converts the given timestamp into a float representing UNIX-epochs.

**Parameters**

- **timestamp** (*String*) – Timestamp in the defined format.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns an float, representing the UNIX-epochs for the given timestamp.

**Return type** Float

**classmethod** `from_json` (*json, format=None*)

Creates a new TimeSeries instance from the given json string.

**Parameters**

- **json** (*String*) – JSON string, containing the time series data. This should be a string created by `TimeSeries.to_json()`.
- **format** (*String*) – Format of the given timestamps. This is used to convert the timestamps into UNIX epochs, if set. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns a TimeSeries instance containing the data.

**Return type** TimeSeries

**Warning** This is probably an unsafe version! Only use it with JSON strings created by `TimeSeries.to_json()`. All assumptions regarding normalization and sort order will be ignored and set to default.

**classmethod** `from_twodim_list` (*datalist, format=None*)

Creates a new TimeSeries instance from the data stored inside a two dimensional list.

**Parameters**

- **datalist** (*List*) – List containing multiple iterables with at least two values. The first item will always be used as timestamp in the predefined format, the second represents the value. All other items in those sublists will be ignored.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns a TimeSeries instance containing the data from datalist.

**Return type** TimeSeries

**initialize\_from\_sql\_cursor** (*sqlcursor*, *format=None*)

Initializes the TimeSeries's data from the given SQL cursor.

**Parameters**

- **sqlcursor** (*SQLCursor*) – Cursor that was holds the SQL result for any given “SELECT timestamp, value, ... FROM ...” SQL query. Only the first two attributes of the SQL result will be used.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if set. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns the number of entries added to the TimeSeries.

**Return type** Integer

**is\_normalized** ()

Returns if the TimeSeries is normalized.

**Returns** Returns `True` if the TimeSeries is normalized, `False` otherwise.

**Return type** Boolean

**is\_sorted** ()

Returns if the TimeSeries is sorted.

**Returns** Returns `True` if the TimeSeries is sorted ascending, `False` in all other cases.

**Return type** Boolean

**normalize** (*normalizationLevel='minute'*, *fusionMethod='average'*, *interpolationMethod='linear'*)

Normalizes the TimeSeries data points.

If this function is called, the TimeSeries gets ordered ascending automatically. The new timestamps will represent the center of each time bucket. Within a normalized TimeSeries, the temporal distance between two consecutive data points is constant.

**Parameters**

- **normalizationLevel** (*String*) – Level of normalization that has to be applied. The available normalization levels are defined in `timeseries.NormalizationLevels`.
- **fusionMethod** (*String*) – Normalization method that has to be used if multiple data entries exist within the same normalization bucket. The available methods are defined in `timeseries.FusionMethods`.
- **interpolationMethod** (*String*) – Interpolation method that is used if a data entry at a specific time is missing. The available interpolation methods are defined in `timeseries.InterpolationMethods`.

**Raise** Raises a `ValueError` if a `normalizationLevel`, `fusionMethod` or `interpolationMethod` have an unknown value.

**sort\_timeseries** (*ascending=True*)

Sorts the data points within the TimeSeries according to their occurrence inline.

**Parameters** **ascending** (*Boolean*) – Determines if the TimeSeries will be ordered ascending or decending. If this is set to decending once, the ordered parameter defined in `TimeSeries.__init__()` will be set to `False` FOREVER.

**Returns** Returns `self` for convenience.

**Return type** `TimeSeries`

**`sorted_timeseries`** (*ascending=True*)

Returns a sorted copy of the `TimeSeries`, preserving the original one.

As an assumption this new `TimeSeries` is not ordered anymore if a new value is added.

**Parameters** **ascending** (*Boolean*) – Determines if the `TimeSeries` will be ordered ascending or descending.

**Returns** Returns a new `TimeSeries` instance sorted in the requested order.

**Return type** `TimeSeries`

**`to_gnuplot_datafile`** (*datafilepath, format=None*)

Dumps the `TimeSeries` into a gnuplot compatible data file.

**Parameters**

- **datafilepath** (*String*) – Path used to create the file. If that file already exists, it will be overwritten!
- **format** (*String*) – Format of the timestamp. This is used to convert the timestamp from UNIX epochs, if set. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns `True` if the data could be written, `False` otherwise.

**Return type** `Boolean`

**`to_json`** (*format=None*)

Returns a JSON representation of the `TimeSeries` data.

**Parameters** **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if set. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns a basestring, containing the JSON representation of the current data stored within the `TimeSeries`.

**Return type** `String`

**`to_twodim_list`** (*format=None*)

Serializes the `TimeSeries` data into a two dimensional list of [timestamp, value] pairs.

**Parameters** **format** (*String*) – Format of the timestamp. This is used to convert the timestamp from UNIX epochs, if set. For valid examples take a look into the `time.strptime()` documentation.

**Returns** Returns a two dimensional list containing [timestamp, value] pairs.

**Return type** `List`



## SMOOTHING METHODS

**class** `pycast.methods.SimpleMovingAverage` (*windowsize=5*)

Implements the simple moving average.

The SMA algorithm will calculate the average value at time *t* based on the datapoints between [*t* - floor(*windowsize* / 2), *t* + floor(*windowsize* / 2)].

**Explanation:** [http://en.wikipedia.org/wiki/Moving\\_average](http://en.wikipedia.org/wiki/Moving_average)

**\_\_init\_\_** (*windowsize=5*)

Initializes the SimpleMovingAverage.

**Parameters** **windowsize** (*Integer*) – Size of the SimpleMovingAverages window.

**Raise** Raises a `ValueError` if *windowsize* is an even or not larger than zero.

**execute** (*timeSeries*)

Creates a new `TimeSeries` containing the SMA values for the predefined *windowsize*.

**Parameters** **timeSeries** (*TimeSeries*) – The `TimeSeries` used to calculate the simple moving average values.

**Returns** `TimeSeries` object containing the smooth moving average.

**Return type** `TimeSeries`

**Note** This implementation aims to support independent for loop execution.



# FORECASTING METHODS

**class** `pycast.methods.ExponentialSmoothing` (*smoothingFactor=0.1, valuesToForecast=1*)  
Implements an exponential smoothing algorithm.

**Explanation:** <http://www.youtube.com/watch?v=J4iODLa9hYw>

**\_\_init\_\_** (*smoothingFactor=0.1, valuesToForecast=1*)  
Initializes the ExponentialSmoothing.

## Parameters

- **smoothingFactor** (*Float*) – Defines the alpha for the ExponentialSmoothing. Valid values are in (0.0, 1.0).
- **valuesToForecast** (*Integer*) – Number of values that should be forecasted.

**Raise** Raises a `ValueError` when `smoothingFactor` has an invalid value.

**execute** (*timeSeries*)  
Creates a new `TimeSeries` containing the smoothed and forecasted values.

**Returns** `TimeSeries` object containing the smoothed `TimeSeries`, including the forecasted values.

**Return type** `TimeSeries`

**Note** The first normalized value is chosen as the starting point.

**class** `pycast.methods.HoltMethod` (*smoothingFactor=0.1, trendSmoothingFactor=0.5, valuesToForecast=1*)  
Implements the Holt algorithm.

**Explanation:** [http://en.wikipedia.org/wiki/Exponential\\_smoothing#Double\\_exponential\\_smoothing](http://en.wikipedia.org/wiki/Exponential_smoothing#Double_exponential_smoothing)

**\_\_init\_\_** (*smoothingFactor=0.1, trendSmoothingFactor=0.5, valuesToForecast=1*)  
Initializes the HoltMethod.

## Parameters

- **smoothingFactor** (*Float*) – Defines the alpha for the ExponentialSmoothing. Valid values are in (0.0, 1.0).
- **trendSmoothingFactor** (*Float*) – Defines the beta for the HoltMethod. Valid values are in (0.0, 1.0).
- **valuesToForecast** (*Integer*) – Defines the number of forecasted values that will be part of the result.

**Raise** Raises a `ValueError` when `smoothingFactor` or `trendSmoothingFactor` has an invalid value.

**execute** (*timeSeries*)

Creates a new TimeSeries containing the smoothed values.

**Returns** TimeSeries object containing the smoothed TimeSeries, including the forecasted values.

**Return type** TimeSeries

**Note** The first normalized value is chosen as the starting point.



# ERROR MEASURES

**class** `pycast.errors.BaseErrorMeasure` (*minimalErrorCalculationPercentage=60*)

Baseclass for all error measures.

**\_\_init\_\_** (*minimalErrorCalculationPercentage=60*)

Initializes the error measure.

**Parameters** **minimalErrorCalculationPercentage** (*Integer*) – The number of entries in an original TimeSeries that have to have corresponding partners in the calculated TimeSeries. Corresponding partners have the same time stamp. Valid values are in [0.0, 100.0].

**Raise** Raises a `ValueError` if `minimalErrorCalculationPercentage` is not in [0.0, 100.0].

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**get\_error** (*startingPercentage=0.0, endPercentage=100.0*)

Calculates the error for the given interval (`startingPercentage`, `endPercentage`) between the TimeSeries given during `BaseErrorMeasure.initialize()`.

**Parameters**

- **startingPercentage** (*Float*) – Defines the start of the interval. This has to be a value in [0.0, 100.0]. It represents the value, where the error calculation should be started. 25.0 for example means that the first 25% of all calculated errors will be ignored.
- **endPercentage** (*Float*) – Defines the end of the interval. This has to be a value in [0.0, 100.0]. It represents the value, after which all error values will be ignored. 90.0 for example means that the last 10% of all local errors will be ignored.

**Returns** Returns a float representing the error.

**Return type** `Float`

**Raise** Raises a `ValueError` in one of the following cases:

- `startingPercentage` not in [0.0, 100.0]
- `endPercentage` not in [0.0, 100.0]
- `endPercentage` < `startingPercentage`

**Raise** Raises a `StandardError` if `BaseErrorMeasure.initialize()` was not successful before.

**initialize** (*originalTimeSeries, calculatedTimeSeries*)

Initializes the ErrorMeasure.

During initialization, all `BaseErrorMeasure.local_errors()` are calculated.

### Parameters

- **originalTimeSeries** (*TimeSeries*) – TimeSeries containing the original data.
- **calculatedTimeSeries** (*TimeSeries*) – TimeSeries containing calculated data. Calculated data is smoothed or forecasted data.

**Returns** Return `True` if the error could be calculated, `False` otherwise based on the `minimalErrorCalculationPercentage`.

**Return type** Boolean

**local\_error** (*originalValue*, *calculatedValue*)

Calculates the error between the two given values.

### Parameters

- **originalValue** (*Numeric*) – Value of the original data.
- **calculatedValue** (*Numeric*) – Value of the calculated TimeSeries that corresponds to `originalValue`.

**Returns** Returns the error measure of the two given values.

**Return type** Numeric

**Raise** Raises a `NotImplementedError` if the child class does not overwrite this method.

**class** `pystac.errors.MeanSquaredError` (*minimalErrorCalculationPercentage=60*)

Implements the mean squared error measure.

**Explanation:** [http://en.wikipedia.org/wiki/Mean\\_squared\\_error](http://en.wikipedia.org/wiki/Mean_squared_error)

**local\_error** (*originalValue*, *calculatedValue*)

Calculates the error between the two given values.

### Parameters

- **originalValue** (*Numeric*) – Value of the original data.
- **calculatedValue** (*Numeric*) – Value of the calculated TimeSeries that corresponds to `originalValue`.

**Returns** Returns the error measure of the two given values.

**Return type** Numeric

**class** `pystac.errors.SymmetricMeanAbsolutePercentageError` (*minimalErrorCalculationPercentage=60*)

Implements the symmetric mean absolute percentage error with a boarder of 200%.

**Explanation:** <http://monashforecasting.com/index.php?title=SMAPE> (Formula (3))

If the calculated value and the original value are equal, the error is 0.

**local\_error** (*originalValue*, *calculatedValue*)

Calculates the error between the two given values.

### Parameters

- **originalValue** (*Numeric*) – Value of the original data.
- **calculatedValue** (*Numeric*) – Value of the calculated TimeSeries that corresponds to `originalValue`.

**Returns** Returns the error measure of the two given values.

**Return type** Numeric

# OPTIMIZATION METHODS

**class** `pycast.optimization.BaseOptimizationMethod` (*errorMeasureClass*, *precision=-1*)  
 Baseclass for all optimization methods.

**\_\_init\_\_** (*errorMeasureClass*, *precision=-1*)  
 Initializes the optimization method.

## Parameters

- **errorMeasureClass** (*BaseErrorMeasure*) – Error measure class from `pycast.errors`
- **precision** (*Integer*) – Defines the accuracy for parameter tuning in  $10^{\text{precision}}$ . This parameter has to be an integer in  $[-10, 0]$ .

**Raise** Raises a `TypeError` if *errorMeasureClass* is not a valid class. Valid classes are derived from `pycast.errors.BaseErrorMeasure`.

**Raise** Raises a `ValueError` if *precision* is not in  $[-10, 0]$ .

**\_\_weakref\_\_**  
 list of weak references to the object (if defined)

**optimize** (*timeSeries*, *forecastingMethods=[]*)  
 Runs the optimization on the given `TimeSeries`.

## Parameters

- **timeSeries** (*TimeSeries*) – `TimeSeries` instance that requires an optimized forecast.
- **forecastingMethods** (*List*) – List of `forecastingMethods` that will be used for optimization.

**Returns** Returns the optimized forecasting method with the smallest error.

**Return type** (`BaseForecastingMethod`, `Dictionary`)

**Raise** Raises a `ValueError` if no *forecastingMethods* is empty.

**class** `pycast.optimization.GridSearch` (*errorMeasureClass*, *precision=-1*)  
 Implements the grid search method for parameter optimization.

`GridSearch` is the brute force method.

**optimize** (*timeSeries*, *forecastingMethods=[]*)  
 Runs the optimization of the given `TimeSeries`.

## Parameters

- **timeSeries** (*TimeSeries*) – `TimeSeries` instance that requires an optimized forecast.
- **forecastingMethods** (*List*) – List of `forecastingMethods` that will be used for optimization.

**Returns** Returns the optimized forecasting method with the smallest error.

**Return type** BaseForecastingMethod, Dictionary

**Raise** Raises a `ValueError` if no forecastingMethods is empty.

**optimize\_forecasting\_method** (*timeSeries*, *forecastingMethod*)

Optimizes the parameters for the given timeSeries and forecastingMethod.

**Parameters**

- **timeSeries** (*TimeSeries*) – TimeSeries instance, containing hte original data.
- **forecastingMethod** (*BaseForecastingMethod*) – ForecastingMethod that is used to optimize the parameters.

**Todo** Errorclass for calculation

**Todo** percentage for start\_error\_measure, end\_error\_measure

**Todo** Definition of the result that will be returned.

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*



# PYTHON MODULE INDEX

## p

`pycast.common.helper`, 1  
`pycast.common.profileme`, 1  
`pycast.common.timeseries`, 1





# INDEX

## Symbols

`__add__()` (pycast.common.timeseries.TimeSeries method), 3  
`__eq__()` (pycast.common.timeseries.TimeSeries method), 3  
`__getitem__()` (pycast.common.timeseries.TimeSeries method), 3  
`__init__()` (pycast.common.timeseries.TimeSeries method), 3  
`__init__()` (pycast.errors.BaseErrorMeasure method), 13  
`__init__()` (pycast.methods.ExponentialSmoothing method), 11  
`__init__()` (pycast.methods.HoltMethod method), 11  
`__init__()` (pycast.methods.SimpleMovingAverage method), 9  
`__init__()` (pycast.optimization.BaseOptimizationMethod method), 15  
`__iter__()` (pycast.common.timeseries.TimeSeries method), 4  
`__len__()` (pycast.common.timeseries.TimeSeries method), 4  
`__setitem__()` (pycast.common.timeseries.TimeSeries method), 4  
`__str__()` (pycast.common.timeseries.TimeSeries method), 4  
`__weakref__` (pycast.common.timeseries.TimeSeries attribute), 4  
`__weakref__` (pycast.errors.BaseErrorMeasure attribute), 13  
`__weakref__` (pycast.optimization.BaseOptimizationMethod attribute), 15

## A

`add_entry()` (pycast.common.timeseries.TimeSeries method), 4  
`apply()` (pycast.common.timeseries.TimeSeries method), 4

## B

`BaseErrorMeasure` (class in pycast.errors), 13  
`BaseOptimizationMethod` (class in pycast.optimization), 15

## C

`convert_epoch_to_timestamp()` (pycast.common.timeseries.TimeSeries class method), 5  
`convert_timestamp_to_epoch()` (pycast.common.timeseries.TimeSeries class method), 5

## E

`execute()` (pycast.methods.ExponentialSmoothing method), 11  
`execute()` (pycast.methods.HoltMethod method), 11  
`execute()` (pycast.methods.SimpleMovingAverage method), 9  
`ExponentialSmoothing` (class in pycast.methods), 11

## F

`from_json()` (pycast.common.timeseries.TimeSeries class method), 5  
`from_twodim_list()` (pycast.common.timeseries.TimeSeries class method), 5

## G

`get_error()` (pycast.errors.BaseErrorMeasure method), 13  
`GridSearch` (class in pycast.optimization), 15

## H

`HoltMethod` (class in pycast.methods), 11

## I

`initialize()` (pycast.errors.BaseErrorMeasure method), 13  
`initialize_from_sql_cursor()` (pycast.common.timeseries.TimeSeries method), 6  
`is_normalized()` (pycast.common.timeseries.TimeSeries method), 6  
`is_sorted()` (pycast.common.timeseries.TimeSeries method), 6

## L

`linear_interpolation()` (in module `pycast.common.helper`),  
1  
`local_error()` (`pycast.errors.BaseErrorMeasure` method),  
14  
`local_error()` (`pycast.errors.MeanSquaredError` method),  
14  
`local_error()` (`pycast.errors.SymmetricMeanAbsolutePercentageError`  
method), 14

## M

`MeanSquaredError` (class in `pycast.errors`), 14

## N

`normalize()` (`pycast.common.timeseries.TimeSeries`  
method), 6

## O

`optimize()` (`pycast.optimization.BaseOptimizationMethod`  
method), 15  
`optimize()` (`pycast.optimization.GridSearch` method), 15  
`optimize_forecasting_method()` (`py-`  
`cast.optimization.GridSearch` method), 16

## P

`profileMe` (in module `pycast.common.profileme`), 1  
`pycast.common.helper` (module), 1  
`pycast.common.profileme` (module), 1  
`pycast.common.timeseries` (module), 1

## S

`SimpleMovingAverage` (class in `pycast.methods`), 9  
`sort_timeseries()` (`pycast.common.timeseries.TimeSeries`  
method), 6  
`sorted_timeseries()` (`py-`  
`cast.common.timeseries.TimeSeries` method),  
7  
`SymmetricMeanAbsolutePercentageError` (class in `py-`  
`cast.errors`), 14

## T

`TimeSeries` (class in `pycast.common.timeseries`), 3  
`to_gnuplot_datafile()` (`py-`  
`cast.common.timeseries.TimeSeries` method),  
7  
`to_json()` (`pycast.common.timeseries.TimeSeries`  
method), 7  
`to_twodim_list()` (`pycast.common.timeseries.TimeSeries`  
method), 7