

pvcaptest Training Session

2024 PVPMC Workshop

Salt Lake City, UT

May 9, 2024

Introduction

Welcome to the 2024 PVPMC pvcaptest training session!

We will be working through installing pvcaptest and then using it within a Jupyter notebook by conducting an example capacity test of a bifacial PV plant.

My name is Ben Taylor and I am the Principal Consultant of Tailored Data Consulting and the creator of pvcaptest.

- github.com/bt-
- LinkedIn
- ben [at] td [dot] consulting

pvcaptest resouces:

- pvcaptest documentation
- Source code on github

pvcaptest

pvcaptest is an open source Python package that facilitates conducting capacity tests following ASTM E2848 - Standard Test Method for Reporting Photovoltaic Non-Concentrator System Performance. The pvcaptest package provides functionality for loading, visualizing, filtering, and regressing capacity testing data as well as summarizing results. pvcaptest is provided as a free open-source tool with the following goals:

- Provide a more efficient, enjoyable and informative workflow for those conducting capacity tests (Be better than Excel)
- Serve as a common tool, so all parties involved in conducting a capacity test could potentially use the same calculations
- Shift time and energy of testing parties away from validating calculations toward investigating performance

- Provide a scriptable tool to facilitate wrapping capacity ratios as a performance metric into larger or long term analysis

There is also the `prtest` module, which includes some basic functionality related to Performance Ratios.

PV Capacity Testing

This tutorial is focused on the use of pvcaptest rather than the capacity testing standard and procedures, but we will begin with a brief review of the capacity test procedure defined by ASTM E2848.

ASTM E2848 - Standard Test Method for Reporting Photovoltaic Non-Concentrator System Performance

> This test method provides measurement and analysis procedures for determining the capacity of a specific photovoltaic system built in a particular place and in operation under natural sunlight.

ASTM E2939 - Standard Practice for Determining Reporting Conditions and Expected Capacity for Photovoltaic Non-Concentrator Systems

> This practice provides procedures for determining the expected capacity of a specific photovoltaic system in a specific geographical location that is in operation under natural sunlight during a specified period of time. The expected capacity is intended for comparison with the measured capacity determined by Test Method E2848.

ASTME E2848 describes a regression model of plant power output based on ambient conditions:

$$P = E(a_1 + a_2 \cdot E + a_3 \cdot T_a + a_4 \cdot v)$$

where,

P - Photovoltaic system power
E - Plane of Array (POA) irradiance
T_a - Ambient temperature
v - wind speed
$a_{1,2,3,4}$ - Regression coefficients

The general steps to conduct a test are shown in the figure below. Nearly the same process of filtering data, fitting the regression, and predicting power at reporting conditions (RCs) is applied for the data from the site and from the energy model. The reporting conditions are values for POA irradiance, ambient temperature, and wind speed that are representative of a clear day during the

data collection period. A *single* set of reporting conditions may be determined from the *either* the measured or modeled data. The RCs are used with the regression formula and two sets regression coefficients to predict measured capacity (site data) and target capacity (modeled data). The ratio of measured capacity to target capacity is the capacity ratio, which is used to evaluate if the capacity test is producing the power expected.



Diagram adapted from Adam Kankiewicz's presentation at PVPMC Workshop May 2023, slide 7.

Tools / Environment

Jupyter Lab / Notebook

This tutorial is written in an ipython notebook (ipynb) and intended to be used in either Jupyter Lab or Jupyter Notebook. I am using Jupyter Lab. The Jupyter documentation describes notebooks as follows:

> A notebook is a shareable document that combines computer code, plain language descriptions, data, rich visualizations like 3D models, charts, graphs and figures, and interactive controls. A notebook, along with an editor (like JupyterLab), provides a fast interactive environment for prototyping and explaining code, exploring and visualizing data, and sharing ideas with others.

This is a format that is particularly well-suited to presenting a capacity test using pvcaptest.

Notebooks are built on top of IPython, which is an interactive Python shell and was the most popular REPL for Python before notebooks. The notebook is made up of cells. There are two types of input cells (code and Markdown) and output cells. Code cells are used to input Python code, which is executed by running the cell through the GUI or by pressing `ctrl + enter` or `shift + enter`. Output cells will show the result of a code cell that generates output, like a print statement or a plotting function.

IPYNB files can be converted using either the `jupyter nbconvert` command (installed with Jupyter lab/notebook) or from the GUI into html files, which will retain most of the interactivity of the pvcaptest data vizualizations.

Python

pvcpatest is written in Python.

> Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. (Wikipedia

There is intentionally not a link to the Python webpage! If you're new to Python you might google

Python, go to the website, and go to the downloads or getting started page. I do not recommend this as you very likely do NOT want to use the version of Python already installed and also do not want to download and install Python itself directly. Skip down to the Installation section to see what you should do to install Python.

Pandas

"Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language." pvcaptest is built on top of Pandas (and other tools). The primary class of pvcaptest, CapData, holds two Pandas DataFrames and provides methods that act on those DataFrames. DataFrames are Pandas equivalent of an Excel worksheet and are for working with tabular data contained in columns.

Statsmodels

Statsmodels is used within pvcaptest handle functionality related to the regressions including fitting and prediction. pvcaptest relies on statsmodels use of the patsy package to allow defining the regression equation with a simple string representing a formula using names of columns of the `CapData.data` DataFrame.

The regression formula is set by default to match the regression formula of the standard and won't need to be changed for a typical capacity test or even the bifacial test of this tutorial. The pvcaptest documentation section Other Bifacial Capacity Test Approaches provides additional detail on this functionality.

pvlib

pvlib is used internally to provide modeled clear sky irradiance and filtering for clear sky periods. Thank you to the pvlib creators and maintainers for a very useful and well documented package!

Conda

> conda provides package, dependency, and environment management for any language.

There are many ways to use Python. pvcaptest recommends installing pvcaptest and its dependencies (Python, Pandas, statsmodels, etc) in an isolated environment and recommends using conda to do this. There are other options available as well.

See the Installing Miniforge section to see recommended way to install conda. Conda, the environment and package manager, was developed by Anaconda (the company) and is open-source and free to use, but the Anaconda (distribution, lots of packages) and Miniconda provided by Anaconda (the company) has more complicated licensing.

Installation

Obtain the Tutorial Files

1. Go to this link to download the tutorial files and donwload the files:

https://bit.ly/pvpmc2024-pvcaptest

You may see a popup to sign in or register. If you do there should be a continue to download link at the bottom of the pop-up.

2. Move the files to your home folder.

Windows: `C:\Users\username`

OSX: `/Users/username`

3. Extract the zip file. You should now have the following files/directories accessible in a folder in your home directory called `pvpmc2024` .

- column_groups.xlsx
- data
- environment.yml
- images
- pvcaptest_training_pvpmc_2024.html
- pvcaptest_training_pvpmc_2024.ipynb
- pvsyst
- site.yml

4. The html file is an html version of the ipynb that I am showing. Open this if you would like to follow along with a copy of the tutorial on your screen. Once we've finished the installation section you'll be able to switch to the ipynb in Jupyter.

Installing Miniforge

The first step to using pvcaptest is to install pvcaptest and Jupyter into an isolated python environment. We are going to use conda to do that for this tutorial. These installations instructions are very similar to the the "Better long term option" of the Recommended Installation in the pvcaptest documentation. The documentation is out of date compared to this tutorial regarding the use of the Anaconda distribution.

Miniforge will install a conda, but with the default source for packages set to conda-forge.

1. Go to the Miniforge github page to download miniforge. Pick the correct installation option for

your operating system and follow the directions.

Windows:

- Use the default options and the .exe installer.

OSX / Linux:

- Open a command line (Type `command + space` to open Spotlight and type `terminal` and open your default terminal app) and copy and paste the commands and hit enter.

Installing pvcaptest

1. Open a command line, which we will use to run the conda commands required to install `pvcaptest` .

Windows: Type `miniforge` into your search bar and select the `Miniforge Prompt` app, which should be the first option.

OSX: Type `command + space` to open Spotlight and type `terminal` and open your default terminal app.

2. Now we can use conda to create a new environment with the specification in the environment.yml file, which will install pvcaptest, Jupyter, and all other dependencies required for the tutorial in a new conda environment called `pvcaptest` .

Tip: If you're new to the command line try hitting tab after beginning to type a word. That should finish the word for you and save you some time and possible troubleshooting typos.

Windows: Type the following command into the terminal and hit enter:

```
conda env create -f .\pvcpmc2024\environment.yml
```

OSX: Type the following command into the terminal and hit enter:

```
conda env create -f ./pvcpmc2024/environment.yml
```

Once the installation finishes you should see a message like the following. The installation process should hopefully not take more than 5 minutes, but it will depend on your computer specs and the internet speed.

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
```

```
#     $ conda activate pvcaptest
#
# To deactivate an active environment, use
#
#     $ conda deactivate

Retrieving notices: ...working... done
```

Go ahead and activate the environment using the command provded.

Try running this command in your terminal as a quick check that pvcaptest did install correctly:

```
python -c 'import captest; print(captest.__version__)
```

You should have verson 0.13.0 installed.

```
0.13.0
```

Congratulations! You now have pvcaptest installed in an isolated conda environment. We can move on to learning how to use pvcaptest!

Starting Jupyter

Type the following on the command line:

```
jupyter lab
```

Now you should have a browser window open that looks similar to the one that I am sharing with Jupyter lab open.

Use the file browser to navigate to the `pvpmc2024` folder and open the `pvcaptest_training_pvpmc_2024.ipynb` file.

Example Capacity Test

Now we will transition to the portion of the tutorial focused on using pvcaptest to conduct an example capacity test of a PV plant with bifacial modules. Note, the plant we are using as an example is not a real plant and the measured data was original generated from the modeled data with noise and errors added.

The modified version of the capacity test flow chart below shows how we will conduct this example test. The `load_data` and `load_pvsyst` functions are the entry points to load data and return instances of the `capdata.CapData` class. This class provides all the functionality within the red boxes to filter data, fit regressions, and calulate reporting conditions.

The `ct.capdata.captest_results_check_pvalues` function takes two instances of the `CapData` class, calculates the capacity ratio, and summarizes the results (pass / fail).



Diagram adapted from Adam Kankiewicz's presentation at PVPMC Workshop May 2023, slide 7.

Imports

The first step working with Python is to import the packages that you want to use. There are conventions for importing commonly used packages like pandas - `import pandas as pd`. This allows you to use the abbreviated `pd` to reference the modules, functions, etc. within the pandas package. We will use the `import captest as ct` convention to import the pvcaptest package. A note on names - the package you import and the package on pypi are called `captest` the name of the project and the conda package name are `pvcaptest`.



```
'0.13.0'
```

Global Test Parameters

There are some parameters that are common between the treatement of the measured and modeled (PVsyst) data, which are defined here as constants e.g. the minimum irradiance and bifaciality.

Measured Data (From SCADA / DAS Historian)

Loading Data

We begin by using the `load_data` function, which reads the file(s) specified by the `path` argument and returns an instance of the `CapData` class. In this example we will calculate reporting conditions from the measured data, so we load and filter the measured data first.

When given the path to a file, as shown here, `load_data` will try to read that file. If you pass a path to a directory, `load_data` will look for and attempt to load all files ending with '.csv' in the specified directory. Other file types can also be loaded by passing your own function to the `file_reader` argument and including the extension (e.g. 'xlsx') as a kwarg.

The `load_data` method loads the data into a pandas DataFrame, which it assigns to the `data` attribute of the `CapData` object. Here we use the pandas DataFrame `head` method to return the first three rows.

	met1_poa_refcell	met2_poa_refcell	met1_poa_pyranometer	met2_poa_pyranometer
1990-10-09 00:00:00	0.0	0.0	0.0	0.0
1990-10-09 00:05:00	0.0	0.0	0.0	0.0
1990-10-09 00:10:00	0.0	0.0	0.0	0.0

3 rows × 29 columns

Column Grouping

In addition to loading data, by default the `load_data` function attempts to parse the column headers and group the columns based on the type of measurement recorded in each column. For each inferred measurement type, `group_columns` creates an abbreviated name and a list of columns that contain measurements of that type. The python dictionary created by `group_columns` is stored in the `column_groups.data` attribute. `column_groups` is a dictionary that display nicely and includes the groups as attibutes for easy access as shown below. If the column grouping returned is not correct, you can provide either your own function to group the columns or a yaml, json, or excel file mapping column group identifiers to the column headings.

Creating a column_groups.xlsx file

```
['met1_poa_pyranometer', 'met2_poa_pyranometer']
```

We will go through a quick example of one way to create a column grouping file:

1. Set the `column_groups_template` kwarg to `True` . This will create a file called `column_groups.xlsx` in the directory you have passed to the `path` argument. In this case that is the `./data/` directory.
2. Open the `column_groups.xlsx` file, which will contain all the column headings. Make sure the column headings are sorted into the groups that you want and add identifiers, which will be the `column_groups` keys when loaded. Note, pvcaptest attempts to sort these for you, but it is not foolproof, so review the headings carefully!
3. Set the `column_groups_template` kwarg to `False` or remove it and set the `group_columns` kwarg to the path to the new `column_groups.xlsx` file.
4. Rerun `ct.load_data` to load the new column grouping from the file.

Site data for Cleary Sky Irradiance from pvlib

pvcaptest will use pvlib to generate modeled clear sky POA and GHI irradiance for to compare to the measured values if you pass the necessary data describing the location and racking of the project. The call to `load_data` above demonstrates two different ways to provide site data: as a python dictionary or as a path to a yaml file.

You'll see in the plotting dashboard below, that modeled clear sky poa irradiance (`poa_mod_csky`) and ghi irradiance (`ghi_mod_csky`) are added to the `data` attribute and available to plot and compare to measured values by default when site data is provded.

CapData Selection Methods - loc & floc

The CapData has the methods `loc` and `floc` to select subsets of columns from the `data` and `data_filtered` DataFrames, respectively. These methods allow easy access to the groups of columns identified in `column_groups` using the `column_group` keys, column names, `regression_cols` keys, or a combination of the three. The `regression_cols` attribute is introduced below. The `column_groups` dictionary also enables much of the functionality of `CapData` methods to perform common capacity testing tasks, like generating scatter plots, filtering data, and performing regressions.

Using the `loc` method with the 'irr_poa_ref_cell' attribute key of `column_groups` to select data from the POA reference cell columns in the `data` DataFrame:

	met1_poa_refcell	met2_poa_refcell
1990-10-09 08:20:00	534.845280	534.812723
1990-10-09 08:25:00	562.246349	553.728808
1990-10-09 08:30:00	539.686886	440.947099

Accessing the two irradiance columns of the `irr_poa_pyran` group, the single column of the `real_pwr_mtr` group, and the `met1_amb_temp` column of the `data_filtered` DataFrame:

	met1_poa_pyranometer	met2_poa_pyranometer	meter_power	met1_amb_temp
1990-10-09 00:00:00	0.0	0.0	-8868.0	17.750666
1990-10-09 00:05:00	0.0	0.0	-8868.0	17.737545
1990-10-09 00:10:00	0.0	0.0	-8868.0	17.648090

The `loc` and `floc` methods can also be used to access the columns that are used to fit the regression by passing the special string `regcols` . However, this requires specifying the regression columns first, which is discussed below in Setting Regression Columns

Bifacial Testing, Calculating E_{Total} / Markdown Cells for Descriptions and Latex for Math

To conduct a bifacial capacity testing following the guidelines of the NREL conference paper, Suggested Modifications for Bifacial Capacity Testing, we must calculate the Total Plane of Array Irradiance (E_{TOTAL}):

$$E_{TOTAL} = E_{POA} + E_{REAR} * \varphi$$

where,

- E_{POA} - Measured front side POA
- E_{REAR} - Measured rear side POA
- φ - Bifaciality factor as defined on the manufacturer's datasheet

This section also demontrates how you can use latex math within markdown cells to write nicely formatted equations.

We have the bifaciality from the global test parameters defined at the beginning of the test.

```
bifaciality: 0.71
```

In the next line we calculate E_{TOTAL} by using the `loc` method (Pandas DataFrame NOT pvcaptest CapData) to select the front and rear plane of array irradiance, take the mean (axis=1 indicates across rows), multiply the rear POA irradiance by the bifaciality, and store the result as a new column labeled as `irr_poa_tot`.

Running the `reset_filter` method will replace the `data_filtered` DataFrame with the `data` DataFrame, which is important to be sure that the new `irr_poa_tot` column is added to the `data_filtered` DataFrame.

Setting Regression Columns

pvcaptest does not attempt to determine which columns of data or groups of columns are the data to be used in the regressions. The link between regression variables and the imported data is made by a dictionary stored in the `regression_cols` attribute. pvcaptest provides the convience method `set_regression_cols` for this purpose. `regression_cols` should be set soon after loading data as many other `CapData` methods rely on this attribute.

The keys of `regression_cols` should be the terms used in the `regression_formula` and the values may point to either column group ids (keys of `column_groups`) or to column labels of `data_filtered`. For this test we set the `power`, `t_amb`, and `w_vel` terms equal to the `column_group` ids pointing to these groups of measurements and the `poa` term to the `irr_poa_tot` column of `data_filtered` we calculated above.

After running `set_regression_cols` we can check the `regression_cols` attribute to view the dictionary and confirm we have the mapping from the regression terms to the correct column or groups of columns. You can also directly assign a dictionary to `regression_cols` rather than using `set_regression_cols`.

```
{'power': 'real_pwr_mtr',
 'poa': 'irr_poa_tot',
 't_amb': 'temp_amb',
 'w_vel': 'wind_speed'}
```

The regression formula by default is set to match the standard, which we can use without any adjustment for this test. The `regression_formula` attribute can be viewed to check the current value for the regression formula or assigned to use a different regression formula. See the statsmodels documentation for more information on how to specify regression formulas.

The `I(poa * poa)` notation allows statsmodels and Patsy to hanlde the multiplication of the terms for us.

```
'power ~ poa + I(poa * poa) + I(poa * t_amb) + I(poa * w_vel) - 1'
```

Once the regression columns are set, the `loc` or `floc` methods will return the data each of the regression terms is mapped to. **Note, in v0.13.0 the `view` and `rview` methods have been replaced by `loc` and `floc`.**

Here we are accessing the same POA irradiance data as above with `loc` and the group name, but now using the regression variable id.

	irr_poa_tot
1990-10-09 08:20:00	615.491579
1990-10-09 08:25:00	638.425862
1990-10-09 08:30:00	593.252978

Accessing the data from `data_filtered` with `floc`:

	irr_poa_tot
1990-10-09 08:20:00	615.491579
1990-10-09 08:25:00	638.425862
1990-10-09 08:30:00	593.252978

As mentioned earlier now that the `regression_cols` attribute is set, you can use the special string `regcols` to access the data that will be regressed against.

	meter_power	irr_poa_tot	met1_amb_temp	met2_amb_temp	met1_windspeed	met
1990-10-09 00:00:00	-8868.0	0.0	17.750666	17.770821	-0.000721	
1990-10-09 00:05:00	-8868.0	0.0	17.737545	17.753030	-0.008876	
1990-10-09 00:10:00	-8868.0	0.0	17.648090	17.689437	0.008686	
1990-10-09 00:15:00	-8868.0	0.0	17.641772	17.582119	0.000598	
1990-10-09 00:20:00	-8868.0	0.0	17.616870	17.555746	-0.009093	
...	...	...	...	...	...	
1990-10-13 23:35:00	-8868.0	0.0	20.037996	20.032399	-0.008095	
1990-10-13 23:40:00	-8868.0	0.0	19.893458	19.862932	-0.002012	
1990-10-13 23:45:00	-8868.0	0.0	19.750967	19.771194	-0.000650	
1990-10-13 23:50:00	-8868.0	0.0	19.585205	19.646627	0.004027	
1990-10-13 23:55:00	-8868.0	0.0	19.459186	19.498768	-0.006371	

1440 rows × 6 columns

Aggregating Measurements in the Same Group

One of the first conviences provided by pvcaptest after defining `regression_cols` and `column_groups` is the ability to easily aggregate the groups of measurements defined by `column_groups`. For example, typically a larger system will have multiple front POA irradiance sensors and the measurements from each sensor will be averaged for each time interval. The `agg_sensors` method of `CapData` allows you to specify the type of aggregation to be applied to each group and the combined terms will be added to `data` and `data_filtered`.

```
Regression variable 't_amb' has been remapped: 'temp_amb' to 'temp_amb_mean_agg'
Regression variable 'w_vel' has been remapped: 'wind_speed' to 'wind_speed_mean_agg'
```

	irr_poa_mean_agg	real_pwr_inv_sum_agg	temp_amb_mean_agg	wind_speed_mean
1990-10-09 00:00:00	0.0	-1200.0	17.760744	-0.0(
1990-10-09 00:05:00	0.0	-1200.0	17.745288	-0.00

2 rows × 35 columns

For any regression terms in `regression_cols` that are mapped to `column_groups` ids and the group is included in those passed to `agg_sensors` the column group id will be replaced by the new aggregated column name. This update to `regression_cols` is printed when running `agg_sensors` as shown above where the `t_amb` regression term has been adjusted to map to the new aggregated column `temp_amb_mean_agg` rather than the column group `temp_amb`:

```
{'power': 'real_pwr_mtr',
 'poa': 'irr_poa_tot',
 't_amb': 'temp_amb_mean_agg',
 'w_vel': 'wind_speed_mean_agg'}
```

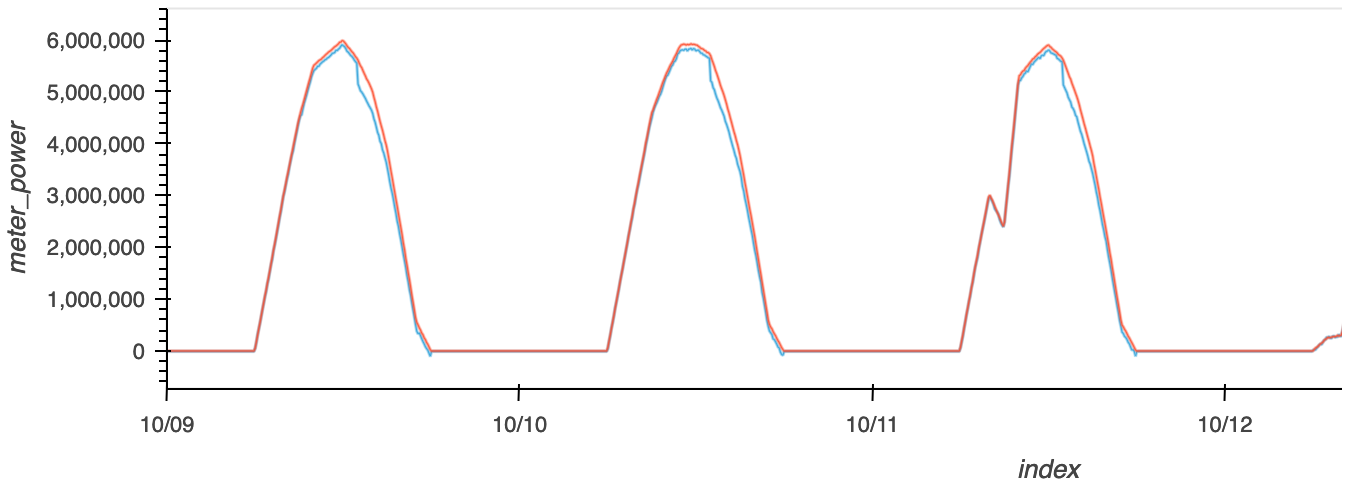
Plotting Dashboard

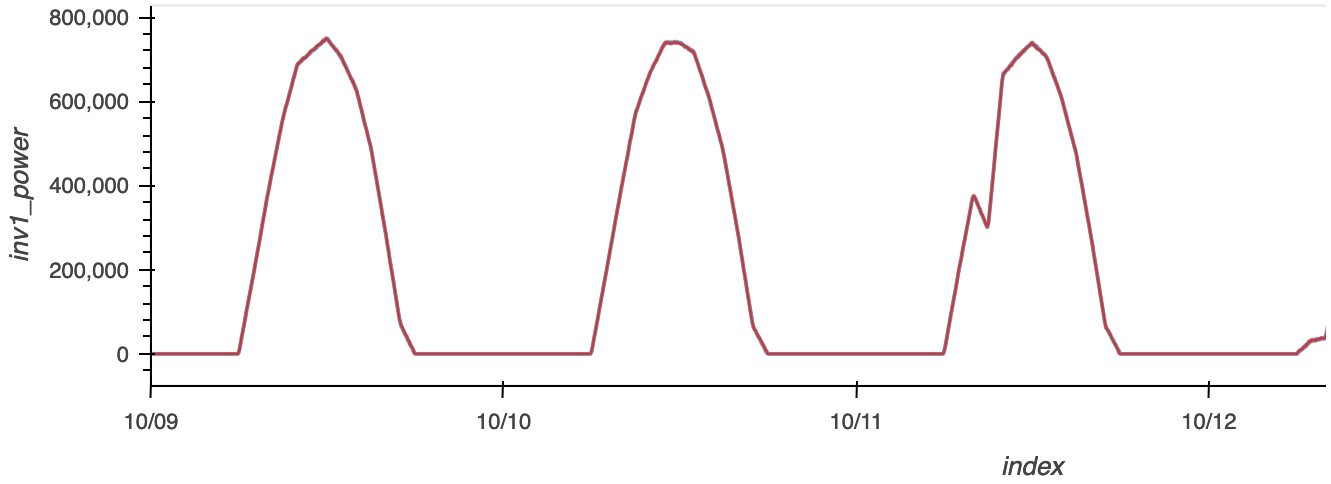
Note, the original `plot` method has been replaced in v0.13.0 and now returns a dashboard to visualizations of the `data`.

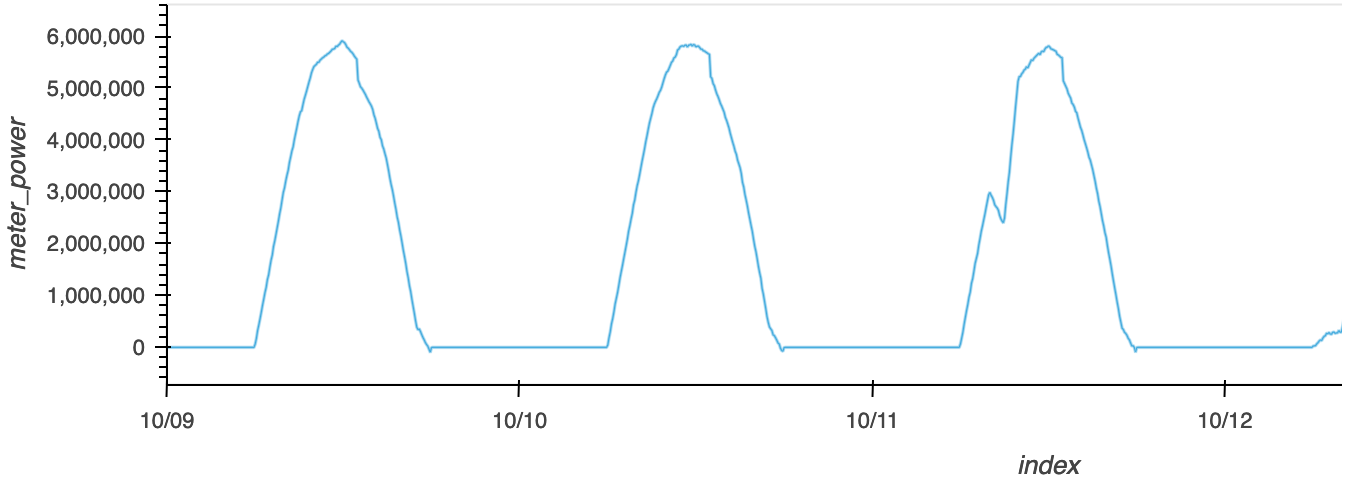
The plotting dashboard provides functionality to explore timeseries plots of the columns in `data` or of the groups of columns defined by `column_groups`. The `Groups` tab provides a default view of a few of the primary groups of data used in capacity testing. The `Layout` tab provides a way to select the groups you would like to view arranged in a column. The `overlay` tab provides a way to overlay groups or selections of columns on a single timeseries plot. The `Scatter` tab allows selecting any two columns of `data` to view in a scatter plot.

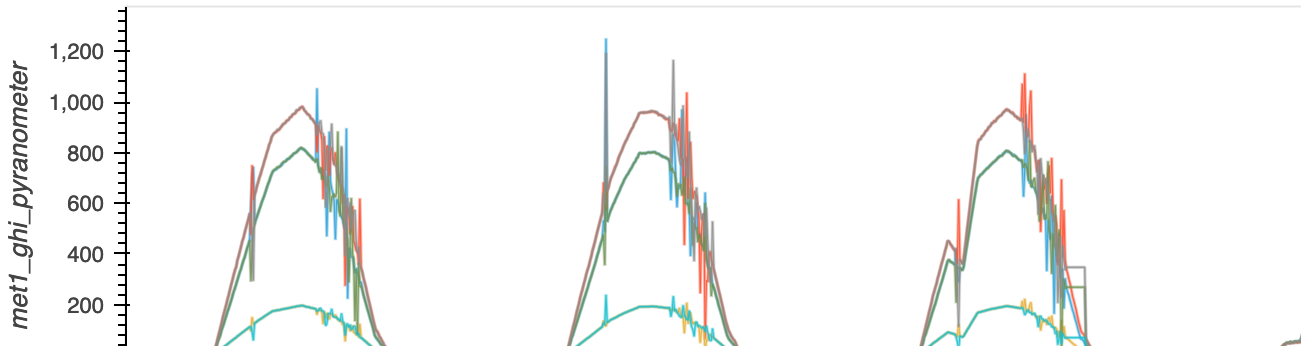
The GUI can be used to build up custom views to display in the `Groups` tab or you can pass regular expressions to the `combine` and `default_groups` key word arguments (kwargs) to customize the `Groups` tab.

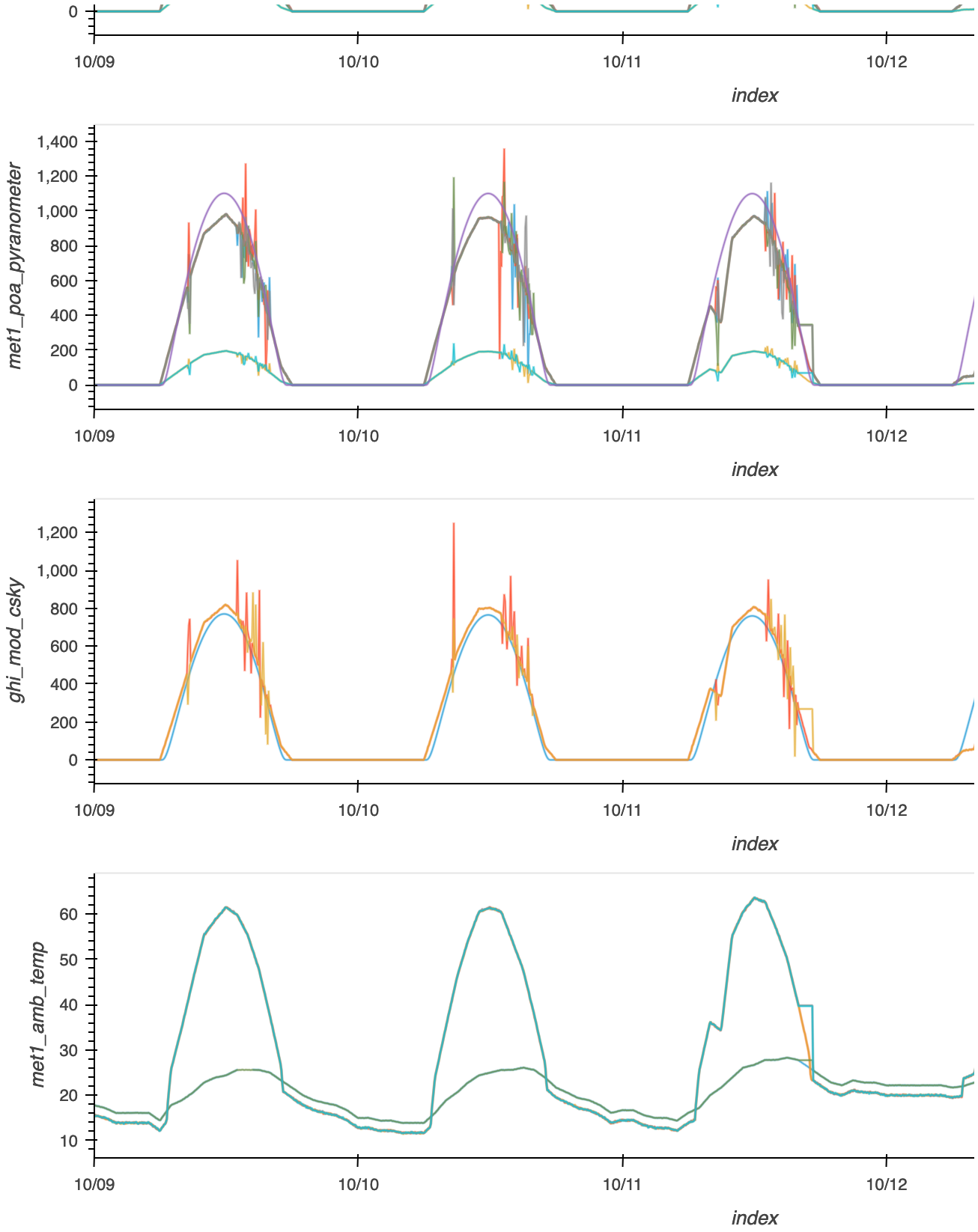
Groups Layout Overlay Scatter


meter_power
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
10/09
10/10
10/11
10/12
index


inv1_power
800,000
600,000
400,000
200,000
0
10/09
10/10
10/11
10/12
index


meter_power
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
10/09
10/10
10/11
10/12
index


met1_ghi_pyranometer
1,200
1,000
800
600
400
200

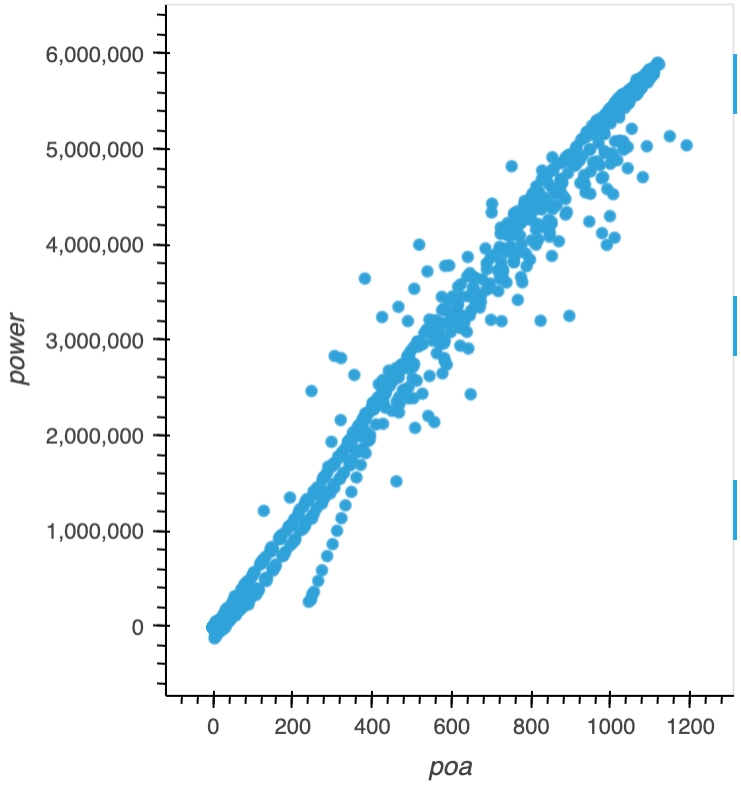


Measured Data - Filtering, Reporting Conditions Calculation, and Regression

The `CapData` class provides a number of convience methods to apply filtering steps as defined in ASTM E2848 or the contractual capacity test. The following section demonstrates the use of the

more commonly used filtering steps to remove measured data points.

A common first step is to review the scatter plot of the POA irradiance against the power production, which we can do with the `scatter_hv` method.



Unstable irradiance filtering and Defining and using Custom Filters

Looking at the plot above there is clearly some time periods of scatter due to unstable irradiance. pvcaptest includes the method `filter_clearsky` which wraps the pvlib function for clear sky filtering. This filter is strict and some contracts require different approaches or allowance for some unstable irradiance. Currently, there are not other methods for filtering out periods of unstable irradiance, but the pvcaptest does make it easy to adapt to this type of requirements or other contract specific filtering requirements.

The `filter_custom` method provides a way to use your own filtering method within captest and update the summary data. The `filter_custom` method allows passing any function or method that takes a DataFrame as the first argument and returns the dataframe with rows removed. Passed methods can be user-defined functions or many Pandas DataFrame methods will also work.

Here we will use a simple filtering function relying on Pandas functionality to remove periods of unstable irradiance.

And here we are using the `unstable_irr_filter` with the `filter_custom` method. In Python a function is an object and can be passed to another function, which is what we do with the

`filter_custom` function. The first argument to pass is the custom function you have defined, in this case `unstable_irr_filter`. That is followed by passing the arguments (args) and / or key word arguments (kwargs) required by `unstable_irr_filter`.

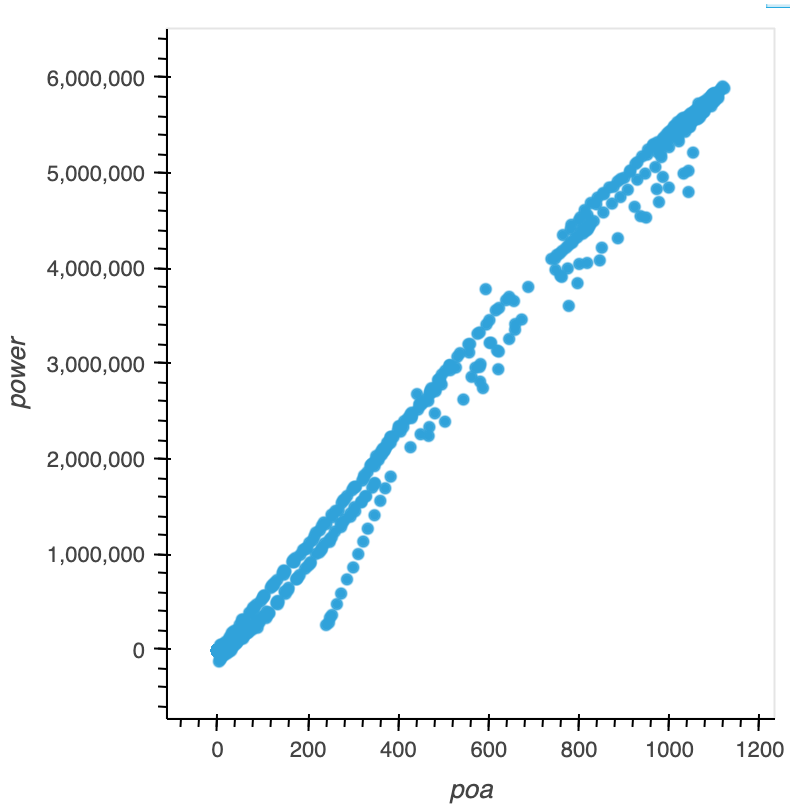
You do not need to pass the `data` argument; `filter_custom` does this for you.

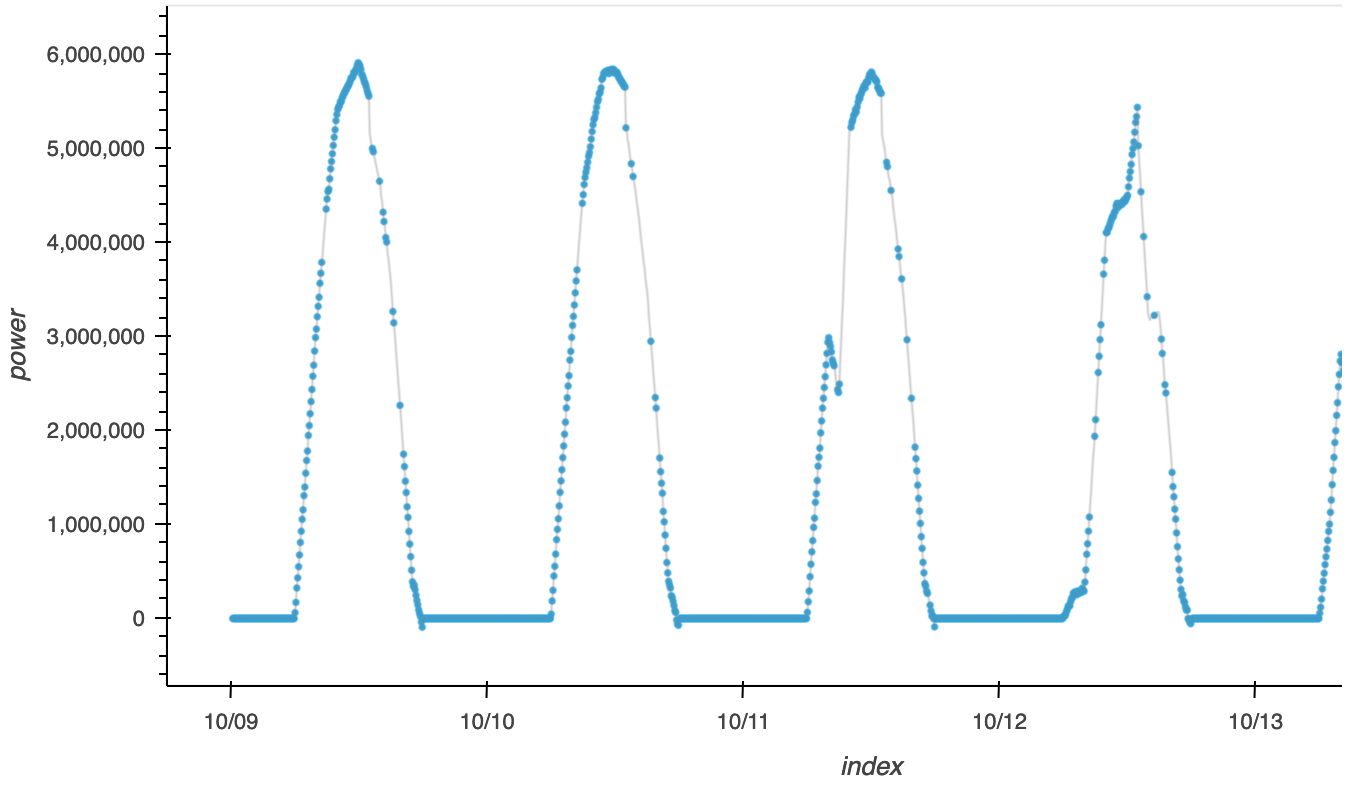
Using this approach pvcaptest will treat your custom filter the same as all other filters including it in the summary table and the plots of the filtering.

The `get_summary` method will return a dataframe summarizing the filtering steps that have been applied, the agruments passed to them, the number of points prior to filtering, and the number of points after filtering. It can be helpful when using `custom_filter` to pass all the arguments as kwargs so they are included in the summary table.

		pts_after_filter	pts_removed	filter_arguments
meas	**filter_custom**	1234	206	unstable_irr_filter, , irr_column: irr_poa_mean_agg, window: 3, threshold: 28

Plotting the data again we can see this filter was effective at removing the obvious scatter. This time we set the kwarg `timeseries` to True to add a timeseries plot of power that is linked to the scatter plot. Selecting data in the scatter plot will highlight the same points in the timeseries.





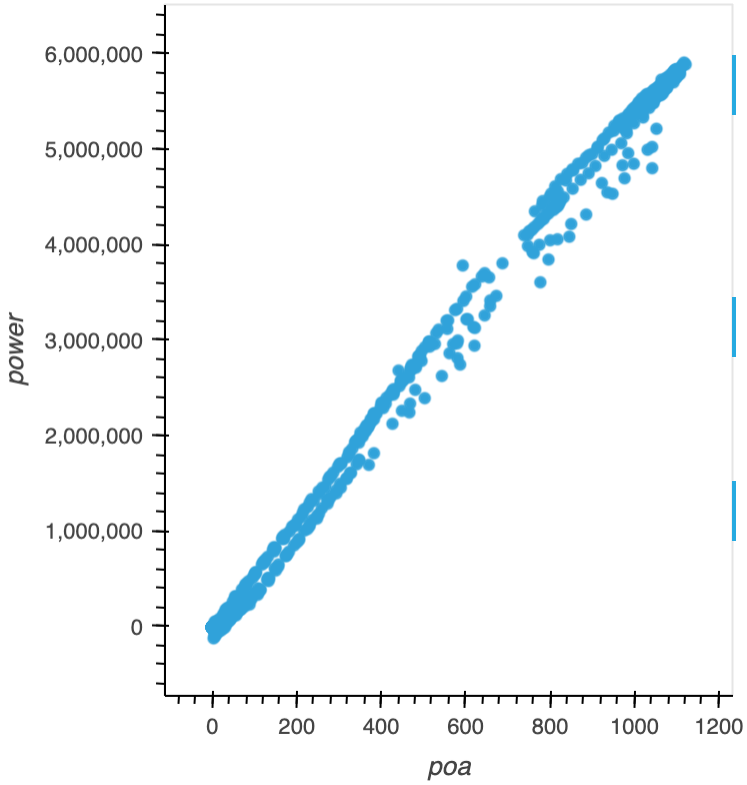
Removing a time period

After removing the periods of unstable irradiance there is a very visible trend at roughly 200 to 400 W/m^2 that does not follow the primary linear trend. We can use the linked scatter plot and and timeseries above to zoom in and select these points in the scatter plot and identify them in the

linked timeseries plot. The `filter_time` method can be used to select or remove specific time periods. Below we use it remove the period of time identified in the plot.

Rerunning the `scatter_hv` and `get_summary` methods will confirm the filter removed the time periods we intended to remove.

		pts_after_filter	pts_removed	filter_arguments
meas	**filter_custom**	1234	206	unstable_irr_filter, , irr_column: irr_poa_mean_agg, window: 3, threshold: 28
	filter_time	1221	13	start: 10/11/1990 16:15, end: 10/11/1990 17:15, drop: True

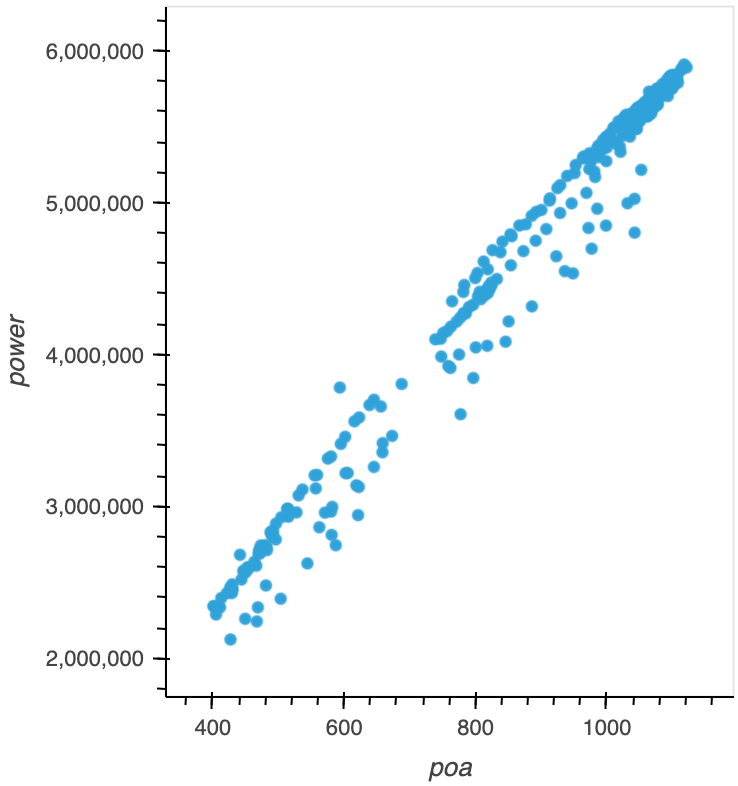


Filtering Irradiance

The `filter_irr` method provides a convient way to remove remove data based on the irradiance measurments. Here we use it to remove periods of low irradiance and high irradiance.

```
Min irr: 400 and Max irr: 1400 set at beginning of test.
```

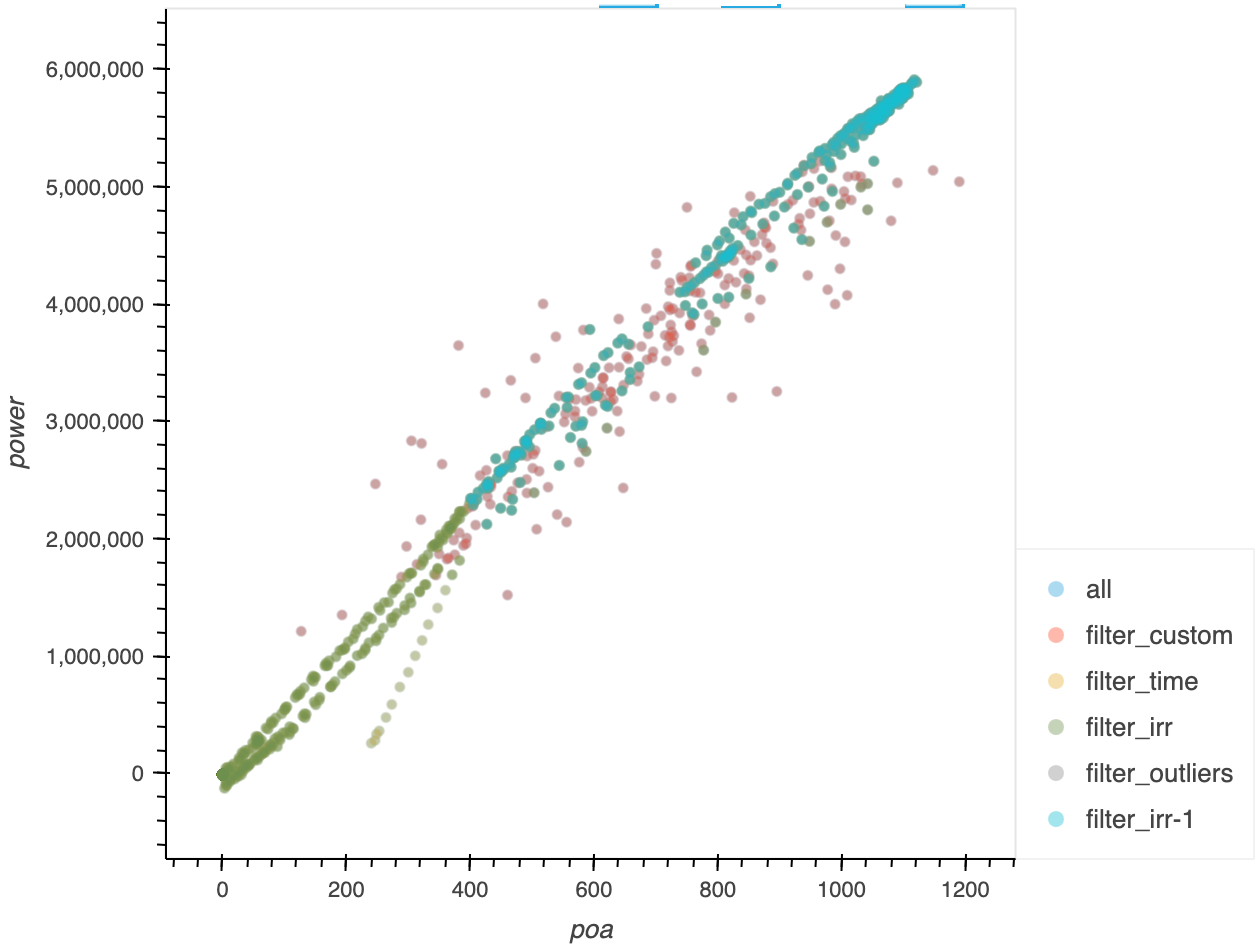
We can re-run the `scatter` method to see the results of the filtering steps.



Filtering Outliers

The `filter_outliers` method uses scikit-learn's elliptic envelope to remove outlier points. This filter is intended to be an automated and repeatable version of the visual filtering step defined in the standard. The default contamination setting is set to remove points that appear as obvious outliers when visually inspecting the scatter plot, but can be adjusted.

This time to review the scatter plot we are using the `scatter_filters` method, which overlays scatter plots of the filtering at teach time step, so it is possible to see which points where removed by which filter.



		pts_after_filter	pts_removed	filter_arguments
meas	**filter_custom**	1234	206	unstable_irr_filter, , irr_column: irr_poa_mean_agg, window: 3, threshold: 28
	filter_time	1221	13	start: 10/11/1990 16:15, end: 10/11/1990 17:15, drop: True
	filter_irr	283	938	400, 1400,
	filter_outliers	271	12	Default arguments
	filter_irr-1	271	0	0, 1500,

Preliminary Regression Filtering (ASTM E28248 9.1.3)

> Averaging intervals for which the residual exceeds two standard deviations of the mean residual should be

investigated and may be excluded if they do not meet the filter criteria.

The `fit_regression` method performs a regression on the data stored in `data_filtered` using the regression equation specified by the standard. The regression equation is stored in the

`regression_formula` attribute as shown below. Regressions are performed using the statsmodels package.

Below, we set the filter argument of the `fit_regression` method to `True` to remove time periods when the residual exceeds two standard deviations of the mean residual.

```
'power ~ poa + I(poa * poa) + I(poa * t_amb) + I(poa * w_vel) - 1'
```

```
NOTE: Regression used to filter outlying points.
```

		pts_after_filter	pts_removed	filter_arguments
meas	**filter_custom**	1234	206	unstable_irr_filter, , irr_column: irr_poa_mean_agg, window: 3, threshold: 28
	filter_time	1221	13	start: 10/11/1990 16:15, end: 10/11/1990 17:15, drop: True
	filter_irr	283	938	400, 1400,
	filter_outliers	271	12	Default arguments
	filter_irr-1	271	0	0, 1500,
	fit_regression	253	18	filter: True, summary: False

Calculation of Reporting Conditions

The `rep_cond` method provide a variety of ways to calculate reporting conditions. Using `rep_cond` the reporting conditions are always calculated from the data store in the `data_filtered` attribute. Refer to the example notebook Reporting Conditions Examples for a thourough explanation of the `rep_cond` functionality. By default the reporting conditions are calcualted following the guidance of ASTM E2939-13 - 60th percentile irradiance, mean ambient temperature, and mean wind speed.

The `rep_cond` method prints out the results when run:

```
Reporting conditions saved to rc attribute.
          poa      t_amb     w_vel
0  1020.5656  24.365916  2.069346
```

...and it stores them in the `rc` attribute as a DataFrame:

	poa	t_amb	w_vel
0	1020.5656	24.365916	2.069346

Filtering Irradiance Based on Reporting Irradiance

Previously we used the irradiance filter to filter out data below 400 W/m^2. The irradiance filter can also be used to filter irradiance based on a percentage band around a reference value. This approach is shown here to remove data where the irradiance is outside of +/- 20% of the reporting irradiance.

```
Min irradiance is 0.8 of the reporting irradiance
```

```
Max irradiance is 1.2 of the reporting irradiance
```

Final Regression Fit

The `fit_regression` method is used again with the default arguments, which result in fitting the regression, printing and storing the results, but not filtering. The result of the regression is a statsmodels `RegressionResultsWrapper` object containing the regression coefficients and other information generated when performing the regression. This object is stored in the CapData `regression_results` attribute.

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                   power   R-squared (uncentered):                   1.0
00
Model:                             OLS   Adj. R-squared (uncentered):              1.0
00
Method:                  Least Squares   F-statistic:                          2.713e+
06
Date:                Thu, 09 May 2024   Prob (F-statistic):                        0.
00
Time:                        14:14:26   Log-Likelihood:                         -180
5.4
No. Observations:                 159   AIC:                                      361
9.
Df Residuals:                     155   BIC:                                      363
1.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
poa             6898.7847     29.510    233.781      0.000    6840.492    6957.078
I(poa * poa)      -0.7498      0.023    -33.298      0.000      -0.794      -0.705
I(poa * t_amb)   -31.5430      0.984    -32.058      0.000     -33.487     -29.599
I(poa * w_vel)     3.7261      1.525      2.443      0.016       0.713       6.739
==============================================================================
Omnibus:                        0.898   Durbin-Watson:                   1.206
Prob(Omnibus):                  0.638   Jarque-Bera (JB):                0.667
Skew:                          -0.154   Prob(JB):                        0.716
Kurtosis:                       3.080   Cond. No.                     1.89e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[3] The condition number is large, 1.89e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The regression coefficients and p-values for each term are attributes available in the `regression_results`, but for a typical capacity test we do not need to access them this way.

```
poa               6898.784679
I(poa * poa)        -0.749795
I(poa * t_amb)     -31.542997
I(poa * w_vel)       3.726091
dtype: float64
```

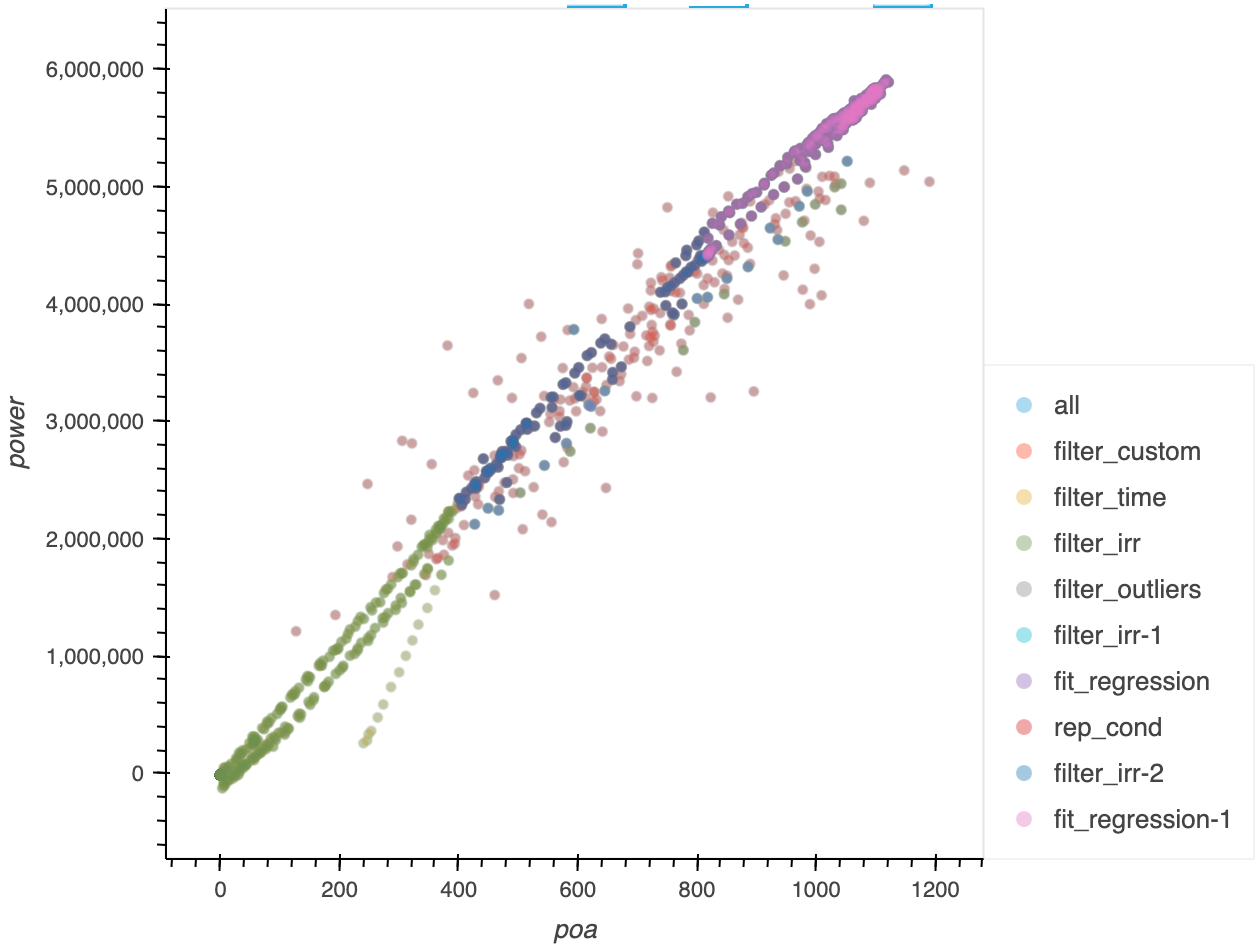
```
poa               1.979745e-199
I(poa * poa)       1.603419e-72
I(poa * t_amb)     2.747470e-70
I(poa * w_vel)     1.568572e-02
dtype: float64
```

Review of Measured Data Filtering

Filtering summary table:

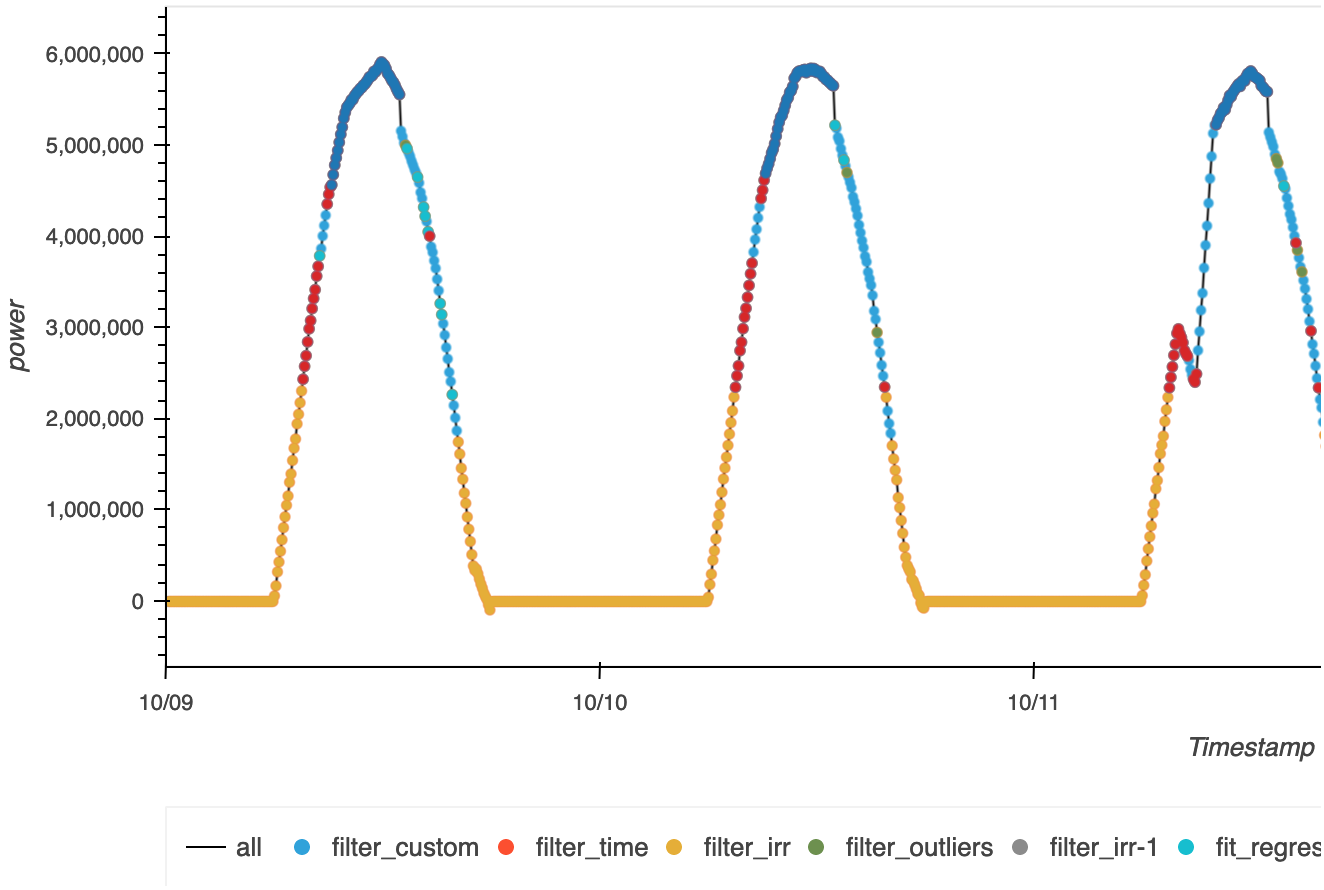
		pts_after_filter	pts_removed	filter_arguments
meas	**filter_custom**	1234	206	unstable_irr_filter, , irr_column: irr_poa_mean_agg, window: 3, threshold: 28
	filter_time	1221	13	start: 10/11/1990 16:15, end: 10/11/1990 17:15, drop: True
	filter_irr	283	938	400, 1400,
	filter_outliers	271	12	Default arguments
	filter_irr-1	271	0	0, 1500,
	fit_regression	253	18	filter: True, summary: False
	rep_cond	253	0	Default arguments
	filter_irr-2	159	94	0.8, 1.2, ref_val: 1020.566
	fit_regression-1	159	0	Default arguments

Overlay of scatter plots showing the points removed by each filter:



The `timeseries_filters` method provides an additional way to visualize the filtering as an overlay of timeseries showing which intervals were removed by which filter.

The legend entries in these two plots correlate to the row labels of the summary table produced by `get_summary`. Clicking the legend entries from left to right and top to botom will show the point removed by each filter for the timeseries and scatter plot, respectively. **Note, using the legends in a different order may not result in a display that matches your expectations.**



Exporting filtering documentation

The `get_filtering_table` creates a DataFrame documenting for every time interval if the periods was removed and, if so, by which filtering step. The cell below shows this data can be combined with the `data` DataFrame to provide documentation of all the data and filtering. The resulting combined DataFrame can easily be saved to a csv or excel file to facilitate review.

	filter_custom	filter_time	filter_irr	filter_outliers	filter_irr-1	fit_regression	rep_co
1990-10-09 00:00:00	1	NaN	NaN	NaN	NaN	NaN	Na
1990-10-09 00:05:00	1	NaN	NaN	NaN	NaN	NaN	Na
1990-10-09 00:10:00	0	0.0	1.0	NaN	NaN	NaN	Na
1990-10-09 00:15:00	0	0.0	1.0	NaN	NaN	NaN	Na
1990-10-09 00:20:00	0	0.0	1.0	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	
1990-10-13 23:35:00	0	0.0	1.0	NaN	NaN	NaN	Na
1990-10-13 23:40:00	0	0.0	1.0	NaN	NaN	NaN	Na
1990-10-13 23:45:00	0	0.0	1.0	NaN	NaN	NaN	Na
1990-10-13 23:50:00	0	0.0	1.0	NaN	NaN	NaN	Na
1990-10-13 23:55:00	0	0.0	1.0	NaN	NaN	NaN	Na

1440 rows × 45 columns

PVsyst Data Loading, Filtering, and Regression

Essentially the same process is repeated to load, filter, and regress the energy model data. This section will show the steps in a much more condensed format using fewer notebook cells.

Loading PVsyst Data

After loading the PVsyst data, the Total Plane of Array Irradiance (E_{TOTAL}) was calculated per the following Equation:

$$E_{TOTAL} = GlobInc + (GlobBak + BackShd) * \varphi$$

where,

- $GlobInc$ - Measured front side POA
- $GlobBak$ - Measured rear side POA
- $BackShd$ - Irradiance lost due to rear shading

- φ - Bifaciality factor as defined on the manufacturer's datasheet

In the supporting data exports E_{TOTAL} is labeled as `irr_poa_tot` .

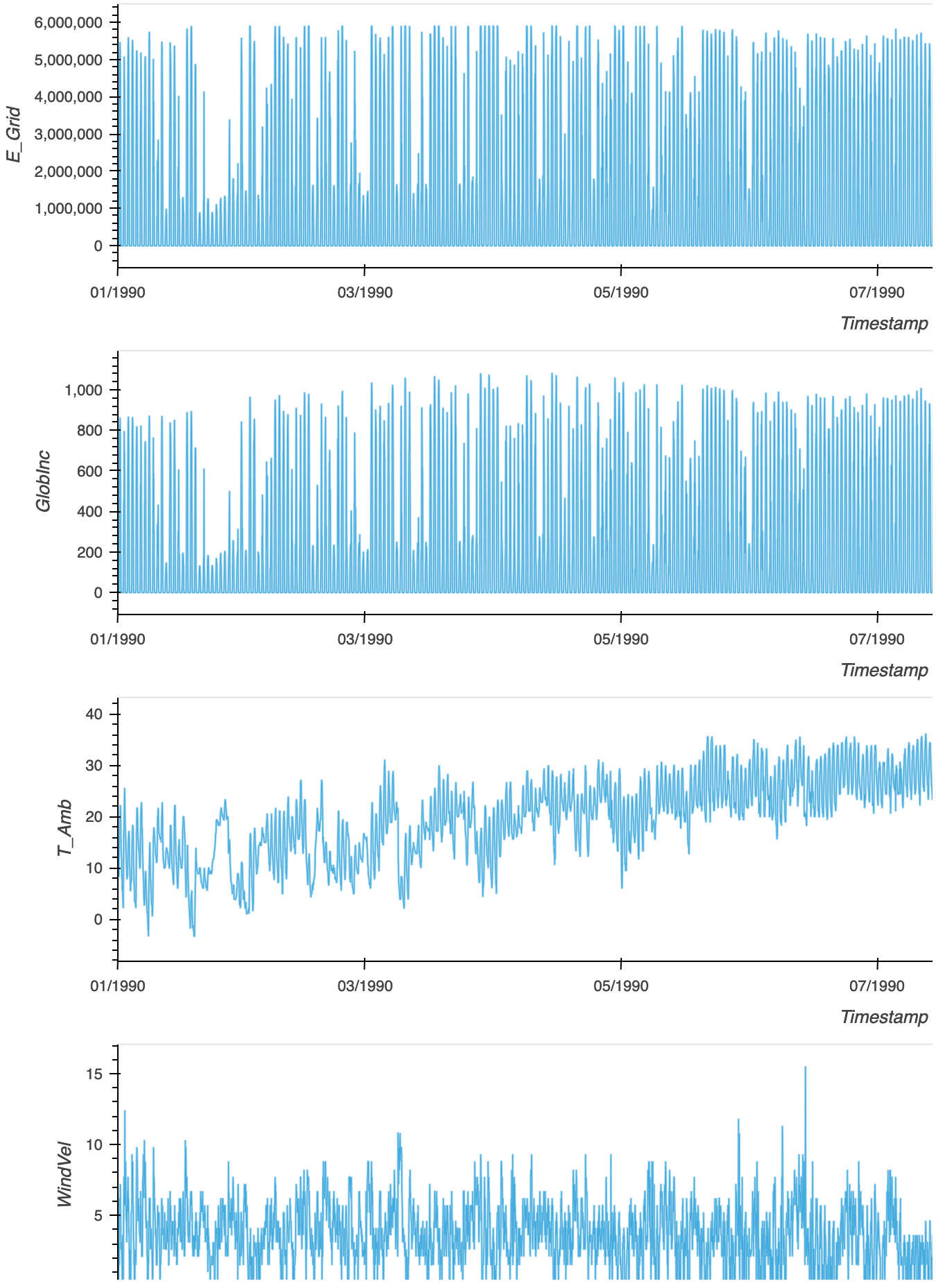
The annual energy of the 8760 for comparison to the .csv and PVsyst pdf report:

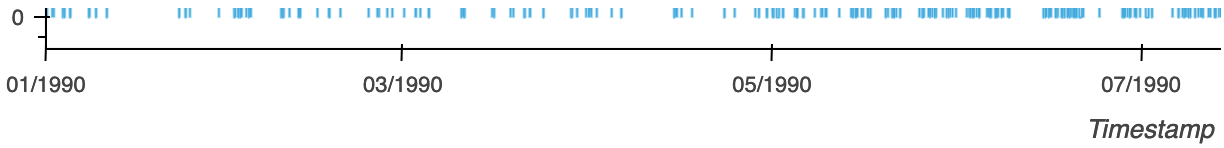
```
Maximum E_Grid: 5,898,420
```

```
Sum of E_Grid: 11,062,732,133
```

The PVsyst data before any filtering is plotted below.

Groups Layout Overlay Scatter


E_Grid
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
01/1990
03/1990
05/1990
07/1990
Timestamp
GlobInc
1,000
800
600
400
200
0
01/1990
03/1990
05/1990
07/1990
Timestamp
T_Amb
40
30
20
10
0
01/1990
03/1990
05/1990
07/1990
Timestamp
WindVel
15
10
5



Filter and Regress PVsyst Data

The regression terms (power, poa, t_amb, w_vel) are mapped to the following columns of the data. These are the columns used to fit the regression.

```
power:  E_Grid
poa:  irr_poa_tot
t_amb:  T_Amb
w_vel:  WindVel
```

Here we calculate the middle of the measured data test period to use for filtering the energy model data. This is more useful with real data over longer time periods :)

Filtering the PVsyst data

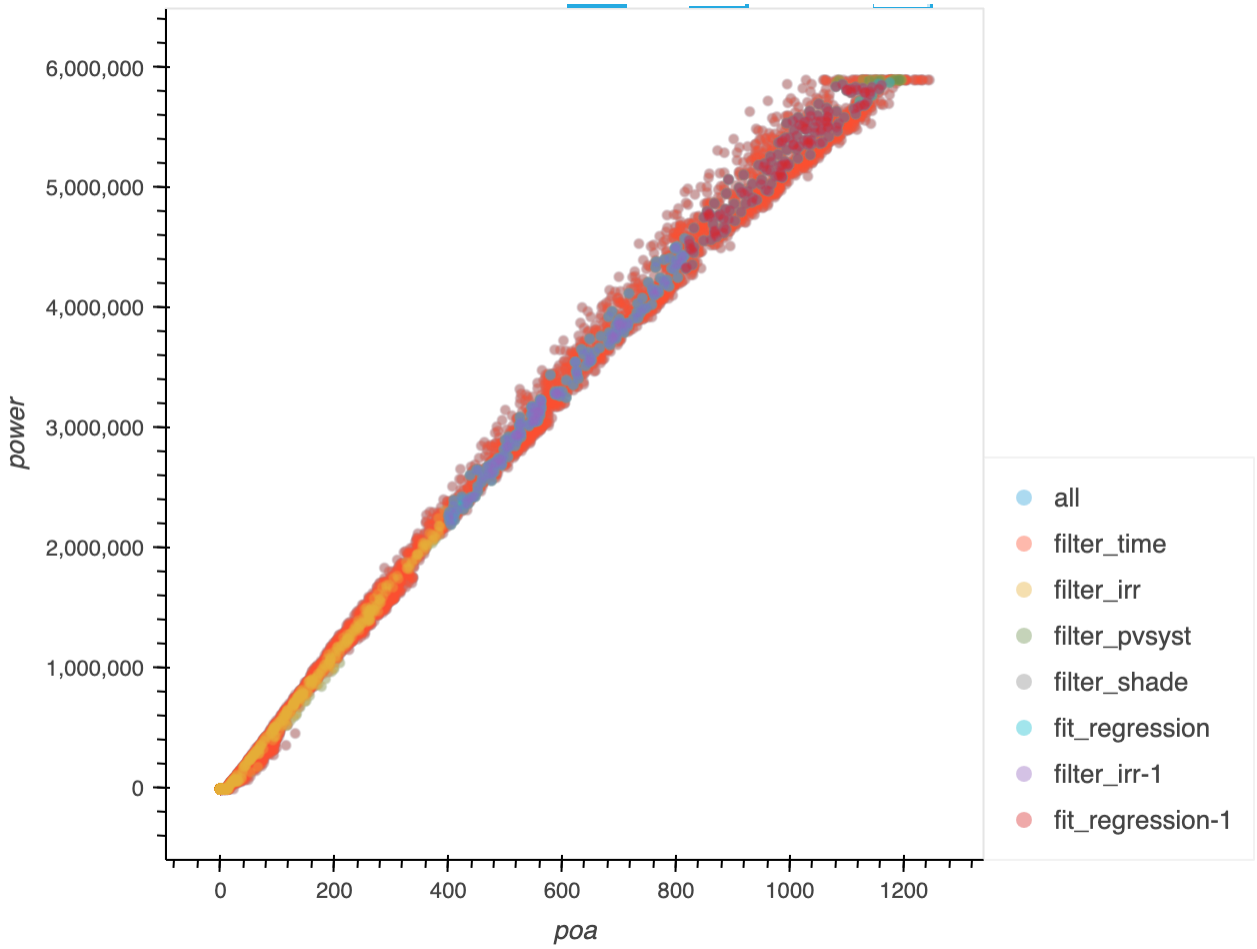
```
NOTE: Regression used to filter outlying points.

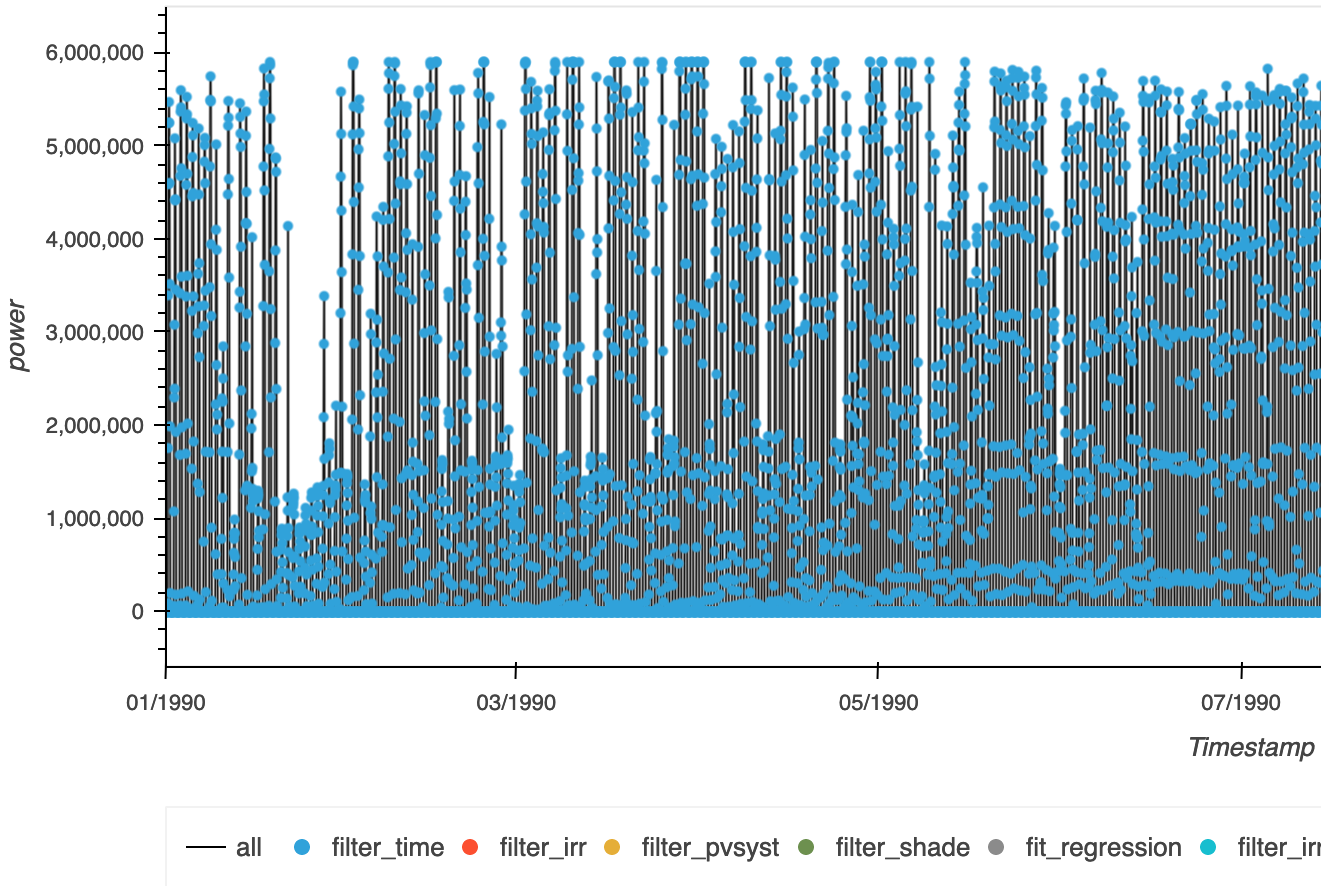

                                 OLS Regression Results
==============================================================================
==
Dep. Variable:                   power   R-squared (uncentered):                   1.0
00
Model:                             OLS   Adj. R-squared (uncentered):              1.0
00
Method:                  Least Squares   F-statistic:                          4.956e+
06
Date:                Thu, 09 May 2024   Prob (F-statistic):                        0.
00
Time:                        14:14:27   Log-Likelihood:                          -164
7.4
No. Observations:                 150   AIC:                                       330
3.
Df Residuals:                     146   BIC:                                       331
5.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
poa             6908.7200     14.610    472.868      0.000    6879.845    6937.595
I(poa * poa)      -0.8488      0.014    -62.545      0.000      -0.876      -0.822
I(poa * t_amb)   -28.5018      0.315    -90.446      0.000     -29.125     -27.879
I(poa * w_vel)    -0.3935      0.635     -0.619      0.537      -1.649       0.862
==============================================================================
Omnibus:                        5.233   Durbin-Watson:                   1.204
Prob(Omnibus):                  0.073   Jarque-Bera (JB):                5.233
Skew:                          -0.456   Prob(JB):                       0.0731
Kurtosis:                       2.927   Cond. No.                     1.24e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[3] The condition number is large, 1.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Review of Energy Model Filtering

		pts_after_filter	pts_removed	filter_arguments
pvsyst	**filter_time**	1440	7320	test_date: 1990-10-11 10:37, days: 60
	filter_irr	353	1087	400, 1200,
	filter_pvsyst	336	17	Default arguments
	filter_shade	336	0	Default arguments
	fit_regression	325	11	filter: True, summary: False
	filter_irr-1	150	175	0.8, 1.2, ref_val: 1020.566
	fit_regression-1	150	0	filter: False, summary: True





Detailed Results

Once the filtering, regressions, and calculation of the reporting conditions is complete for the measured and modeled data are complete, the final step is to calculate the results. The `capdata` module includes a number of functions that accept two `CapData` objects as arguments and return summary or results.

The `get_summary` function (not to be confused with the `CapData.get_summary` method simplifies displaying all the filtering performed on the measured and modeled data.

		pts_after_filter	pts_removed	filter_arguments
meas	**filter_custom**	1234	206	unstable_irr_filter, , irr_column: irr_poa_mean_agg, window: 3, threshold: 28
	filter_time	1221	13	start: 10/11/1990 16:15, end: 10/11/1990 17:15, drop: True
	filter_irr	283	938	400, 1400,
	filter_outliers	271	12	Default arguments
	filter_irr-1	271	0	0, 1500,
	fit_regression	253	18	filter: True, summary: False
	rep_cond	253	0	Default arguments
	filter_irr-2	159	94	0.8, 1.2, ref_val: 1020.566
	fit_regression-1	159	0	Default arguments
pvsyst	**filter_time**	1440	7320	test_date: 1990-10-11 10:37, days: 60
	filter_irr	353	1087	400, 1200,
	filter_pvsyst	336	17	Default arguments
	filter_shade	336	0	Default arguments
	fit_regression	325	11	filter: True, summary: False
	filter_irr-1	150	175	0.8, 1.2, ref_val: 1020.566
	fit_regression-1	150	0	filter: False, summary: True

The following method provides a summary of the data intervals remaining compared against any contractual requirement. The required quantity of data points should be provided in equivalent operating hours eg (750) 1-minute intervals is equivalent to 12.5 hours of operating time.

```
length of test period to date: 5 days
sufficient points have been collected. 150.0 points required; 159 points collected
```

The Reporting Conditions can be displayed again by simply accessing the `rc` attribute of the `CapData` instance used to calculate them.

	poa	t_amb	w_vel
0	1020.5656	24.365916	2.069346

Capacity Test Results

We use the `captest_results_check_pvalues` method to display a final summary of the test results.

This function will display the results with and without the check of the regression p values described in section 9.2 of the standard and will also display the regression coefficients (parameters) and p values for both regressions.

```
Using reporting conditions from das.

Capacity Test Result:         PASS
Modeled test output:          5457149.477
Actual test output:           5483199.303
Tested output ratio:          1.005
Tested Capacity:              1004.774
Bounds:                       970.0, None


Using reporting conditions from das.

Capacity Test Result:         PASS
Modeled test output:          5457980.554
Actual test output:           5483199.303
Tested output ratio:          1.005
Tested Capacity:              1004.621
Bounds:                       970.0, None


100.480% - Cap Ratio
100.460% - Cap Ratio after pval check
```

	das_pvals	sim_pvals	das_params	sim_params
poa	0.00000	0.00000	6,898.78468	6,908.71997
I(poa * poa)	0.00000	0.00000	-0.74979	-0.84880
I(poa * t_amb)	0.00000	0.00000	-31.54300	-28.50179
I(poa * w_vel)	0.01569	0.53672	3.72609	-0.39352

The `capdata_results` method provides a more simple output and also returns the capacity ratio as a decimal number.

```
Using reporting conditions from das.

Capacity Test Result:         PASS
Modeled test output:          5457149.477
Actual test output:           5483199.303
Tested output ratio:          1.005
Tested Capacity:              1004.774
Bounds:                       970.0, None


1.0047735225424215

The capacity ratio is 100.48%
```

Finally, we can use the `overlay_scatters` function to overlay scatter plots of the data remaining after filtering from the meas and modeled `CapData` objects.

