Sunshine Analytics

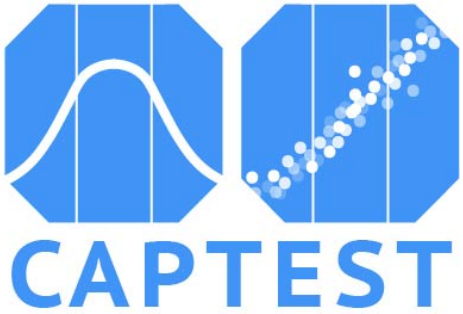
captest

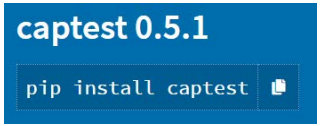
Open Source Package for Reproducible Performance Testing

May 15, 2019 I Ben Taylor, Exyte Energy & Jessica Forbess, Sunshine Analytics

Agenda

- Background – ASTM E2848
- Motivation
  - Efficiency and Improved Workflow / Tools
  - Standardization of Calculations
  - Reproducible Test Calculations
- Essential Functionality
  - Package Structure
  - Load data
  - Identify Data to be used in regression
  - Visually review data
  - Aggregate Columns and/or Filter Columns
  - Filter rows
  - Perform regressions
  - Calculate reporting conditions
  - Results – summarize filtering steps
  - Results – compare regressions
- Goals for Development
  - Expand user base
  - Technical Review
  - OSS development and consensus building
  - New features, testing, bug squashing



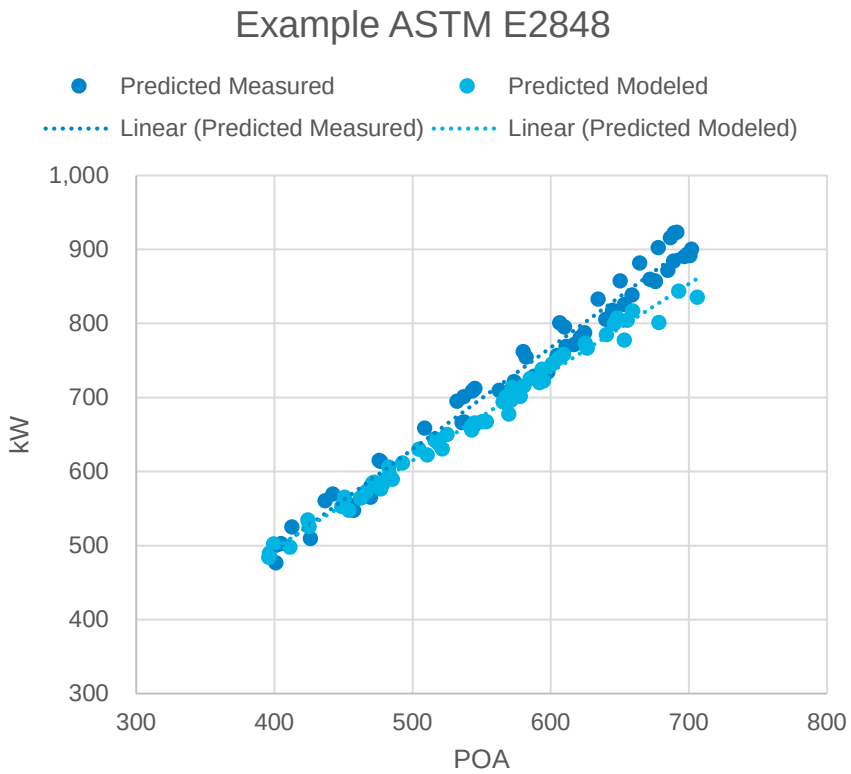


https://github.com/bt-/pvcaptest

Basis of ASTM E2848

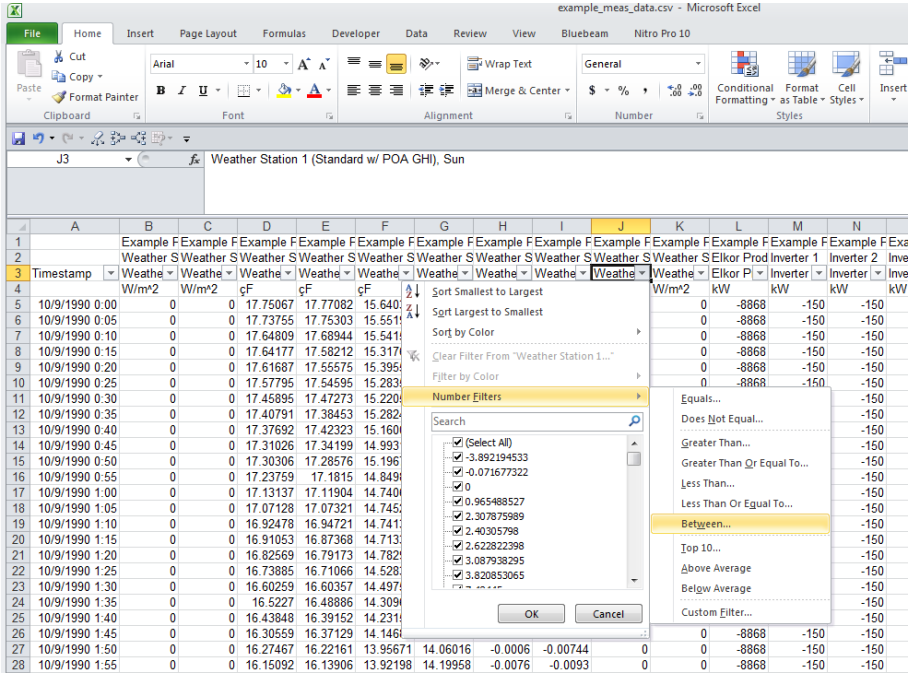
Proving a constructed PV plant matches the model

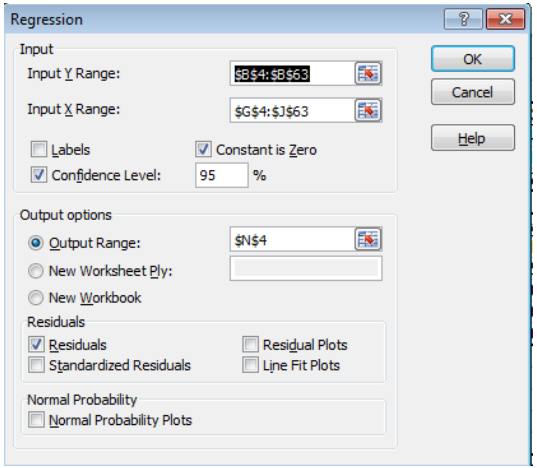
- Multi-linear regression
  - AC Power ~ POA, ambient temperature, wind speed
  - Compare regressions of modeled and measured data to make sure they are within expected range
  - Graph is rough visual, only shows POA correlation
- 11 page ASTM standard plus N (1-20) pages of test exhibit
  - Complexity defining what data and test conditions are valid



Motivation

Efficiency and Improved Workflow

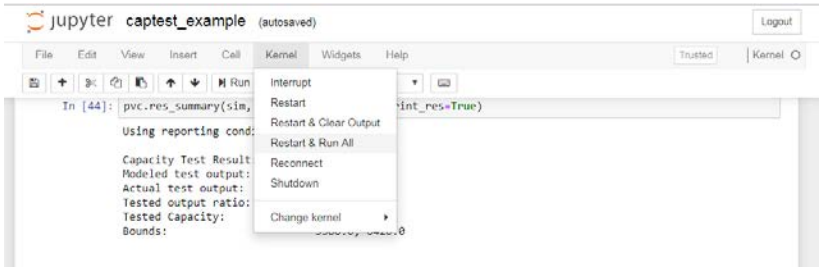


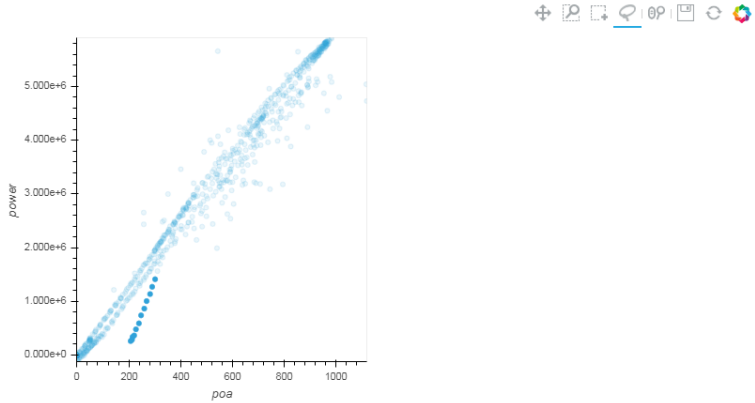


Motivation

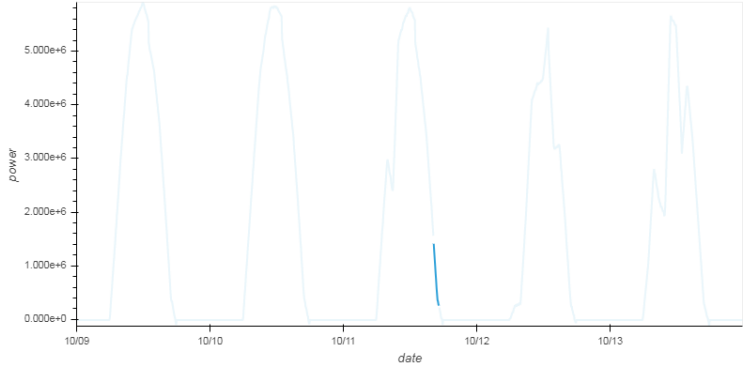
Efficiency and Improved Workflow

Edit, Restart, & Run All





Using Bokeh and Holoviews Packages for Interactive Visualization



Motivation

Standardization of Calculations

```
In [16]: das.filter_irr(200, 2000)
```

```
In [43]: pvc.get_summary(das, sim)
```

Out[43]:

		pts_after_filter	pts_removed	filter_arguments
das	filter_sensors	1245	195	(),
	custom_filter	1240	5	(<function DataFrame.dropna at 0x0000000008687...
	filter_irr	424	816	(200, 2000),
	filter_outliers	407	17	(),
	reg_cpt	385	22	(), 'filter': True, 'summary': False
	rep_cond	385	0	(),
	filter_irr	294	91	(0.5, 1.5), 'ref_val': 762.7265615000001
	reg_cpt	294	0	(),
sim	filter_time	1441	7319	(), 'test_date': '10/11/1990', 'days': 60
	filter_irr	397	1044	(200, 930),
	filter_pvsyst	397	0	(),
	filter_irr	284	113	(0.5, 1.5), 'ref_val': 762.7265615000001
	reg_cpt	284	0	(),

```
2095    @update_summary
2096    def filter_irr(self, low, high, ref_val=None, col_name=None, inplace=True):
2097        """
2098        Filter on irradiance values.
2099 ›      
2118        """
2119        if col_name is None:
2120            irr_col = self.__get_poa_col()
2121        else:
2122            irr_col = col_name
2123
2124        df_flt = flt_irr(self.df_flt, irr_col, low, high,
2125                         ref_val=ref_val)
2126        if inplace:
2127            self.df_flt = df_flt
2128        else:
2129            return df_flt
```

```
380 def flt_irr(df, irr_col, low, high, ref_val=None):
381     """
382     Top level filter on irradiance values.
383 ›   
401     if ref_val is not None:
402         low *= ref_val
403         high *= ref_val
404
405     df_renamed = df.rename(columns={irr_col: 'poa'})
406
407     flt_str = '@low <= ' + 'poa' + ' <= @high'
408     indx = df_renamed.query(flt_str).index
409
410     return df.loc[indx, :]
```

Motivation
Standardization of Calculations

Complement ASTM E2848

- The ASTM standard defines a “Standard Test Method for Reporting Photovoltaic Non-Concentrator System Performance”
- One of the goals of the Captest package is to provide a standard set of functions/methods implementing the data manipulation, filtering, regressions, and visualization described in E2848
    - Pragmatically, will also need to allow for adjustments per exhibit language

Github as a platform for building consensus

- Capacity testing requires agreement between multiple parties – Developer / Owner / EPC / Financer
- If these parties agree on calculations within a standard open source library, the review of capacity test results should be much easier and more consistent project-to-project

Motivation
Reproducible Calculations

Capacity Test Reports in Jupyter Notebook

- Present data, visualizations, narrative text, images, and calculations together in a report that can be re-run.

Requirements to re-run notebook:

- Input data (.csv) measured & simulated
- Jupyter notebook (.ipynb) with analysis
- *Environment with captest package and dependencies installed*

Filtering Measured Data

The `CapData` class provides a number of convience methods to apply filtering steps as defined in ASTM E2848. The following section demonstrates the use of the more commonly used filtering steps to remove measured data points.

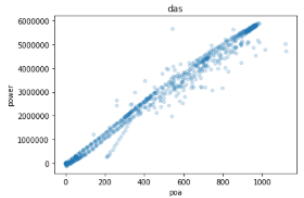
```
In [46]: # Uncomment and run to copy over the filtered dataset with the unfiltered data.
         das.reset_flt()
```

A common first step is to review the scatter plot of the POA irradiance against the power production. The `scatter` method returns a basic non-interactive version of this plot as shown below.

If you have the optional dependency Holoviews installed, `scatter_hv` will return an interactive scatter plot. Additionally, `scatter_hv` includes an option to return a timeseries plot of power that is linked to the scatter plot, so points selected in the scatter plot will be highlighted in the time series.

```
In [1]: # Uncomment the below line to use scatter_hv with linked time series
        # das.scatter_hv(timeseries=True)
```

```
In [11]: das.scatter()
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0xda85cf8>
```



In this example, we have multiple measurements of the same value from different sensors. In this case a common first step is to compare measurements from the different sensors and remove data for timestamps where the measurements differ above some acceptable threshold. The `filter_sensors` method provides a convient method to accomplish this taks for the groups of measurements identified as regression values.

```
In [12]: das.filter_sensors()
```

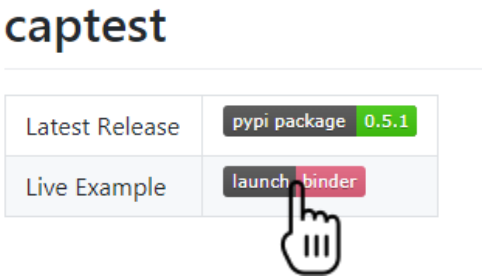
Essential Functionality

Essential Functionality

1. **Package structure**
2. **Load data – modeled or measured**
   - **Group data**
3. **Identify Data to be used in regression**
4. **Visually review data**
5. **Aggregate Columns and/or Filter Columns**
6. **Filter rows**
7. **Perform regressions**
8. **Calculate reporting conditions**
9. **Results – summarize filtering steps**
10. **Results – compare regressions**

[Link to live example](#).

https://github.com/bt-/pvcaptest



Essential Functionality
Package Structure

captest package

captest module

- CapData class
- Capacity test functions

future modules

- NREL weather corrected PR test

Essential Functionality
Loading Data and Grouping Measurements

CapData.load_data()

- Two ways to load csv files:
  - Read a specific csv file – modeled data (tested on PVsyst) or measured data
  - Read all csv files in a directory – expects each file to have the same columns and appends rows

- Background tasks:
  - Group columns by measurement type and store in the ‘trans’ attribute as a dictionary
  - Copies data to df_flt attribute
  - If clear_sky is True, adds modeled clear sky GHI and POA columns to dataframe

Essential Functionality

Loading Data and Grouping Measurements

Measurement Types (type_defs)	Measurement Sub Types (sub_type_defs)	Irradiance Sensor Types (irr_sensor_defs)
irr	ghi	ref_cell
temp	poa	pyran
wind	amb	clear_sky
pf	mod	
op_state	mtr	
real_pwr	inv	
shade		
pvsyst_losses		
index		

CapData.load_data()

- Calls CapData._CapData__set_trans
- irr : irradiance, irr, plane of array, poa, ghi, global, glob, w/m^2, w/m2, w/m, w/

CapData.trans	irr-poa-		irr-ghi-		temp-amb-		temp-mod-		wind--		-mtr-
Timestamp	Weather Station 1 (Standard w/ POA GHI), Sun W/m^2	Weather Station 2 (Standard with POA GHI), Sun W/m^2	Weather Station 1 (Standard w/ POA GHI), Sun2 W/m^2	Weather Station 2 (Standard with POA GHI), Sun2 W/m^2	Weather Station 1 (Standard w/ POA GHI), TempF çF	Weather Station 2 (Standard with POA GHI), TempF çF	Weather Station 1 (Standard w/ POA GHI), Temp1 çF	Weather Station 2 (Standard with POA GHI), Temp1 çF	Weather Station 1 (Standard w/ POA GHI), WindSpeed mph	Weather Station 2 (Standard with POA GHI), WindSpeed mph	Elkor Production Meter, KW kW
10/9/1990 0:00	0	0	0	0	17.75067	17.77082	15.64035	15.52654	-0.00072	-0.00722	-8868
10/9/1990 0:05	0	0	0	0	17.73755	17.75303	15.55192	15.70683	-0.00888	-0.00719	-8868
10/9/1990 0:10	0	0	0	0	17.64809	17.68944	15.54152	15.50686	0.008686	0.002557	-8868
10/9/1990 0:15	0	0	0	0	17.64177	17.58212	15.31764	15.51052	0.000598	-0.01181	-8868
10/9/1990 0:20	0	0	0	0	17.61687	17.55575	15.39555	15.53016	-0.00909	0.005773	-8868
10/9/1990 0:25	0	0	0	0	17.57795	17.54595	15.28356	15.28266	-0.00074	0.000656	-8868
10/9/1990 0:30	0	0	0	0	17.45895	17.47273	15.22055	15.36971	0.004863	0.000722	-8868
10/9/1990 0:35	0	0	0	0	17.40791	17.38453	15.28244	15.25047	0.007353	0.005497	-8868

Essential Functionality

Identify data to be used in regression

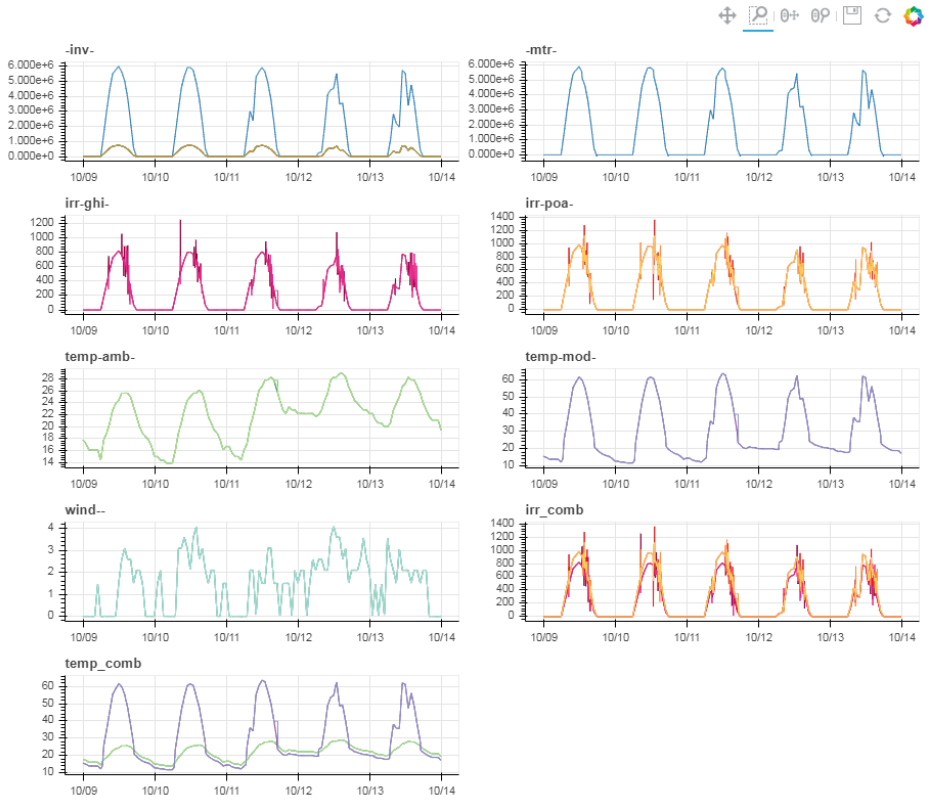
CapData. set_reg_trans(power='-mtr-',
poa='irr-poa-',
t_amb='temp-amb-',
w_vel='wind--')

CapData.reg_trans	poa				t_amb				w_vel		power
CapData.trans	irr-poa-		irr-ghi-		temp-amb-		temp-mod-		wind--		-mtr-
Timestamp	Weather Station 1 (Standard w/ POA GHI), Sun W/m^2	Weather Station 2 (Standard with POA GHI), Sun W/m^2	Weather Station 1 (Standard w/ POA GHI), Sun2 W/m^2	Weather Station 2 (Standard with POA GHI), Sun2 W/m^2	Weather Station 1 (Standard w/ POA GHI), TempF çF	Weather Station 2 (Standard with POA GHI), TempF çF	Weather Station 1 (Standard w/ POA GHI), Temp1 çF	Weather Station 2 (Standard with POA GHI), Temp1 çF	Weather Station 1 (Standard w/ POA GHI), WindSpeed mph	Weather Station 2 (Standard with POA GHI), WindSpeed mph	Elkor Production Meter, KW kW
10/9/1990 0:00	0	0	0	0	17.75067	17.77082	15.64035	15.52654	-0.00072	-0.00722	-8868
10/9/1990 0:05	0	0	0	0	17.73755	17.75303	15.55192	15.70683	-0.00888	-0.00719	-8868
10/9/1990 0:10	0	0	0	0	17.64809	17.68944	15.54152	15.50686	0.008686	0.002557	-8868
10/9/1990 0:15	0	0	0	0	17.64177	17.58212	15.31764	15.51052	0.000598	-0.01181	-8868
10/9/1990 0:20	0	0	0	0	17.61687	17.55575	15.39555	15.53016	-0.00909	0.005773	-8868
10/9/1990 0:25	0	0	0	0	17.57795	17.54595	15.28356	15.28266	-0.00074	0.000656	-8868
10/9/1990 0:30	0	0	0	0	17.45895	17.47273	15.22055	15.36971	0.004863	0.000722	-8868
10/9/1990 0:35	0	0	0	0	17.40791	17.38453	15.28244	15.25047	0.007353	0.005497	-8868

Essential Functionality
Visually Review Data

CapData.plot()

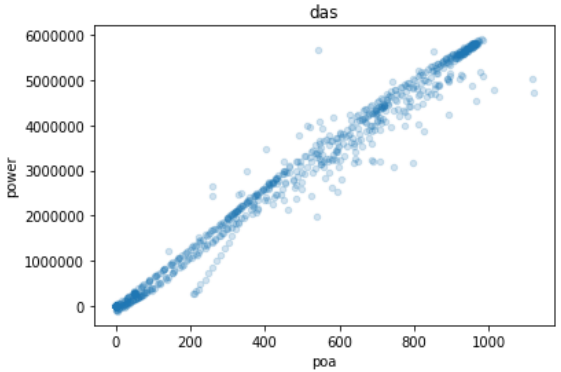
- Creates Bokeh gridplot of data
- One plot per key in the translation (CapData.trans) dictionary attribute
- One line on each plot per column in the list of (CapData.trans) values
- Flexible – height, width, marker, columns of plots, plot subset of groups, change order of plots, include legends, merge plots (measured irradiance and modeled irradiance)
- Interactive- linked zooming and panning, hover tool



Essential Functionality
Visually Review Data

CapData.scatter

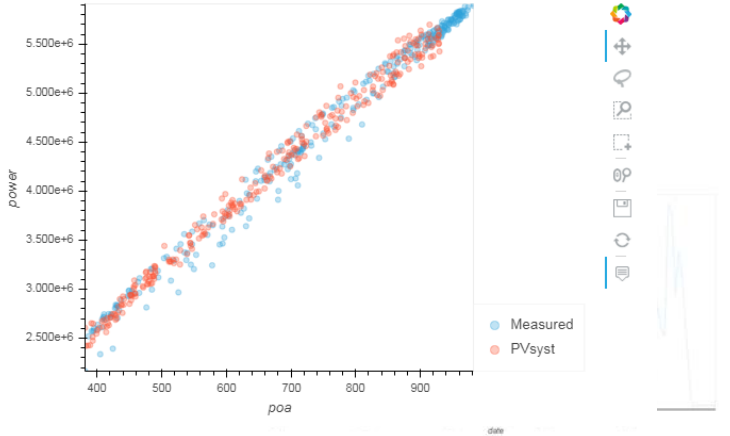
- Matplotlib poa vs. power scatter



CapData.scatter_hv

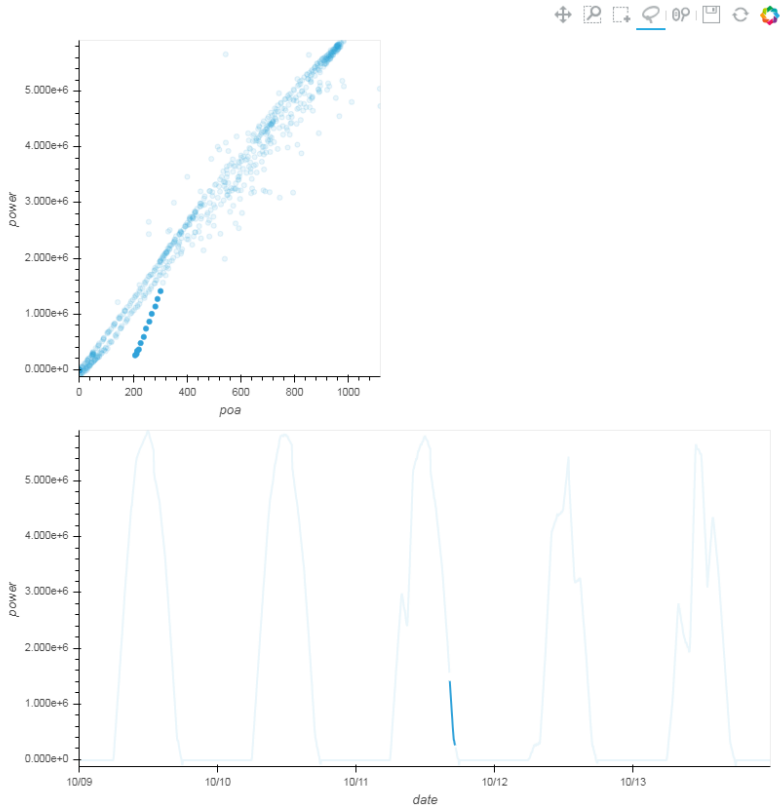
- Holoviews plot with or without linked time series

```
In [58]: %%opts Scatter (alpha=0.3)
         %%opts Scatter [width=600]
         das.scatter_hv().relabel('Measured') * sim.scatter_hv().relabel('PVsyst')
Out[58]:
```



Essential Functionality
Visually Review Data

Holoviews plot with linked time series



Essential Functionality
Aggregate Columns of Data

CapData.agg_sensors()

- Aggregate measurements of the same variable from different sensors.

```
In [10]: das.agg_sensors(agg_map={'-inv-':'sum', 'irr-poa-':'mean', 'temp-amb-':'mean', 'wind--':'mean'})

In [11]: das.df.head()
```

Out[11]:

	-inv-sum-agg	irr-poa-mean-agg	temp-amb-mean-agg	wind--mean-agg	Weather Station 1 Sun, W/m^2	Weather Station 2 Sun, W/m^2	Weather Station 1 Sun2, W/m^2	Weather Station 2 Sun2, W/m^2	Weather Station 1 TempF, çF	Weather Station 2 TempF, çF	...	Elkor Production Meter KW, kW	Inverter 1 KW, kW	Inverter 2 KW, kW	Inverter 3 KW, kW	Inverte 4 KV kV
1990-10-09 00:00:00	-1200.0	0.0	17.760744	-0.003971	0.0	0.0	0.0	0.0	17.750666	17.770821	...	-8868.0	-150.0	-150.0	-150.0	-150.
1990-10-09 00:05:00	-1200.0	0.0	17.745288	-0.008036	0.0	0.0	0.0	0.0	17.737545	17.753030	...	-8868.0	-150.0	-150.0	-150.0	-150.
1990-10-09 00:10:00	-1200.0	0.0	17.668763	0.005622	0.0	0.0	0.0	0.0	17.648090	17.689437	...	-8868.0	-150.0	-150.0	-150.0	-150.
1990-10-09 00:15:00	-1200.0	0.0	17.611945	-0.005605	0.0	0.0	0.0	0.0	17.641772	17.582119	...	-8868.0	-150.0	-150.0	-150.0	-150.
1990-10-09 00:20:00	-1200.0	0.0	17.586308	-0.001660	0.0	0.0	0.0	0.0	17.616870	17.555746	...	-8868.0	-150.0	-150.0	-150.0	-150.

5 rows × 24 columns

Essential Functionality

Filter Methods – Remove rows of data

- Filters data stored in CapData.df_flt and saves result to CapData.df_flt
- Filter methods update summary data
- CapData.reset_flt will overwrite CapData.df_flt with DataFrame in CapData.df and reset summary data

```
In [25]: das.get_summary()
```

Out[25]:

		pts_after_filter	pts_removed	filter_arguments
das	filter_sensors	1245	195	(),
	custom_filter	1240	5	(<function DataFrame.dropna at 0x0000000008688...
	filter_irr	424	816	(200, 2000),
	filter_outliers	407	17	(),
	reg_cpt	385	22	(), 'filter': True, 'summary': False

Essential Functionality
Filter Methods

- filter_irr – low, high, percentage band around ref value
- filter_pvsyst – FShdBm, IL Pmin, IL Pmax, IL Vmin, IL Vmax
- filter_time – time between two dates or time period around a date
  - Future feature - Additional functionality and/or additional time filtering methods
- filter_outliers – applies elliptic envelope from scikit-learn
  - Future feature – lasso outliers on scatter plot to remove
- filter_pf – removes data where power factor is below given threshold
- filter_sensors – removes data where measurements of the same property differ by more than x%
  - Future feature – allow parameter to select hard value (for different exhibit requirements)
- filter_clearsky – keeps only clear data using the pvlib's detect_clearsky
- custom_filter – updates summary information for pandas DataFrame methods or user custom functions
  - Can pass any function that takes a dataframe and returns the dataframe with rows removed
  - pd.DataFrame.dropna – remove rows with any or all missing data
  - pd.DataFrame.between_time – select data between times of day

Essential Functionality
Perform Regressions

- Captest uses statsmodels to run regressions, which in turn also uses patsy
- The regression equation defined in ASTM E2848 is defined by default in CapData.reg_fml

```
In [51]: sim.reg_fml
Out[51]: 'power ~ poa + I(poa * poa) + I(poa * t_amb) + I(poa * w_vel) - 1'
```

- Regression results are saved to CapData.ols_model

```
In [53]: sim.ols_model.params
Out[53]: poa               7662.794466
         I(poa * poa)        -0.833510
         I(poa * t_amb)     -31.284536
         I(poa * w_vel)      -1.208663
         dtype: float64

In [54]: sim.ols_model.pvalues
Out[54]: poa               0.000000e+00
         I(poa * poa)      2.323363e-170
         I(poa * t_amb)    3.211807e-170
         I(poa * w_vel)     2.887841e-01
         dtype: float64
```

```
In [45]: sim.reg_cpt()
```

OLS Regression Results

Dep. Variable:	power	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	2.587e+06
Date:	Sat, 11 May 2019	Prob (F-statistic):	0.00
Time:	17:40:42	Log-Likelihood:	-3245.9
No. Observations:	284	AIC:	6500.
Df Residuals:	280	BIC:	6514.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
poa	7662.7945	15.456	495.779	0.000	7632.370	7693.219
I(poa * poa)	-0.8335	0.013	-64.665	0.000	-0.859	-0.808
I(poa * t_amb)	-31.2845	0.484	-64.585	0.000	-32.238	-30.331
I(poa * w_vel)	-1.2087	1.137	-1.063	0.289	-3.447	1.030

Omnibus:	25.890	Durbin-Watson:	2.014
Prob(Omnibus):	0.000	Jarque-Bera (JB):	9.635
Skew:	-0.169	Prob(JB):	0.00809
Kurtosis:	2.163	Cond. No.	6.11e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.11e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Essential Functionality
Calculate Reporting Conditions

- Captest.rep_cond used to calculate reporting conditions
- Separate statistical aggregation (mean, median, etc.) can be specified for each reporting conditions – POA, temperature, and wind speed
- ‘Irradiance Balance’ option - reporting irradiance determined by finding the irradiance that results in a balance of points within a +/- percent range of the reporting irradiance
- Always uses the filtered data in df_flt for calculations
- Can be used to calculate reporting conditions by month or at other frequencies
- Reporting conditions saved as a DataFrame to the CapData.rc attribute
    - Results function requires a CapData object with rc set and one with rc set to None

- See example notebook “Reporting Conditions Examples.ipynb” in github repository
- https://github.com/bt-/pvcaptest/tree/master/examples

Essential Functionality
Summary of Filtering

```
In [46]: pvc.get_summary(das, sim)
```

Out[46]:

		pts_after_filter	pts_removed	filter_arguments
das	filter_sensors	1245	195	(),
	custom_filter	1240	5	(<function DataFrame.dropna at 0x0000000008688...
	filter_irr	424	816	(200, 2000),
	filter_outliers	407	17	(),
	reg_cpt	385	22	(), 'filter': True, 'summary': False
	rep_cond	385	0	(),
	filter_irr	294	91	(0.5, 1.5), 'ref_val': 762.7265615000001
	reg_cpt	294	0	(),
sim	filter_time	1441	7319	(), 'test_date': '10/11/1990', 'days': 60
	filter_irr	397	1044	(200, 930),
	filter_pvsyst	397	0	(),
	filter_irr	284	113	(0.5, 1.5), 'ref_val': 762.7265615000001
	reg_cpt	284	0	(),

das and sim are both CapData objects

Essential Functionality
Results

- CapData.res_summary
- sim and das are both CapData objects
- One but not both of the CapData objects must have reporting conditions set
- Both CapData objects must have regression results saved to CapData.ols_model
- Calculates and compares capacities using the regression parameter from each CapData object and the reporting conditions
- Expects the CapData object for simulated data first and measured data second.

```
In [47]: pvc.res_summary(sim, das, 6000, '+/- 7', print_res=True)

Using reporting conditions from das.

Capacity Test Result:         PASS
Modeled test output:          4781111.801
Actual test output:           4733483.041
Tested output ratio:          0.990
Tested Capacity:              5940.229
Bounds:                       5580.0, 6420.0


Using reporting conditions from das.

Capacity Test Result:         PASS
Modeled test output:          4782948.044
Actual test output:           4733483.041
Tested output ratio:          0.990
Tested Capacity:              5937.948
Bounds:                       5580.0, 6420.0


99.000% - Cap Ratio
98.970% - Cap Ratio after pval check
```

Out[47]:

	das_pvals	sim_pvals	das_params	sim_params
poa	0.00000	0.00000	7,781.41551	7,662.79447
I(poa * poa)	0.00000	0.00000	-0.45689	-0.83351
I(poa * t_amb)	0.00000	0.00000	-51.77576	-31.28454
I(poa * w_vel)	0.01091	0.28878	12.33830	-1.20866

Goals for Development

Goals for Development
Expand user base

Add Users

- How many people here run or review capacity tests?
- Is this something you would consider using?
- What would stop you?

Goals for Development
Technical Review

Expert Signoff

- Independent Engineers who have more flexibility in reviewing test results could add this to the review process to compare to their current methods
- Researchers using ASTM for internal metrics can add this method

Goals for Development
Future work

Additional features / bug fixing

- More interactive plotting
- Better measurement grouping
- Improved documentation
- Other capacity test methods than ASTM e2848?

Links to Resources

- jupyter-notebook-beginner-guide
- ASTM E2848 - purchase required
- Pandas Documentation
- Holoviews Documentation
- pvlib Documentation
- Statsmodels Documentation