PVAnalytics

pvlib

May 28, 2020

CONTENTS:

PVAnalytics is a python library that supports analytics for PV systems. It provides functions for quality control, filtering, and feature labeling and other tools supporting the analysis of PV system-level data.

The source code for PVAnalytics is hosted on github.

CHAPTER

ONE

CONTENTS

1.1 API Reference

1.1.1 Quality

Irradiance

The `check_*_limits_qcrad` functions use the QCRad algorithm[1] to identify irradiance measurements that are beyond physical limits.

quality.irradiance.check_ghi_limits_qcrad(...)	Test for physical limits on GHI using the QCRad criteria.
quality.irradiance.check_dhi_limits_qcrad(...)	Test for physical limits on DHI using the QCRad criteria.
quality.irradiance.check_dni_limits_qcrad(...)	Test for physical limits on DNI using the QCRad criteria.

pvanalytics.quality.irradiance.check_ghi_limits_qcrad

pvanalytics.quality.irradiance.**check_ghi_limits_qcrad**(*ghi*, *solar_zenith*, *dni_extra*, *limits=None*)

Test for physical limits on GHI using the QCRad criteria.

Test is applied to each GHI value. A GHI value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Parameters

- **ghi** (*Series*) – Global horizontal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **limits** (*dict, default QCRAD_LIMITS*) – Must have keys 'ghi_ub' and 'ghi_lb'. For 'ghi_ub' value is a dict with keys {'mult', 'exp', 'min'} and float values. For 'ghi_lb' value is a float.

[1] C. N. Long and Y. Shi, An Automated Quality Assessment and Control Algorithm for Surface Radiation Measurements, The Open Atmospheric Science Journal 2, pp. 23-37, 2008.

Returns True where value passes limits test.

Return type Series

Notes

pvanalytics.quality.irradiance.check_dhi_limits_qcrad

pvanalytics.quality.irradiance.**check_dhi_limits_qcrad**(*dhi*, *solar_zenith*, *dni_extra*, *limits=None*)

Test for physical limits on DHI using the QCRad criteria.

Test is applied to each DHI value. A DHI value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Parameters

- **dhi** (*Series*) – Diffuse horizontal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **limits** (*dict, default QCRAD_LIMITS*) – Must have keys 'dhi_ub' and 'dhi_lb'. For 'dhi_ub' value is a dict with keys {'mult', 'exp', 'min'} and float values. For 'dhi_lb' value is a float.

Returns True where value passes limit test.

Return type Series

Notes

pvanalytics.quality.irradiance.check_dni_limits_qcrad

pvanalytics.quality.irradiance.**check_dni_limits_qcrad**(*dni*, *solar_zenith*, *dni_extra*, *limits=None*)

Test for physical limits on DNI using the QCRad criteria.

Test is applied to each DNI value. A DNI value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Parameters

- **dni** (*Series*) – Direct normal irradiance in W/m^2

- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **limits** (*dict, default QCRAD_LIMITS*) – Must have keys 'dni_ub' and 'dni_lb'. For 'dni_ub' value is a dict with keys {'mult', 'exp', 'min'} and float values. For 'dni_lb' value is a float.

Returns True where value passes limit test.

Return type Series

Notes

All three checks can be combined into a single function call.

quality.irradiance.check_irradiance_limits_qcrad(...)	Test for physical limits on GHI, DHI or DNI using the QCRad criteria.

pvanalytics.quality.irradiance.check_irradiance_limits_qcrad

pvanalytics.quality.irradiance.**check_irradiance_limits_qcrad**(*solar_zenith*, *dni_extra*, *ghi=None*, *dhi=None*, *dni=None*, *limits=None*)

Test for physical limits on GHI, DHI or DNI using the QCRad criteria.

Criteria from[1] are used to determine physically plausible lower and upper bounds. Each value is tested and a value passes if value > lower bound and value < upper bound. Lower bounds are constant for all tests. Upper bounds are calculated as

$$ub = min + mult * dni\_extra * cos(solar\_zenith)^{exp}$$

Note: If any of *ghi*, *dhi*, or *dni* are None, the corresponding element of the returned tuple will also be None.

Parameters

- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dni_extra** (*Series*) – Extraterrestrial normal irradiance in W/m^2
- **ghi** (*Series or None, default None*) – Global horizontal irradiance in W/m^2
- **dhi** (*Series or None, default None*) – Diffuse horizontal irradiance in W/m^2
- **dni** (*Series or None, default None*) – Direct normal irradiance in W/m^2

[1] C. N. Long and Y. Shi, An Automated Quality Assessment and Control Algorithm for Surface Radiation Measurements, The Open Atmospheric Science Journal 2, pp. 23-37, 2008.

- **limits** (*dict, default QCRAD_LIMITS*) – for keys 'ghi_ub', 'dhi_ub', 'dni_ub', value is a dict with keys {'mult', 'exp', 'min'} and float values. For keys 'ghi_lb', 'dhi_lb', 'dni_lb', value is a float.

Returns

- **ghi_limit_flag** (*Series*) – True for each value that is physically possible. None if *ghi* is None.
- **dhi_limit_flag** (*Series*) – True for each value that is physically possible. None if *dni* is None.
- **dhi_limit_flag** (*Series*) – True for each value that is physically possible. None if *dhi* is None.

Notes

References

Irradiance measurements can also be checked for consistency.

quality.irradiance.check_irradiance_consistency_qcrad(...)	Check consistency of GHI, DHI and DNI using QCRad criteria.

pvanalytics.quality.irradiance.check_irradiance_consistency_qcrad

pvanalytics.quality.irradiance.**check_irradiance_consistency_qcrad**(*ghi*, *solar_zenith*, *dhi*, *dni*, *param=None*)

Check consistency of GHI, DHI and DNI using QCRad criteria.

Uses criteria given in[1] to validate the ratio of irradiance components.

> **Warning:** Not valid for night time. While you can pass data from night time to this function, be aware that the truth values returned for that data will not be valid.

Parameters

- **ghi** (*Series*) – Global horizontal irradiance in W/m^2
- **solar_zenith** (*Series*) – Solar zenith angle in degrees
- **dhi** (*Series*) – Diffuse horizontal irradiance in W/m^2
- **dni** (*Series*) – Direct normal irradiance in W/m^2
- **param** (*dict*) – keys are 'ghi_ratio' and 'dhi_ratio'. For each key, value is a dict with keys 'high_zenith' and 'low_zenith'; for each of these keys, value is a dict with keys

[1] C. N. Long and Y. Shi, An Automated Quality Assessment and Control Algorithm for Surface Radiation Measurements, The Open Atmospheric Science Journal 2, pp. 23-37, 2008.

'zenith_bounds', 'ghi_bounds', and 'ratio_bounds' and value is an ordered pair [lower, upper] of float.

Returns

- **consistent_components** (*Series*) – True where *ghi*, *dhi* and *dni* components are consistent.
- **diffuse_ratio_limit** (*Series*) – True where diffuse to GHI ratio passes limit test.

Notes

References

GHI and POA irradiance can be validated against clearsky values to eliminate data that is unrealistically high.

quality.irradiance.clearsky_limits(measured, ...)	Identify irradiance values which do not exceed clearsky values.

pvanalytics.quality.irradiance.clearsky_limits

pvanalytics.quality.irradiance.**clearsky_limits**(*measured*, *clearsky*, *csi_max=1.1*)

Identify irradiance values which do not exceed clearsky values.

Uses `pvlib.irradiance.clearsky_index()` to compute the clearsky index as the ratio of *measured* to *clearsky*. Compares the clearsky index to *csi_max* to identify values in *measured* that are less than or equal to *csi_max*.

Parameters

- **measured** (*Series*) – Measured irradiance in W/m^2.
- **clearsky** (*Series*) – Expected clearsky irradiance in W/m^2.
- **csi_max** (*float, default 1.1*) – Maximum ratio of *measured* to *clearsky* (clearsky index).

Returns True for each value where the clearsky index is less than or equal to *csi_max*.

Return type Series

Notes

Gaps

Identify gaps in the data.

quality.gaps.interpolation_diff(x[, window, ...])	Identify sequences which appear to be linear.
quality.gaps.stale_values_diff(x[, window, ...])	Identify stale values in the data.

pvanalytics.quality.gaps.interpolation_diff

pvanalytics.quality.gaps.**interpolation_diff**(*x*, *window=3*, *rtol=1e-05*, *atol=1e-08*)

Identify sequences which appear to be linear.

Sequences are linear if the first difference appears to be constant. For a window of length N, the last value (index N-1) is flagged if all values in the window appear to be a line segment.

Parameters

- **x** (*Series*) – data to be processed
- **window** (*int, default 3*) – number of sequential values that, if the first difference is constant, are classified as a linear sequence
- **rtol** (*float, default 1e-5*) – tolerance relative to max(abs(x.diff()) for detecting a change
- **atol** (*float, default 1e-8*) – absolute tolerance for detecting a change in first difference

Returns True for each value that is part of a linear sequence

Return type Series

Raises **ValueError** – If window < 3.

Notes

pvanalytics.quality.gaps.stale_values_diff

pvanalytics.quality.gaps.**stale_values_diff**(*x*, *window=3*, *rtol=1e-05*, *atol=1e-08*)

Identify stale values in the data.

For a window of length N, the last value (index N-1) is considered stale if all values in the window are close to the first value (index 0).

Parameters

- **x** (*Series*) – data to be processed
- **window** (*int, default 3*) – number of consecutive values which, if unchanged, indicates stale data
- **rtol** (*float, default 1e-5*) – relative tolerance for detecting a change in data values
- **atol** (*float, default 1e-8*) – absolute tolerance for detecting a change in data values

- **rtol and atol have the same meaning as in** (*Parameters*) –
- **numpy.allclose** –

Returns True for each value that is part of a stale sequence of data

Return type Series

Raises **ValueError** – If window < 2.

Notes

The following functions identify days with incomplete data.

quality.gaps.completeness_score(series[, ...])	Calculate a data completeness score for each day.
quality.gaps.complete(series[, ...])	Select data points that are part of days with complete data.

pvanalytics.quality.gaps.completeness_score

pvanalytics.quality.gaps.**completeness_score**(*series*, *freq=None*, *keep_index=True*)

Calculate a data completeness score for each day.

The completeness score for a given day is the fraction of time in the day for which there is data (a value other than NaN). The time duration attributed to each value is equal to the timestamp spacing of *series*, or *freq* if it is specified. For example, a 24-hour time series with 30 minute timestamp spacing and 24 non-NaN values would have data for a total of 12 hours and therefore a completeness score of 0.5.

Parameters

- **series** (*Series*) – A DatetimeIndexed series.
- **freq** (*str, default None*) – Interval between samples in the series as a pandas frequency string. If None, the frequency is inferred using `pandas.infer_freq()`.
- **keep_index** (*boolean, default True*) – Whether or not the returned series has the same index as *series*. If False the returned series will be indexed by day.

Returns A series of floats giving the completeness score for each day (fraction of the day for which *series* has data).

Return type Series

Raises **ValueError** – If *freq* is longer than the frequency inferred from *series*.

pvanalytics.quality.gaps.complete

pvanalytics.quality.gaps.**complete**(*series*, *minimum_completeness=0.333*, *freq=None*)

Select data points that are part of days with complete data.

A day has complete data if its completeness score is greater than or equal to *minimum_completeness*. The completeness score is calculated by *completeness_score()*.

Parameters

- **series** (*Series*) – The data to be checked for completeness.
- **minimum_completeness** (*float, default 0.333*) – Fraction of the day that must have data.
- **freq** (*str, default None*) – The expected frequency of the data in *series*. If none then the frequency is inferred from the data.

Returns A series of booleans with True for each value that is part of a day with completeness greater than *minimum_completeness*.

Return type Series

Raises **ValueError** – See *completeness_score()*.

See also:

completeness_score()

Many data sets may have leading and trailing periods of days with sporadic or no data. The following functions can be used to remove those periods.

quality.gaps.start_stop_dates(series[, days])	Get the start and end of data excluding leading and trailing gaps.
quality.gaps.trim(series[, days])	Mask the beginning and end of the data if not all True.
quality.gaps.trim_incomplete(series[, ...])	Trim the series based on the completeness score.

pvanalytics.quality.gaps.start_stop_dates

pvanalytics.quality.gaps.**start_stop_dates**(*series*, *days=10*)

Get the start and end of data excluding leading and trailing gaps.

Parameters

- **series** (*Series*) – A DatetimeIndexed series of booleans.
- **days** (*int, default 10*) – The minimum number of consecutive days where every value in *series* is True for data to start or stop.

Returns

- **start** (*Datetime or None*) – The first valid day. If there are no sufficiently long periods of valid days then None is returned.
- **stop** (*Datetime or None*) – The last valid day. None if start is None.

pvanalytics.quality.gaps.trim

pvanalytics.quality.gaps.**trim**(*series*, *days=10*)

Mask the beginning and end of the data if not all True.

Parameters

- **series** (*Series*) – A DatetimeIndexed series of booleans
- **days** (*int, default 10*) – Minimum number of consecutive days that are all True for 'good' data to start.

Returns A series of booleans with True for all data points between the first and last block of *days* consecutive days that are all True in *series*. If *series* does not contain such a block of consecutive True values, then the returned series will be entirely False.

Return type Series

See also:

start_stop_dates()

pvanalytics.quality.gaps.trim_incomplete

pvanalytics.quality.gaps.**trim_incomplete**(*series*, *minimum_completeness=0.333333*, *days=10*, *freq=None*)

Trim the series based on the completeness score.

Combines *completeness_score()* and *trim()*.

Parameters

- **series** (*Series*) – A DatetimeIndexed series.
- **minimum_completeness** (*float, default 0.333333*) – The minimum completeness score for each day.
- **days** (*int, default 10*) – The number of consecutive days with completeness greater than *minumum_completeness* for the 'good' data to start or end. See *start_stop_dates()* for more information.
- **freq** (*str, default None*) – The expected frequency of the series. See *completeness_score()* fore more information.

Returns A series of booleans with the same index as *series* with False up to the first complete day, True between the first and the last complete days, and False following the last complete day.

Return type Series

See also:

trim(), *completeness_score()*

Outliers

Functions for detecting outliers.

quality.outliers.tukey(data[, k])	Identify outliers based on the interquartile range.
quality.outliers.zscore(data[, zmax])	Identify outliers using the z-score.
quality.outliers.hampel(data[, window, ...])	Identify outliers by the Hampel identifier.

pvanalytics.quality.outliers.tukey

pvanalytics.quality.outliers.**tukey**(*data*, *k=1.5*)

Identify outliers based on the interquartile range.

A value *x* is considered an outlier if it does *not* satisfy the following condition

$$Q_1 - k(Q_3 - Q_1) \leq x \leq Q_3 + k(Q_3 - Q_1)$$

where Q_1 is the value of the first quartile and Q_3 is the value of the third quartile.

Parameters

- **data** (*Series*) – The data in which to find outliers.
- **k** (*float, default 1.5*) – Multiplier of the interquartile range. A larger value will be more permissive of values that are far from the median.

Returns A series of booleans with True for each value that is an outlier.

Return type Series

pvanalytics.quality.outliers.zscore

pvanalytics.quality.outliers.**zscore**(*data*, *zmax=1.5*)

Identify outliers using the z-score.

Points with z-score greater than *zmax* are considered as outliers.

Parameters

- **data** (*Series*) – A series of numeric values in which to find outliers.
- **zmax** (*float*) – Upper limit of the absolute values of the z-score.

Returns A series of booleans with True for each value that is an outlier.

Return type Series

pvanalytics.quality.outliers.hampel

pvanalytics.quality.outliers.**hampel**(*data*, *window=5*, *max_deviation=3.0*, *scale=1.4826*)

Identify outliers by the Hampel identifier.

The Hampel identifier is computed according to[1].

Parameters

- **data** (*Series*) – The data in which to find outliers.
- **window** (*int or offset, default 5*) – The size of the rolling window used to compute the Hampel identifier.
- **max_deviation** (*float, default 3.0*) – Any value with a Hampel identifier > *max_deviation* standard deviations from the median is considered an outlier.
- **scale** (*float, default 1.4826*) – MAD scale estimate. The standard deviation is calculated as $scale * MAD$. The default gives an estimate for the standard deviation of Gaussian distributed data.

Returns True for each value that is an outlier according to its Hampel identifier.

Return type Series

References

Time

Quality control related to time. This includes things like time-stamp spacing, time-shifts, and time zone validation.

[1] Pearson, R.K., Neuvo, Y., Astola, J. et al. Generalized Hampel Filters. EURASIP J. Adv. Signal Process. 2016, 87 (2016). https://doi.org/10.1186/s13634-016-0383-6

quality.time.spacing(times, freq)	Check that the spacing between *times* conforms to *freq*.

pvanalytics.quality.time.spacing

pvanalytics.quality.time.**spacing**(*times*, *freq*)

Check that the spacing between *times* conforms to *freq*.

Parameters

- **times** (*DatetimeIndex*) –
- **freq** (*string or Timedelta*) – Expected frequency of *times*.

Returns True when the difference between one time and the time before it conforms to *freq*.

Return type Series

Notes

Weather

Quality checks for weather data.

quality.weather.relative_humidity_limits(...)	Identify relative humidity values that are within limits.
quality.weather.temperature_limits(... [, limits])	Identify temperature values that are within limits.
quality.weather.wind_limits(wind_speed[, limits])	Identify wind speed values that are within limits.

pvanalytics.quality.weather.relative_humidity_limits

pvanalytics.quality.weather.**relative_humidity_limits**(*relative_humidity*, *limits=(0, 100)*)

Identify relative humidity values that are within limits.

Parameters

- **relative_humidity** (*Series*) – Relative humidity in %.
- **limits** (*tuple, default (0, 100)*) – (lower bound, upper bound) for relative humidity.

Returns True if *relative_humidity* >= lower bound and *relative_humidity* <= upper_bound.

Return type Series

Notes

pvanalytics.quality.weather.temperature_limits

pvanalytics.quality.weather.**temperature_limits**(*air_temperature*, *limits=(-35.0, 50.0)*)

Identify temperature values that are within limits.

Parameters

- **air_temperature** (*Series*) – Air temperature [C].
- **temp_limits** (*tuple, default (-35, 50)*) – (lower bound, upper bound) for temperature.

Returns True if *air_temperature* > lower bound and *air_temperature* < upper bound.

Return type Series

Notes

pvanalytics.quality.weather.wind_limits

pvanalytics.quality.weather.**wind_limits**(*wind_speed*, *limits=(0.0, 50.0)*)

Identify wind speed values that are within limits.

Parameters

- **wind_speed** (*Series*) – Wind speed in m/s
- **wind_limits** (*tuple, default (0, 50)*) – (lower bound, upper bound) for wind speed.

Returns True if *wind_speed* >= lower bound and *wind_speed* < upper bound.

Return type Series

Notes

1.1.2 Features

Functions for detecting features in the data.

Clipping

Functions for identifying inverter clipping

features.clipping.levels(ac_power[, window, ...])	Label clipping in AC power data based on levels in the data.

pvanalytics.features.clipping.levels

pvanalytics.features.clipping.**levels**(*ac_power*, *window=4*, *fraction_in_window=0.75*, *rtol=0.005*, *levels=2*)

Label clipping in AC power data based on levels in the data.

Parameters

- **ac_power** (*Series*) – Time series of AC power measurements.
- **window** (*int, default 4*) – Number of data points in a window used to detect clipping.
- **fraction_in_window** (*float, default 0.75*) – Fraction of points which indicate clipping if AC power at each point is close to the plateau level.
- **rtol** (*float, default 5e-3*) – A point is close to a clipped level M if abs(ac_power - M) < rtol * max(ac_power)
- **levels** (*int, default 2*) – Number of clipped power levels to consider.

Returns True when clipping is indicated.

Return type Series

Notes

Copyright (c) 2019 SolarArbiter. See the file LICENSES/SOLARFORECASTARBITER_LICENSE at the top level directory of this distribution and at https://github.com/pvlib/pvanalytics/blob/master/LICENSES/SOLARFORECASTARBITER_LICENSE for more information.

Clearsky

features.clearsky.reno(ghi, ghi_clearsky)	Identify times when GHI is consistent with clearsky conditions.

pvanalytics.features.clearsky.reno

pvanalytics.features.clearsky.**reno**(*ghi*, *ghi_clearsky*)

Identify times when GHI is consistent with clearsky conditions.

Uses the function `pvlib.clearsky.detect_clearsky()`.

Note: Must be given GHI data with regular (constant) time intervals of 15 minutes or less.

Parameters

- **ghi** (*Series*) – Global horizontal irradiance in W/m^2. Must have an index with time intervals of at most 15 minutes.
- **ghi_clearsky** (*Series*) – Global horizontal irradiance in W/m^2 under clearsky conditions.

Returns True when clear sky conditions are indicated.

Return type Series

Raises **ValueError** – if the time intervals are greater than 15 minutes.

Notes

Clear-sky conditions are inferred when each of six criteria are met; see `pvlib.clearsky.detect_clearsky()` for references and details. Threshold values for each criterion were originally developed for ten minute windows containing one-minute data[1]. As indicated in[2], the algorithm also works for longer windows and data at different intervals if threshold criteria are roughly scaled to the window length. Here, the threshold values are based on [1] with the scaling indicated in [2].

References

Time

The following functions can be used to differentiate night-time and day-time based on power or irradiance data. This is useful if you do not know the timezone of the index for your data or for verifying that the timezone is correct. Both functions identify the same feature, but in slightly different ways. *features.daylight.frequency()* is useful if your data has substantial outliers or other excessively large values; however, it may fail if substantial portions of the night-time data is greater than 0. *features.daylight.level()* can handle positive night-time data (so long as night-time values are substantially lower than day-time data) but may be more suceptible to large outliers.

features.daylight.frequency(power_or_irradiance)	Identify daytime periods based on frequency of positive data.
features.daylight.level(power_or_irradiance)	Identify daytime periods based on a minimum power/irradiance threshold.

pvanalytics.features.daylight.frequency

pvanalytics.features.daylight.**frequency**(*power_or_irradiance*, *threshold=0.8*, *minimum_days=60*)

Identify daytime periods based on frequency of positive data.

Data is aggregated by minute of the day and the mean number of positive values for each minute is calculated. Each minute with at least *threshold* times the mean number of positive values is considered day-time.

Parameters

[1] Reno, M.J. and C.W. Hansen, "Identification of periods of clear sky irradiance in time series of GHI measurements" Renewable Energy, v90, p. 520-531, 2016.

[2] B. H. Ellis, M. Deceglie and A. Jain, "Automatic Detection of Clear-Sky Periods From Irradiance Data," in IEEE Journal of Photovoltaics, vol. 9, no. 4, pp. 998-1005, July 2019. doi: 10.1109/JPHOTOV.2019.2914444

- **power_or_irradiance** (*Series*) – DatetimeIndexed series of power or irradiance measurements.
- **threshold** (*float, default 0.8*) – Fraction of data that must have positive power or irradiance measurements for a time to be considered "day".
- **minimum_days** (*int, default 60*) – Minimum number of days with data. If *power_or_irradiance* has fewer days with positive data then a ValueError is raised.

Returns A DatetimeIndexed series with True for each index that is during daylight hours.

Return type Series

Raises **ValueError** – if there are less than *minimum_days* of positive data.

Notes

This function is derived from the pvfleets_qa_analysis project. Copyright (c) 2020 Alliance for Susteainable Energy, LLC.

pvanalytics.features.daylight.level

pvanalytics.features.daylight.**level**(*power_or_irradiance*, *threshold=0.2*, *quantile=0.95*)

Identify daytime periods based on a minimum power/irradiance threshold.

Power or irradiance data is aggregated by minute of day and the mean is computed at each minute. A minute is marked as daytime when the mean value is greater than or equal to *threshold* times the *quantile*-percent of *power_or_irradiance*.

Parameters

- **power_or_irradiance** (*Series*) – DatetimeIndexed power or irradiance data.
- **threshold** (*float, default 0.2*) – Mean power at each minute of the day must be greater than or equal to *threshold* * max where max is the *quantile*-percent quantile of the data for the minute to be considered daytime.
- **quantile** (*float, default 0.95*) – Quantile to use as the upper bound of the data.

Returns A series of booleans with True for timestamps when the sun is up.

Return type Series

References

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX