PVAnalytics

pvlib

Aug 21, 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

You may want to identify entire days that have unrealistically high or low insolation. The following function examines daily insolation, validating that it is within a reasonable range of the expected clearsky insolation for the same day.

quality.irradiance.daily_insolation_limits(...)	Check that daily insolation lies between minimum and maximum values.

pvanalytics.quality.irradiance.daily_insolation_limits

pvanalytics.quality.irradiance.**daily_insolation_limits**(*irrad*, *clearsky*, *daily_min=0.4*, *daily_max=1.25*)

Check that daily insolation lies between minimum and maximum values.

Irradiance measurements and clear-sky irradiance on each day are integrated with the trapezoid rule to calculate daily insolation.

Parameters

- **irrad** (*Series*) – Irradiance measurements (GHI or POA).
- **clearsky** (*Series*) – Clearsky irradiance.
- **daily_min** (*float, default 0.4*) – Minimum ratio of daily insolation to daily clearsky insolation.
- **daily_max** (*float, default 1.25*) – Maximum ratio of daily insolation to daily clearsky insolation.

Returns True for values on days where the ratio of daily insolation to daily clearsky insolation is between *daily_min* and *daily_max*.

Return type Series

Notes

The default limits (*daily_max* and *daily_min*) have been set for GHI and POA irradiance for systems with *fixed* azimuth and tilt. If you pass POA irradiance for a tracking system it is recommended that you increase *daily_max* to 1.35.

The default values for *daily_min* and *daily_max* were taken from the PVFleets QA Analysis project.

Gaps

Identify gaps in the data.

quality.gaps.interpolation_diff(x[, window, ...])	Identify sequences which appear to be linear.

pvanalytics.quality.gaps.interpolation_diff

pvanalytics.quality.gaps.**interpolation_diff**(*x*, *window=6*, *rtol=1e-05*, *atol=1e-08*, *mark='tail'*)

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 *rtol* and *atol* have the same meaning as in `numpy.allclose()`.

Parameters

- **x** (*Series*) – data to be processed

- **window** (*int, default 6*) – 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
- **mark** (*str, default 'tail'*) – How much of the window to mark `True` when a sequence of interpolated values is detected. Can be one of 'tail', 'end', or 'all'.
  - If 'tail' (the default) then every point in the window *except* the first point is marked `True`.
  - If 'end' then the first *window - 1* values in an interpolated sequence are marked `False` and all subsequent values in the sequence are marked `True`.
  - If 'all' then every point in the window *including* the first point is marked `True`.

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

Return type Series

Raises **ValueError** – If *window* < *3* or *mark* is not one of 'tail', 'end', or 'all'.

Notes

Data sometimes contains sequences of values that are "stale" or "stuck." These are contiguous spans of data where the value does not change within the precision given. The functions below can be used to detect stale values.

Note: If the data has been altered in some way (i.e. temperature that has been rounded to an integer value) before being passed to these functions you may see unexpectedly large amounts of stale data.

quality.gaps.stale_values_diff(x[, window, ...])	Identify stale values in the data.
quality.gaps.stale_values_round(x[, window, ...])	Identify stale values by rounding.

pvanalytics.quality.gaps.stale_values_diff

pvanalytics.quality.gaps.**stale_values_diff**(*x*, *window=6*, *rtol=1e-05*, *atol=1e-08*, *mark='tail'*)

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 *rtol* and *atol* have the same meaning as in `numpy.allclose()`.

Parameters

- **x** (*Series*) – data to be processed

- **window** (*int, default 6*) – 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
- **mark** (*str, default 'tail'*) – How much of the window to mark `True` when a sequence of stale values is detected. Can one be of 'tail', 'end', or 'all'.
  - If 'tail' (the default) then every point in the window *except* the first point is marked `True`.
  - If 'end' then the first *window - 1* values in a stale sequence sequence are marked `False` and all subsequent values in the sequence are marked `True`.
  - If 'all' then every point in the window *including* the first point is marked `True`.

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

Return type Series

Raises **ValueError** – If *window* < *2* or *mark* is not one of 'tail', 'end', or 'all'.

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.

pvanalytics.quality.gaps.stale_values_round

pvanalytics.quality.gaps.**stale_values_round**(*x*, *window=6*, *decimals=3*, *mark='tail'*)

Identify stale values by rounding.

A value is considered stale if it is part of a sequence of length *window* of values that are identical when rounded to *decimals* decimal places.

Parameters

- **x** (*Series*) – Data to be processed.
- **window** (*int, default 6*) – Number of consecutive identical values for a data point to be considered stale.
- **decimals** (*int, default 3*) – Number of decimal places to round to.
- **mark** (*str, default 'tail'*) – How much of the window to mark `True` when a sequence of stale values is detected. Can be one of 'tail', 'end', or 'all'.
  - If 'tail' (the default) then every point in the window *except* the first point is marked `True`.
  - If 'end' then the first *window - 1* values in a stale sequence sequence are marked `False` and all subsequent values in the sequence are marked `True`.
  - If 'all' then every point in the window *including* the first point is marked `True`.

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

Return type Series

Raises **ValueError** – If *mark* is not one of 'tail', 'end', or 'all'.

Notes

Based on code from the pvfleets_qa_analysis project. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

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.

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

[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

- **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

Utilities

The `quality.util` module contains general-purpose/utility functions for building your own quality checks.

quality.util.check_limits(val[, ...])	Check whether a value falls withing the given limits.
quality.util.daily_min(series, minimum[, ...])	Return True for data on days when the day's minimum exceeds *minimum*.

pvanalytics.quality.util.check_limits

pvanalytics.quality.util.**check_limits**(*val*, *lower_bound=None*, *upper_bound=None*, *inclusive_lower=False*, *inclusive_upper=False*)

Check whether a value falls withing the given limits.

At least one of *lower_bound* or *upper_bound* must be provided.

Parameters

- **val** (*array_like*) – Values to test.
- **lower_bound** (*float, default None*) – Lower limit.
- **upper_bound** (*float, default None*) – Upper limit.
- **inclusive_lower** (*bool, default False*) – Whether the lower bound is inclusive (*val* >= *lower_bound*).
- **inclusive_upper** (*bool, default False*) – Whether the upper bound is inclusive (*val* <= *upper_bound*).

Returns True for every value in *val* that is between *lower_bound* and *upper_bound*.

Return type array_like

Raises **ValueError** – if *lower_bound* nor *upper_bound* is provided.

Notes

pvanalytics.quality.util.daily_min

pvanalytics.quality.util.**daily_min**(*series*, *minimum*, *inclusive=False*)

Return True for data on days when the day's minimum exceeds *minimum*.

Parameters

- **series** (*Series*) – A Datetimeindexed series of floats.
- **minimum** (*float*) – The smallest acceptable value for the daily minimum.
- **inclusive** (*boolean, default False*) – Use greater than or equal to when comparing daily minimums from *series* to *minimum*.

Returns True for values on days where the minimum value recorded on that day is greater than (or equal to) *minimum*.

Return type Series

Notes

This function is derived from code in the pvfleets_qa_analysis project under the terms of the 3-clause BSD license. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

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

In addition to validating temperature by comparing with limits, module temperature should be positively correlated with irradiance. Poor correlation could indicate that the sensor has become detached from the module, for example. Unlike other functions in the `quality` module which return Boolean masks over the input series, this function returns a single Boolean value indicating whether the entire series has passed (`True`) or failed (`False`) the quality check.

quality.weather.module_temperature_check(...)	Test whether the module temperature is correlated with irradiance.

pvanalytics.quality.weather.module_temperature_check

pvanalytics.quality.weather.**module_temperature_check**(*module_temperature*, *irradiance*, *correlation_min=0.5*)

Test whether the module temperature is correlated with irradiance.

Parameters

- **module_temperature** (*Series*) – Time series of module temperature.
- **irradiance** (*Series*) – Time series of irradiance with the same index as *module_temperature*. This should be of relatively high quality (outliers and other problems removed).
- **correlation_min** (*float, default 0.5*) – Minimum correlation between *module_temperature* and *irradiance* for the module temperature sensor to 'pass'

Returns True if the correlation between *module_temperature* and *irradiance* exceeds *correlation_min*.

Return type bool

References

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.
features.clipping.threshold(ac_power[, ...])	Detect clipping based on a maximum power threshold.

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

pvanalytics.features.clipping.threshold

pvanalytics.features.clipping.**threshold**(*ac_power*, *slope_max=0.0035*, *power_min=0.75*, *power_quantile=0.995*, *freq=None*)

Detect clipping based on a maximum power threshold.

This is a two-step process. First a clipping threshold is identified, then any values in *ac_power* greater than or equal to that threshold are flagged.

The clipping threshold is determined by computing a 'daily power curve' which is the *power_quantile* quantile of all values in *ac_power* at each minute of the day. This gives a rough estimate of the maximum power produced at each minute of the day.

The daily power curve is normalized by its maximum and the minutes of the day are identified where the normalized curve's slope is less than *slope_max*. If there is a continuous period of time spanning at least one hour where the slope is less than *slope_max* and the value of the normalized daily power curve is greater than *power_min* times the median of the normalized daily power curve then the data has clipping in it. If no sufficiently long period with both a low slope and high power exists then there is no clipping in the data. The average of the daily power curve (not normalized) during the longest period that satisfies the criteria above is the clipping threshold.

Parameters

- **ac_power** (*Series*) – DatetimeIndexed series of AC power data.
- **slope_max** (*float, default 0.0035*) – Maximum absolute value of slope of AC power quantile for clipping to be indicated. The default value has been derived empirically to prevent false positives for tracking PV systems.
- **power_min** (*float, default 0.75*) – The power during periods with slope less than *slope_max* must be greater than *power_min* times the median normalized daytime power.
- **power_quantile** (*float, default 0.995*) – Quantile used to calculate the daily power curve.
- **freq** (*string, default None*) – A pandas string offset giving the frequency of data in *ac_power*. If None then the frequency is inferred from the series index.

Returns True when *ac_power* is greater than or equal to the clipping threshold.

Return type Series

Notes

This function is based on the pvfleets_qa_analysis project.

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

[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

Orientation

System orientation refers to mounting type (fixed or tracker) and the azimuth and tilt of the mounting. A system's orientation can be determined by examining power or POA irradiance on days that are relatively sunny.

This module provides functions that operate on power or POA irradiance to identify system orientation on a daily basis. These functions can tell you whether a day's profile matches that of a fixed system or system with a single-axis tracker.

Care should be taken when interpreting function output since other factors such as malfunctioning trackers can interfere with identification.

features.orientation.fixed_nrel(...[, ...])	Flag days that match the profile of a fixed PV system on a sunny day.
features.orientation.tracking_nrel(...[, ...])	Flag days that match the profile of a single-axis tracking PV system on a sunny day.

pvanalytics.features.orientation.fixed_nrel

pvanalytics.features.orientation.**fixed_nrel**(*power_or_irradiance*, *daytime*, *r2_min=0.94*, *min_hours=5*, *peak_min=None*)

Flag days that match the profile of a fixed PV system on a sunny day.

This algorithm relies on the observation that the power profile of a fixed tilt PV system often resembles a quadratic polynomial on a sunny day, with a single peak when the sun is near the system azimuth.

A day is marked True when the r^2 for a quadratic fit to the power data is greater than *r2_min*.

Parameters

- **power_or_irradiance** (*Series*) – Timezone localized series of power or irradiance measurements.
- **daytime** (*Series*) – Boolean series with True for times that are during the day. For best results this mask should exclude early morning and evening as well as night. Data at these times may have problems with shadows that interfere with curve fitting.
- **r2_min** (*float, default 0.94*) – Minimum r^2 of a quadratic fit for a day to be marked True.
- **min_hours** (*float, default 5.0*) – Minimum number of hours with data to attempt a fit on a day.
- **peak_min** (*float, default None*) – The maximum *power_or_irradiance* value for a day must be greater than *peak_min* for a fit to be attempted. If the maximum for a day is less than *peak_min* then the day will be marked False.

Returns True for values on days where *power_or_irradiance* matches the expected parabolic profile for a fixed PV system on a sunny day.

Return type Series

Notes

This algorithm is based on the PVFleets QA Analysis project. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

pvanalytics.features.orientation.tracking_nrel

pvanalytics.features.orientation.**tracking_nrel**(*power_or_irradiance*, *daytime*, *r2_min=0.915*, *r2_fixed_max=0.96*, *min_hours=5*, *peak_min=None*, *quadratic_mask=None*)

Flag days that match the profile of a single-axis tracking PV system on a sunny day.

This algorithm relies on the observation that the power profile of a single-axis tracking PV system tends to resemble a quartic polynomial on a sunny day, I.e., two peaks are observed, one before and one after the sun crosses the tracker azimuth. By contrast, the power profile for a fixed tilt PV system often resembles a quadratic polynomial on a sunny day, with a single peak when the sun is near the system azimuth.

The algorithm fits both a quartic and a quadratic polynomial to each day's data. A day is marked True if the quartic fit has a sufficiently high r^2 and the quadratic fit has a sufficiently low r^2. Specifically, a day is marked True when three conditions are met:

1. a restricted quartic[1] must fit the data with r^2 greater than *r2_min*
2. the r^2 for the restricted quartic fit must be greater than the r^2 for a quadratic fit
3. the r^2 for a quadratic fit must be less than *r2_fixed_max*

Values on days where any one of these conditions is not met are marked False.

Parameters

- **power_or_irradiance** (*Series*) – Timezone localized series of power or irradiance measurements.
- **daytime** (*Series*) – Boolean series with True for times that are during the day. For best results this mask should exclude early morning and late afternoon as well as night. Data at these times may have problems with shadows that interfere with curve fitting.
- **r2_min** (*float, default 0.915*) – Minimum r^2 of a quartic fit for a day to be marked True.
- **r2_fixed_max** (*float, default 0.96*) – If the r^2 of a quadratic fit exceeds *r2_fixed_max*, then tracking/fixed cannot be distinguished and the day is marked False.
- **min_hours** (*float, default 5.0*) – Minimum number of hours with data to attempt a fit on a day.
- **peak_min** (*float, default None*) – The maximum *power_or_irradiance* value for a day must be greater than *peak_min* for a fit to be attempted. If the maximum for a day is less than *peak_min* then the day will be marked False.
- **quadratic_mask** (*Series, default None*) – If None then *daytime* is used. This Series is used to remove morning and afternoon times from the data before applying a quadratic fit. The mask should typically exclude more data than *daytime* in order to eliminate long tails in the morning or afternoon that can appear if a tracker is stuck in a West or East orientation.

Returns Boolean series with True for every value on a day that has a tracking profile (see criteria above).

Return type Series

[1] The specific quartic used for this fit is centered within 70 minutes of 12:00, the y-value at the center must be within 15% of the median for the day, and it must open downwards.

Notes

This algorithm is based on the PVFleets QA Analysis project. Copyright (c) 2020 Alliance for Sustainable Energy, LLC.

1.1.3 System

The following function can be used to infer system orientation from power or plane of array irradiance measurements.

system.infer_orientation_daily_peak(...)	Determine system azimuth and tilt from power or POA using solar azimuth at the daily peak.

pvanalytics.system.infer_orientation_daily_peak

pvanalytics.system.**infer_orientation_daily_peak**(*power_or_poa*, *sunny*, *tilts*, *azimuths*, *solar_azimuth*, *solar_zenith*, *ghi*, *dhi*, *dni*)

Determine system azimuth and tilt from power or POA using solar azimuth at the daily peak.

The time of the daily peak is estimated by fitting a quadratic to to the data for each day in *power_or_poa* and finding the vertex of the fit. A brute force search is performed on clearsky POA irradiance for all pairs of candidate azimuths and tilts (*azimuths* and *tilts*) to find the pair that results in the closest azimuth to the azimuths calculated at the peak times from the curve fitting step. Closest is determined by minimizing the sum of squared difference between the solar azimuth at the peak time in *power_or_poa* and the solar azimuth at maximum clearsky POA irradiance.

The accuracy of the tilt and azimuth returned by this function will vary with the time-resolution of the clearsky and solar position data. For the best accuracy pass *solar_azimuth*, *solar_zenith*, and the clearsky data (*ghi*, *dhi*, and *dni*) with one-minute timestamp spacing. If *solar_azimuth* has timestamp spacing less than one minute it will be resampled and interpolated to estimate azimuth at each minute of the day. Regardless of the timestamp spacing these parameters must cover the same days as *power_or_poa*.

Parameters

- **power_or_poa** (*Series*) – Timezone localized series of power or POA irradiance measurements.
- **sunny** (*Series*) – Boolean series with True for values during clearsky conditions.
- **tilts** (*array-like*) – Candidate tilts in degrees.
- **azimuths** (*array-like*) – Candidate azimuths in degrees.
- **solar_azimuth** (*Series*) – Time series of solar azimuth.
- **solar_zenith** (*Series*) – Time series of solar zenith.
- **ghi** (*Series*) – Clear sky GHI.
- **dhi** (*Series*) – Clear sky DHI.
- **dni** (*Series*) – Clear sky DNI.

Returns

- **azimuth** (*float*)
- **tilt** (*float*)

Notes

Based on PVFleets QA project.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX