PV Clustering

June 7, 2020

This notebook is based on the conference paper available here: http://www.ibpsa.org/proceedings/BS2019/BS2019_210725.pdf (http://www.ibpsa.org/proceedings/BS2019/BS2019_210725.pdf)

Paper Abstract: This paper demonstrates a clustering method for grouping PVs of arbitrary orientation affected by nonuniform local shading. For a project with 44,000 PV panels cladding doubly curved roof surfaces, every panel has a unique orientation. In addition, clerestory windows cause non-uniform near-field shading. The combination of curvature and shading causes every panel to receive a different amount of irradiance at any time. The PV panel with lowest output (lowest irradiance) in a MPPT string limits the output of the whole string. The method for grouping 44,000 PV panels into 296 MPPT strings affects the total annual energy producing. The PV array grouped with clustering generated 7.7% more energy annually compared to the same array grouped in a simple grid.

The clustering method from the paper (which used octave) has be re-coded using python and scikit-learn in this notebook.

Here we're exploring the methods (and a few additional methods) with roof bay 5 as an example. Roof bay 5 contains 1175 PV panels, has a shadow causing clearstory at the upper left edge.

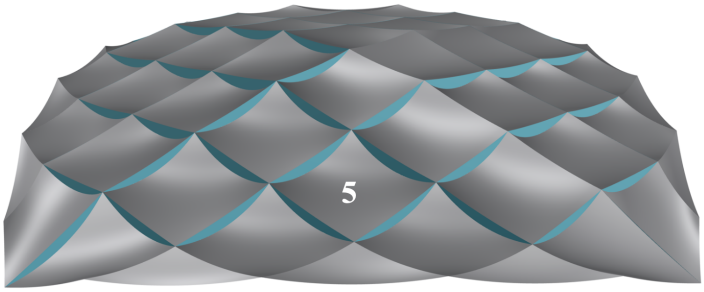


Figure 3: Rendering of CAD model of roof surfaces with roof bay 5 identified.

In [1]:
```
%matplotlib notebook
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import time
```

PV panel coordinates

The first file to load is the position and normal vector for all the PV panels. These positions represent the location of the panels in real world space with +Y as north, +X as east, and +Z as up.

In [2]:
```
PV_geometry = pd.read_csv('bay_05_wnormals.txt',
                          index_col=False,
                          sep='\t',
                          names=['x','y','z','xdir','ydir','zdir'])
PV_geometry.head()
```

Out[2]:

	x	y	z	xdir	ydir	zdir
0	-63.4939	-218.917	57.39409	0.340272	0.126066	0.931838
1	-61.3550	-217.368	56.53611	0.311589	0.121175	0.942459
2	-61.3002	-220.484	56.73805	0.321068	0.116131	0.939909
3	-59.1910	-215.821	55.75290	0.283460	0.114895	0.952077
4	-59.1422	-218.938	55.93800	0.293467	0.111522	0.949442

Irradiance on selected sunny days

The next file to load conatins irradiance on the panels at hourly increments on sunny days. The days are chosen from a TMY weather file, and the panel irradiance was simulated using the diffuse and dirrect irradiance, and position of the sun from the increments in the weather file. There are 73 hours from seven days. These are the values used to cluster the PV panels.

The days selected are: 1/21, 2/25, 3/24, 4/21, 5/21, 6/27, and 12/23

The datafram has 73 columns (one for each daylight hour on the 7 days) and 1175 rows (one for each PV panel on bay 5

In [3]:
```
sunny_days= pd.read_csv('sky/sunnydays.wea',
                        index_col=False,
                        skiprows=6,
                        header=None,
                        sep=' ',
                        names=['month','day','hour','DNI','DHI'])
sunny_days['h'] = np.floor(sunny_days['hour'])
sunny_days['m'] = 60 * ( sunny_days['hour'] - sunny_days['h'] )
sunny_days['timestamp'] = sunny_days.apply(
    lambda x : f"{x['month']:0.0f}/{x['day']:0.0f} {x['h']:0.0f}:{x['m']:0.0f}",
    axis=1)
```

In [4]:
```
PV_irrad_sunny_days = pd.read_csv('bay_05_irrad_sunnydays.txt',
                        index_col=False,
                        skiprows=7,
                        header=None,
                        sep='\t',
                        names=sunny_days['timestamp'].tolist())
PV_irrad_sunny_days.head(10)
```

Out[4]:

	1/21 9:30	1/21 10:30	1/21 11:30	1/21 12:30	1/21 13:30	1/21 14:30	1/21 15:30	2/25 10:30	2/25 11:30	2/25 12:30	...	12/23 7:30	12/2 8:3
0	185.61412	235.31117	262.07126	233.78324	191.07989	112.836170	69.058983	323.63188	412.06757	421.53892	...	3.775695	59.6907
1	191.09562	247.65979	275.94636	252.08739	219.80486	121.193680	67.294195	329.30928	425.12970	441.34019	...	3.466167	62.2614
2	197.79418	251.03109	279.63151	253.76834	210.55725	132.885560	72.842641	333.35112	429.02639	446.89413	...	3.938332	65.7489
3	197.02886	261.64145	290.60440	277.29829	239.44388	101.896760	65.932735	334.31540	444.92044	467.20822	...	3.264447	64.1750
4	202.61895	260.94418	294.61436	276.83813	237.35804	153.860160	72.034608	339.69334	447.40253	466.52906	...	3.646420	67.8901
5	203.50100	264.56974	296.59723	273.56021	232.89315	150.909690	75.273669	344.18574	445.74786	461.85050	...	4.026886	72.1067
6	201.49283	272.36561	311.75741	296.76131	263.52980	78.859785	64.931857	339.22711	462.23101	489.32475	...	3.214098	68.8547
7	203.86451	270.96599	312.32526	296.81755	257.52038	173.769390	71.265191	345.02636	462.58618	490.69643	...	3.634854	70.6415
8	211.50909	274.19865	310.27010	295.45725	253.52074	169.721400	75.124070	347.71737	461.39589	481.40367	...	3.889603	72.5827
9	214.08655	274.60066	310.38494	291.03226	251.09469	163.564260	77.169332	350.97300	459.79475	482.87665	...	4.356566	74.0808

10 rows × 73 columns

Annual Hourly Irradiance

The final file to load contains Irradiance on each PV panel for all hours in the TMY weather file. The dataframe has 8760 columns (one for each hour of the year) and 1175 rows (one for each PV panel).

In [5]:
```
all_days= pd.read_csv('sky/Moffett.wea',
                      index_col=False,
                      skiprows=6,
                      header=None,
                      sep=' ',
                      names=['month','day','hour','DNI','DHI'])
all_days['h'] = np.floor(all_days['hour'])
all_days['m'] = 60 * ( all_days['hour'] - all_days['h'] )
all_days['timestamp'] = all_days.apply(
    lambda x : f"{x['month']:0.0f}/{x['day']:0.0f} {x['h']:0.0f}:{x['m']:0.0f}",
    axis=1)
```

In [6]:
```
PV_irrad_all_days = pd.read_csv('bay_05_irrad_alldays.txt',
                                index_col=False,
                                skiprows=7,
                                header=None,
                                sep='\t',
                                names=all_days['timestamp'].tolist())

PV_irrad_all_days.head()
```

Out[6]:

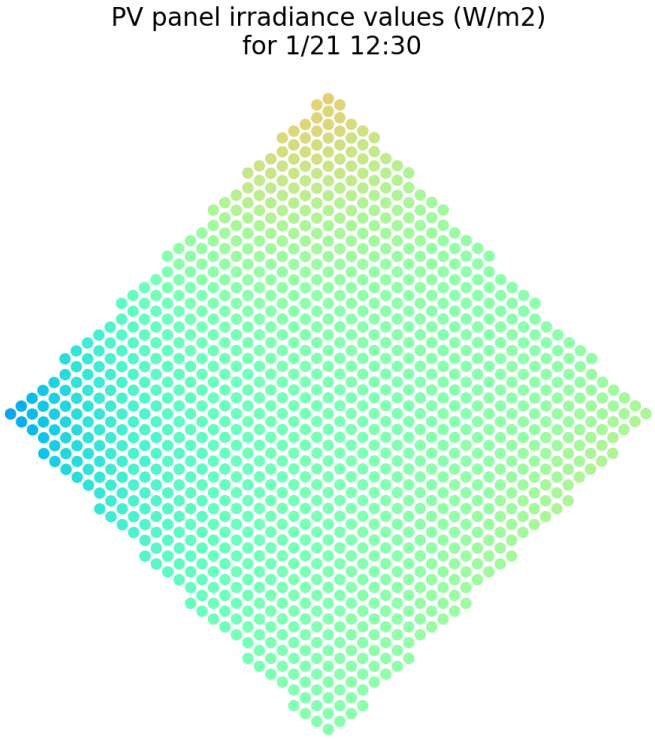
	1/1 0:30	1/1 1:30	1/1 2:30	1/1 3:30	1/1 4:30	1/1 5:30	1/1 6:30	1/1 7:30	1/1 8:30	1/1 9:30	...	12/31 14:30	12/31 15:30	12/31 16:30	12/31 17:30	12/31 18:30	12/31 19:30	12/3 20:
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.450213	59.822393	161.14958	...	96.907864	38.279221	6.188614	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.258428	62.526320	165.30597	...	95.591616	37.701426	6.095520	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.554192	65.736090	173.08235	...	99.839848	39.342195	6.350118	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.139230	64.059414	171.12973	...	94.543385	37.232654	6.019378	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.379202	67.458865	174.49689	...	99.277545	39.045406	6.300263	0.0	0.0	0.0	0

5 rows × 8760 columns

Data Exploration with Plotting

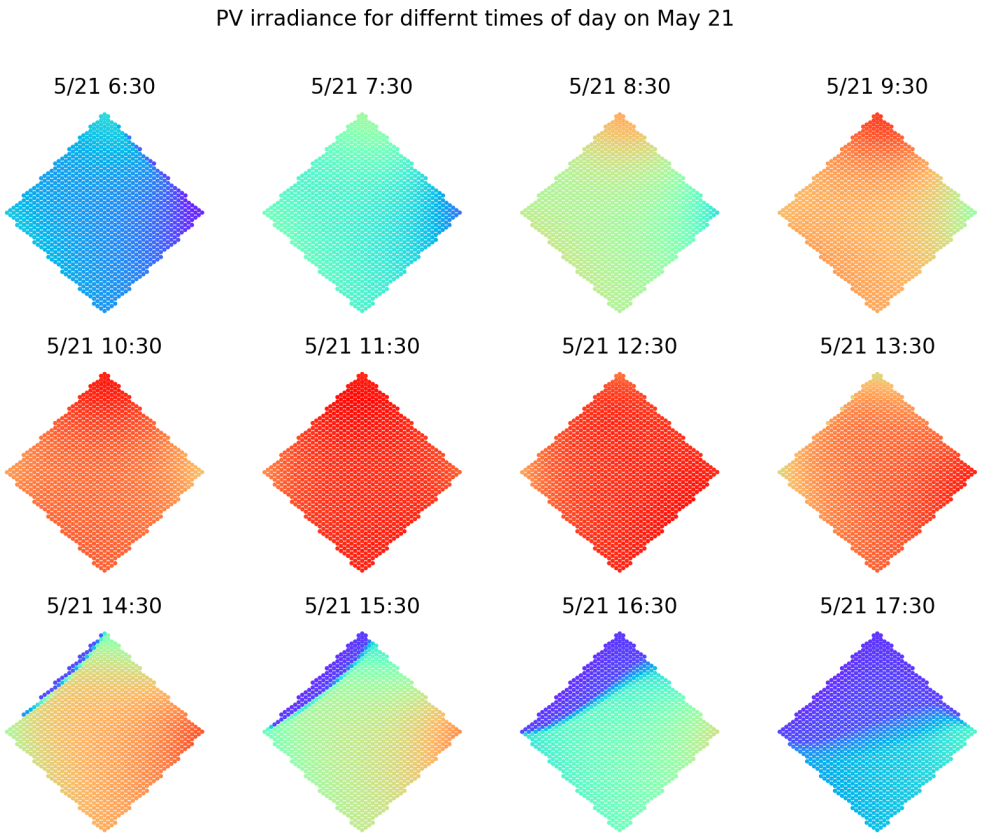
In [7]:
```
## this is a helper function to plot PV panels
def PV_scatter(position_x, position_y, color, cmap='rainbow', vmin=0, vmax=1000, size=3, title=''):
    fig = plt.figure(figsize=(6,6))
    plt.scatter(position_x, position_y, c=color,
                s=size, cmap=cmap, vmin=vmin, vmax=vmax)
    plt.gca().set_title(title)
    plt.gca().spines['left'].set_visible(False)
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.gca().spines['bottom'].set_visible(False)
    plt.gca().xaxis.set_visible(False)
    plt.gca().yaxis.set_visible(False)
```

In [8]:
```
irrad_max = PV_irrad_sunny_days.max().max()
column_to_plot = 3

PV_scatter(PV_geometry['x'], PV_geometry['y'],
           color=PV_irrad_sunny_days.iloc[:,column_to_plot],
           size=20,
           vmax=irrad_max,
           title='PV panel irradiance values (W/m2)\n for ' + PV_irrad_sunny_days.columns[column_to_pl
ot])
```



Plotting hourly irradiance maps for a sunny day in May.

In [9]:
```
irrad_max = PV_irrad_sunny_days.max().max()
irrad_plot, axs = plt.subplots(3,4,figsize=(10,7.5))
irrad_plot.suptitle('PV irradiance for differnt times of day on May 21')
first_chart = 38
for i,ax in enumerate(axs.ravel()):
    ax.scatter(PV_geometry['x'], PV_geometry['y'], c=PV_irrad_sunny_days.iloc[:,i+first_chart],
               s=2, cmap='rainbow', vmin=0, vmax=irrad_max)
    ax.spines['left'].set_visible(False)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    ax.set_title(PV_irrad_sunny_days.columns[i+first_chart])
irrad_plot.figsize=(16,12)
```



Principal Component Analysis

Reducing from 73 irradiance dimensions to 2 dimensions for plotting.

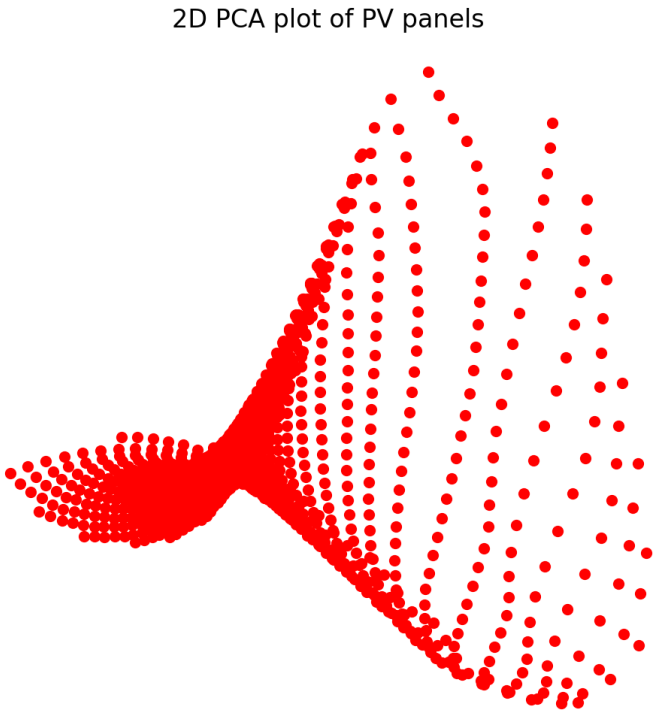
In [10]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca_irrad_sunny_days = pca.fit_transform(PV_irrad_all_days)

print("explained variance ratio: ",pca.explained_variance_ratio_)
```

```
explained variance ratio:  [0.68805444 0.15149994 0.06084838]
```

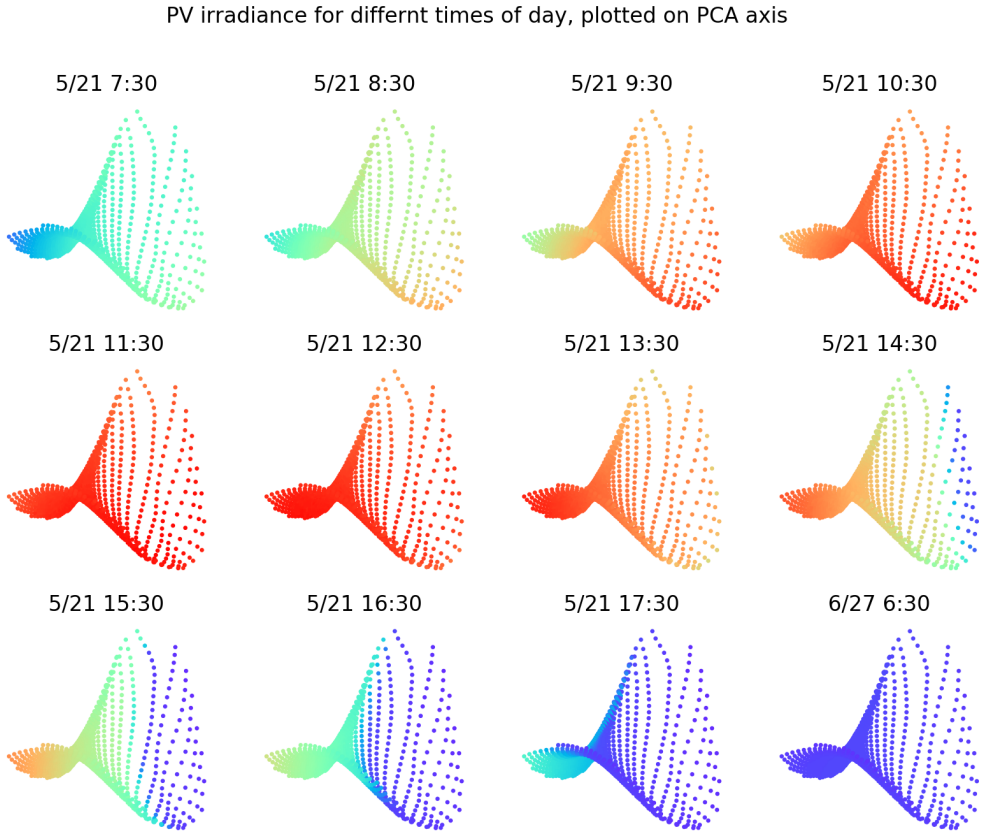
The explained variance ratio describes how much of the total variance in the data is captured by each dimension in the PCA. In this example, the first dimension (X axis) recreates 68.8% of the total variance in the 73-dimensional data. The second dimension (y-axis) recreates 15.1% of variance in the 73-dimensional data. So with this PCA transform we can illustrate 84% of the variance in the original data with 2 dimensions.

And now to plot the PV panels in the PCA transform.

In [11]:
```
PV_scatter(pca_irrad_sunny_days[:,0], pca_irrad_sunny_days[:,1],
           color=[(1,0,0)], size=20, title='2D PCA plot of PV panels')
```



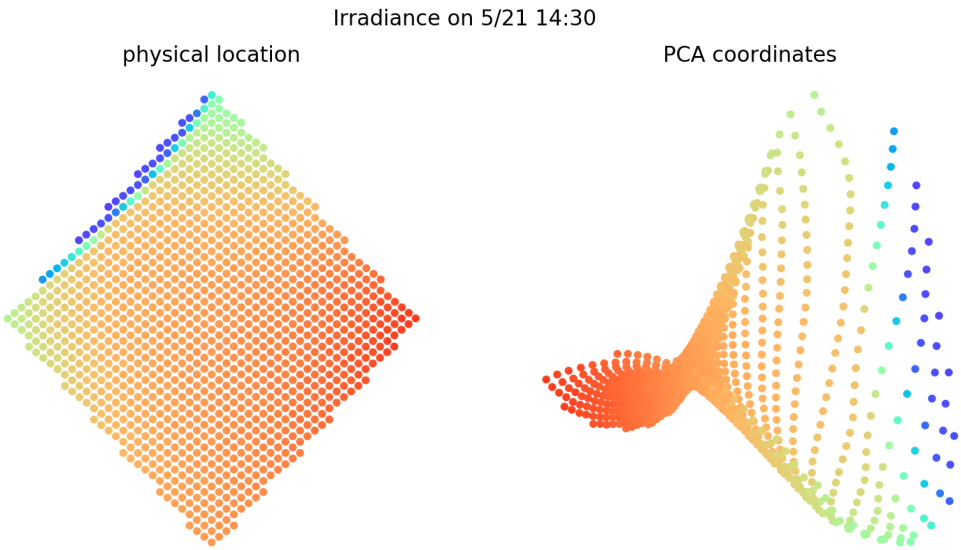
In [369]:
```
irrad_max = PV_irrad_sunny_days.max().max()
irrad_plot, axs = plt.subplots(3,4,figsize=(10,7.5))
irrad_plot.suptitle('PV irradiance for differnt times of day, plotted on PCA axis')
first_chart = 39
for i,ax in enumerate(axs.ravel()):
    ax.scatter(pca_irrad_sunny_days[:,0], pca_irrad_sunny_days[:,1], c=PV_irrad_sunny_days.iloc[:,i+fi
rst_chart],
               s=2, cmap='rainbow', vmin=0, vmax=irrad_max)
    ax.spines['left'].set_visible(False)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    ax.set_title(PV_irrad_sunny_days.columns[i+first_chart])
irrad_plot.figsize=(16,12)
```



Plotting side-by-side irradiance maps in physical location and PCA coordinates.

In [12]:
```
## this is a helper function to plot irradiance on real world location alongside PCA coordinates
def PV_double_scatter(dot_color, cmap='rainbow', vmin=0, vmax=1000, size=12, title=''):
    fig, axs = plt.subplots(1,2,figsize=(10,5))
    for i,ax in enumerate(axs.ravel()):
        ax.spines['left'].set_visible(False)
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.spines['bottom'].set_visible(False)
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)

    axs[0].scatter(PV_geometry['x'], PV_geometry['y'], c=dot_color,
                s=size, cmap=cmap, vmin=vmin, vmax=vmax)
    axs[0].set_title('physical location')

    axs[1].scatter(pca_irrad_sunny_days[:,0], pca_irrad_sunny_days[:,1], c=dot_color,
                s=size, cmap=cmap, vmin=vmin, vmax=vmax)
    axs[1].set_title('PCA coordinates')

    fig.suptitle(title)
```

In [13]:
```
column_to_plot=46
PV_double_scatter(PV_irrad_sunny_days.iloc[:,column_to_plot],
                title='Irradiance on ' + PV_irrad_sunny_days.columns[column_to_plot] )
```



Kmeans Clustering

In [14]:
```
from sklearn.cluster import KMeans
```

Each roofbay has 8 MPPTs which can accomodate between 35 and 220 PV panels. We want our clustering algorithm to create eight clusters.

In [15]:
```
n_clust = 8
```

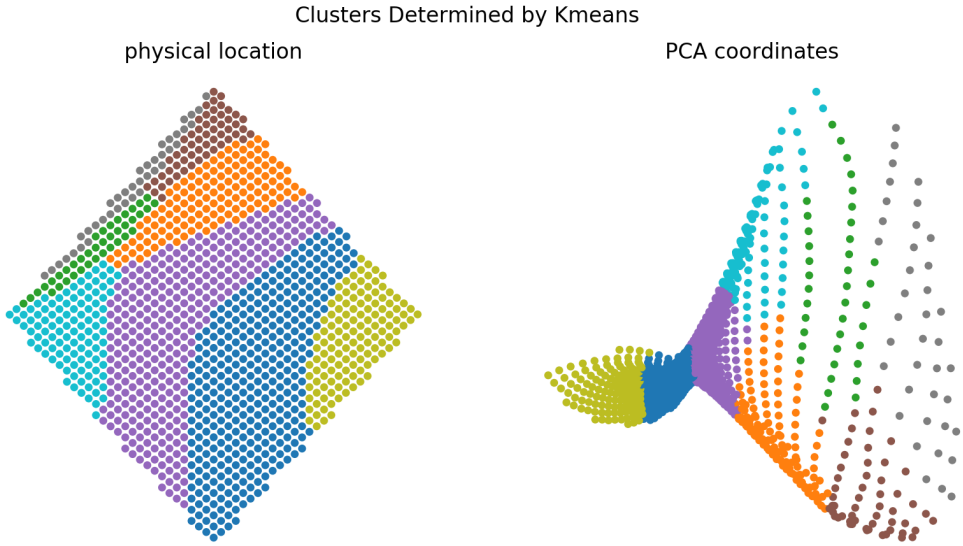
We feed the clustering algorithm the dataframe containing panel irradiance for 73 daylight hours on seven sunny days. The KMeans algorithm creates groups that minimize eucledian distance between each panel's irradiance vector and the centroid for the group. The result is eight groups of panels where each panels irradiance is closer to the center of the group to which it belongs than any other group. The grouped PV panels therefore have simmilar irradiance profiles to each other.

In [16]:
```
%%time
clust_km = KMeans(n_clusters=n_clust)
clust_km.fit(PV_irrad_sunny_days)
```

```
CPU times: user 647 ms, sys: 118 ms, total: 766 ms
Wall time: 132 ms
```

Out[16]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [17]:
```
PV_double_scatter(clust_km.labels_, cmap='tab10', vmax=7,
           title='Clusters Determined by Kmeans'
          )
```



The KMeans algortihm created groups in less than a second. However, there is no way to constrain the sizes of the groups with KMeans (or any other common clustering algorithm) and several of the created groups violate the size constraints of the MPPTs. For example, the group along the upper left edge has less than 35 panels, and the two groups in the center have more than 220 panels.

Iterative Reassignment

An iterative reassignment process was used to move PVs to other groups. The process is as follows:

1. Reassign panels as follows: a. In groups that exceed MPPT maximum, move the furthest PVs from the centroid (in 73-dimensional irracdiance space) to the next closest group. b. In groups that have too few panels, grab the next closest PVs from other groups.
2. Recalculate centroids of all clusters
3. Recalculate distance from all PV panels to all cluster centroids
4. Reassign all panels to the group with the closes centroid.

The last step moves back towards the original KMeans result, but we would like a solution where panels are in a prefered group, however we don't continue to calculate new centroid and reasign repeatedly, because that eventually returns us to the Kmeans result.

We also use an iteratively incremented value that causes the process to reassign one more panel in each iteration. This step acts to prevent the process from getting stuck at an equilibrium that repeats the same result without meeting the constraints.

Cluster size constraints

In [18]:
```
cluster_minimum = 35
cluster_maximum = 220
```

Helper Functions

The following functions are used to perform the steps in the iterative reassignment process.

In [19]:
```
def meets_size_constraints(cluster_sizes, min_size, max_size):
    '''checks each cluster to ensure it meets minimum and maximum size constraints.
       Returns True if all clusters meet constraints, otherwise False'''
    meets_constraints = [ a[0] >= min_size and a[0] <= max_size for a in cluster_sizes ]
    return(all(meets_constraints))

def calc_cluster_distances(datapoints, centers):
    '''calculates euclidean distance from data points to all cluster centers.
       returns numpy array of distances.
       datapoins is an MxD array
       centers in an NxD array
       returns an MxN array of distances from points (M) to centers (N)
       vectorized implementation borrowed from:
       https://medium.com/@souravdey/l2-distance-matrix-vectorization-trick-26aa3247ac6c'''
    squared_distances = (-2 * np.dot(datapoints, centers.T)
                         + np.sum(centers**2, axis=1)
                         + np.sum(datapoints**2, axis=1)[:, np.newaxis])
    distances = squared_distances ** 0.5
    return(distances)

def gather_more(distances, cluster_index, new_total):
    '''sets the distance of new_total closest points to centroid to zero so that the points are asigne
d
       to that centroid, increasing the number of points in a cluster to new_total'''
    ordered = sorted(distances[:,cluster_index])
    cutoff = ordered[new_total]
    distances[ distances[:,cluster_index] < cutoff, cluster_index ] = 0
    return(distances)

def shed_some(distances, cluster_index, new_total):
    '''sets the distance of points further than the new_total number of closest points to a infinity
       so that these points are asinged to a different cluster, liminting the number of points in the
       selected cluster to new_total'''
    ordered = sorted(distances[:,cluster_index])
    cutoff = ordered[new_total]
    distances[ distances[:,cluster_index] >= cutoff, cluster_index ] = np.inf
    return(distances)

def assign_to_closest(distances):
    '''takes the index of the shortest distance for each data point and assigns the data point to that
cluster'''
    cluster_id = np.array([ d.argmin() for d in distances ])
    return(cluster_id)

def calc_cluster_centers(datapoints, num_clusters, cluster_id):
    '''calculates new cluster centroids by averaging datapoint coordinates assigned to a cluster '''
    centroids = [[ np.mean(d) for d in datapoints[ cluster_id == cls_id].T ] for cls_id in range(num_c
lusters) ]
    return(np.array(centroids))

def calc_inertia(distances, cluster_id):
    '''calculates the inertia (summed squared distance of datapoints to its assigned cluster)'''
    square_distances = [ dist[cls_id]**2 for dist, cls_id in zip(distances, cluster_id)]
    inertia = np.sum(square_distances)
    return(inertia)

def calc_cluster_sizes(cluster_ids, num_clusters):
    cluster_sizes = np.array([ [sum( cluster_ids == i ),i] for i in range(num_clusters)])
    return(cluster_sizes)
```

In [25]:
```
#assign the dataframe to a numpy array to use vectorization
clustering_datapoints = PV_irrad_sunny_days.to_numpy()

#initalize the clusters with the kmeans result
cluster_ids = clust_km.labels_
cluster_centroids = clust_km.cluster_centers_

#calculate a distance matrix
cluster_distances = calc_cluster_distances(clustering_datapoints, cluster_centroids)

#calculate the initial cluster sizes
cluster_sizes = calc_cluster_sizes(cluster_ids, n_clust)

#calculate the initial inertia
inertia = calc_inertia(cluster_distances, cluster_ids)

#print sizes and initial inertia
print(cluster_sizes[:,0], inertia)
```

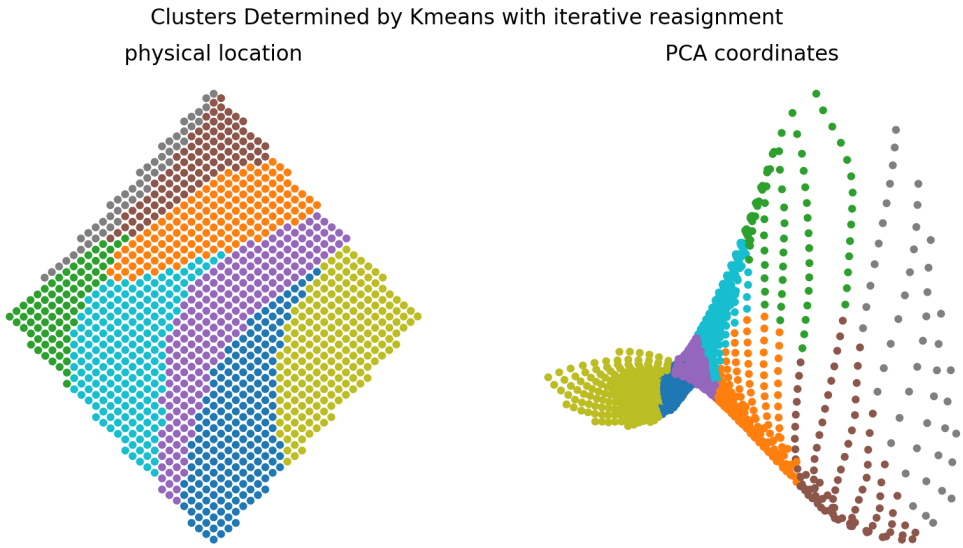
```
[356 131  39 323  55  39 126 106] 92825021.64898461
```

In [26]:
```
%%time
#This cell performs the iterative reassignment.
#itr keeps track of how many iterations we've done.
itr=0

#while constrains are not met (and iterator is less than the maximum size)
while not meets_size_constraints(cluster_sizes, cluster_minimum, cluster_maximum) and itr < cluster_ma
ximum:

    for clust in cluster_sizes:
        if clust[0] > cluster_maximum:
            #if cluster exceeds max size, reduce size of cluster to (maximum - itr)
            cluster_distances = shed_some(cluster_distances, clust[1], cluster_maximum-itr)

        elif clust[0] < cluster_minimum:
            #if cluster is less than min size, increase size of cluster to (minimum + itr)
            cluster_distances = gather_more(cluster_distances, clust[1], cluster_minimum+itr)

        #modifying max and min size with itr allows us to slowly increase the severity of the cluster
        #reassignment avoiding getting stuck in equilibrium.

    cluster_ids = assign_to_closest(cluster_distances)
    cluster_centroids = calc_cluster_centers(clustering_datapoints, n_clust, cluster_ids)
    cluster_distances = calc_cluster_distances(clustering_datapoints, cluster_centroids)
    pre_inertia = calc_inertia(cluster_distances, cluster_ids)
    cluster_ids = assign_to_closest(cluster_distances)

    cluster_sizes = calc_cluster_sizes(cluster_ids, n_clust)
    inertia = calc_inertia(cluster_distances, cluster_ids)

    print(itr, cluster_sizes[:,0], f'{pre_inertia:0.1f}', f'{inertia:0.1f}')
    itr+=1
```

```
0 [267 154  41 296  67  39 187 124] 115780833.3 101123887.9
1 [263 163  43 276  74  39 187 130] 112865927.6 101980885.2
2 [260 171  43 259  79  39 189 135] 113898437.1 103751981.1
3 [261 174  43 250  83  39 189 136] 114244850.1 104630556.6
4 [261 176  43 244  87  39 189 136] 114733481.7 105300097.4
5 [265 175  44 240  93  39 186 133] 114442899.7 105415460.3
6 [270 179  45 230  96  40 186 129] 114023095.0 105741533.1
7 [273 178  48 226  96  42 186 126] 113802016.1 105775204.3
8 [274 181  49 221  95  45 186 124] 113537856.8 105618303.6
9 [278 180  51 216  96  46 188 120] 113097458.7 105561013.6
10 [266 156  50 270  92  46 182 113] 104946065.3 101082744.3
11 [264 168  52 237  94  46 191 123] 111947266.3 104502132.6
12 [266 172  53 227  96  46 191 124] 112733754.6 105360267.5
13 [270 174  53 221  97  46 190 124] 113137982.6 105841278.3
14 [271 178  53 215  98  46 191 123] 113430316.4 106103063.0
15 [263 156  51 269  93  46 183 114] 105468784.3 101579556.7
16 [256 168  53 241  94  46 192 125] 113027895.4 105110268.2
17 [261 172  52 228  97  46 192 127] 113702716.8 105999477.0
18 [266 177  52 217  99  46 192 126] 113877249.8 106396562.6
19 [261 156  51 268  94  46 183 116] 105691197.7 101776209.7
20 [250 167  52 245  95  46 192 128] 113332111.9 105472847.3
21 [259 171  53 229  97  46 192 128] 114268920.8 106344926.0
22 [268 177  53 214  99  46 192 126] 114537439.1 106877637.4
23 [260 156  51 265  94  46 184 119] 105951780.0 101954241.9
24 [248 168  52 244  95  46 193 129] 113734276.7 105671602.0
25 [254 171  53 231  97  46 192 131] 114803334.4 106704917.8
26 [256 176  53 225  99  46 192 128] 115001620.5 107148192.5
27 [259 177  53 222 100  46 192 126] 114721494.0 107152053.9
28 [263 180  53 214 101  46 192 126] 114649918.3 107213130.0
29 [256 157  51 269  95  46 182 119] 106254204.6 102255319.3
30 [246 170  52 245  95  46 192 129] 113825129.1 105865559.9
31 [252 175  53 229  97  46 192 131] 115412501.3 107081236.5
32 [255 177  54 224  99  46 192 128] 115583720.1 107526241.9
33 [259 179  54 218 100  46 192 127] 115444370.9 107633350.6
34 [253 156  52 269  95  46 184 120] 106415428.6 102310258.7
35 [245 168  53 244  97  46 192 130] 114415498.2 106161665.9
36 [252 174  54 228  99  46 192 130] 115567130.7 107398182.3
37 [251 177  54 224 100  46 192 131] 115950602.7 107927208.9
38 [255 180  54 217 101  46 192 130] 115971733.5 108057068.5
39 [249 157  52 269  95  46 184 123] 106824721.2 102596493.2
40 [241 169  53 245  97  46 192 132] 115271236.7 106610903.1
41 [249 175  54 227  99  46 192 133] 116100356.5 107750515.2
42 [250 179  54 220 100  46 192 134] 116363498.7 108227710.1
43 [240 156  52 280  95  46 183 123] 106771181.2 102551661.3
44 [239 169  54 243  97  46 192 135] 115960268.1 106979266.3
45 [241 175  54 229  99  46 193 138] 116858509.0 108256195.2
46 [246 179  54 223 100  46 192 135] 116844685.0 108518348.5
47 [247 182  54 219 101  46 192 134] 116918672.6 108630223.2
48 [246 161  52 271  95  46 181 123] 107235920.8 102837065.0
49 [239 171  54 240  97  46 192 136] 116407879.6 107382750.7
50 [237 176  54 229  99  46 196 138] 117461010.4 108651220.6
51 [243 181  54 220 100  46 195 136] 117671002.9 108990088.1
52 [238 158  52 279  95  46 183 124] 107139102.4 102784080.9
53 [234 171  54 242  97  46 195 136] 116928054.5 107582041.0
54 [233 176  54 231  99  46 198 138] 117753373.6 108826141.3
55 [239 181  54 222 100  46 196 137] 118134818.5 109285609.9
56 [241 181  54 219 101  46 196 137] 118216140.6 109423389.1
57 [235 161  52 277  95  46 185 124] 107683453.2 103159570.6
58 [229 172  54 243  97  46 197 137] 117602360.4 108043899.3
59 [229 178  54 228 100  46 201 139] 118416201.5 109346871.6
60 [233 181  54 220 101  46 201 139] 118878635.3 109914401.6
61 [233 159  52 279  95  46 186 125] 107963433.8 103400836.9
62 [222 172  54 245  97  46 200 139] 118314731.9 108435766.7
63 [226 178  54 226  99  46 205 141] 118911286.9 109768424.0
64 [230 181  54 217 100  46 206 141] 119579515.9 110365117.6
65 [229 158  52 281  95  46 189 125] 108300568.7 103723961.5
66 [217 172  54 244  97  46 205 140] 118832637.6 108828874.9
67 [291 167  54 211  95  46 175 136] 108671933.9 103472595.9
68 [241 154  51 282  92  46 177 132] 106496414.9 101972872.3
69 [207 170  55 246  95  46 204 152] 121540107.7 109760636.6
70 [290 164  54 208  94  47 175 143] 108933516.9 103808687.9
71 [240 155  51 277  91  47 177 137] 106615136.5 102213520.0
72 [205 169  55 241  95  47 205 158] 122128723.3 110005227.6
73 [286 162  53 210  95  47 175 147] 108958214.6 104013503.5
74 [237 151  52 278  92  47 176 142] 106523867.5 102291935.4
75 [200 168  57 242  95  47 204 162] 122726796.7 110428722.2
```

```
76 [278 162  55 210  95  47 175 153] 109357443.0 104547915.9
77 [227 152  52 285  93  47 173 146] 106776775.2 102771696.0
78 [194 168  58 241  97  47 205 165] 123187150.9 110813784.0
79 [277 162  58 205  96  48 176 153] 109407761.7 104787692.4
80 [225 152  54 279  94  49 172 150] 106885896.6 103012103.4
81 [191 166  64 237  98  49 204 166] 122482311.8 110205568.0
82 [274 159  68 202  97  49 175 151] 108738558.5 104615397.4
83 [227 150  67 267  95  49 170 150] 106471388.4 103006323.5
84 [191 161  74 221  98  49 203 178] 120419575.1 109566691.7
85 [260 157  73 204  96  49 173 163] 108228089.7 104995968.0
86 [219 148  70 264  95  49 169 161] 106316433.3 103233405.1
87 [259 151  70 225  94  49 161 166] 106319118.5 104036894.5
88 [201 150  79 201  95  49 205 195] 120601200.6 110298561.0
CPU times: user 13.8 s, sys: 2.38 s, total: 16.2 s
Wall time: 2.75 s
```

In [27]:
```
PV_double_scatter(cluster_ids, cmap='tab10', vmax=7,
           title='Clusters Determined by Kmeans with iterative reasignment'
          )
```



DBScan Clustering

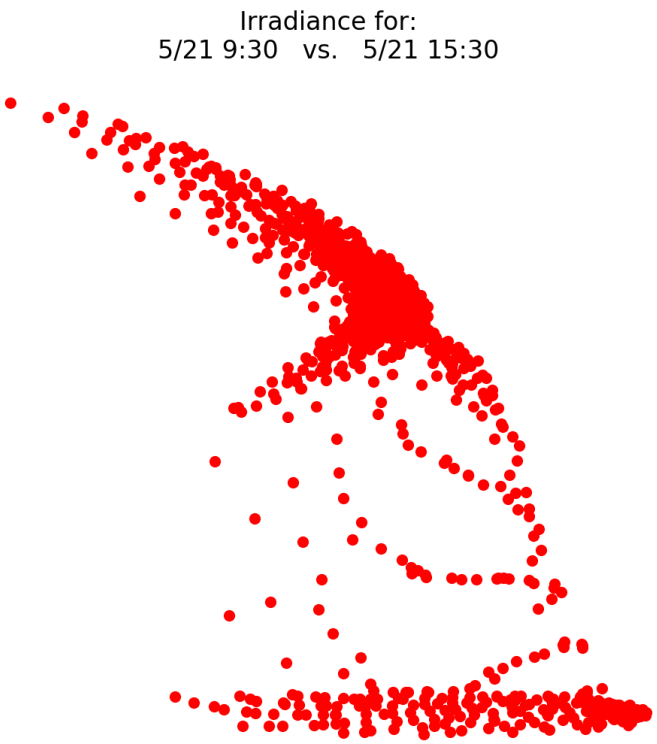
In [22]:
```
from sklearn.cluster import DBSCAN
```

Someone suggested exploring DBscan as a method to cluster based on irradiance profile. DBscan finds clusters of higher density sepearted by regions of lower density. The plot below shows irradiance for two times (9:30 and 15:30) on May 21. You can see two distinct clusters of values, the group with low y-values result from a shadow on the roof at 15:30. DBscan easily seperates these into two clusters, so you can see how DBscan could potentially be a good method for clustering based on shadow pattern.

However, when you look at the PCA diagram above there is a region of high density and a region of low density. The shadow moves across the roof slowly hour by hour, and taken in totality, there aren't multiple regions of high density. The result with DBscan on the 73-dimensional dataset is a single large cluster, with two small clusters along the top left edge.

Another problem with DBscan for this application is the inability to pre-determine the number of clusters. The system design has 8 electrical groups, so we want exactly that number of clusters.

In [23]:
```
PV_scatter(PV_irrad_sunny_days.iloc[:,42], PV_irrad_sunny_days.iloc[:,48],
           color='red',
           size=20,
           vmax=irrad_max,
           title='Irradiance for:\n'+PV_irrad_sunny_days.columns[41]+'   vs.   '+PV_irrad_sunny_days.c
olumns[47])
```

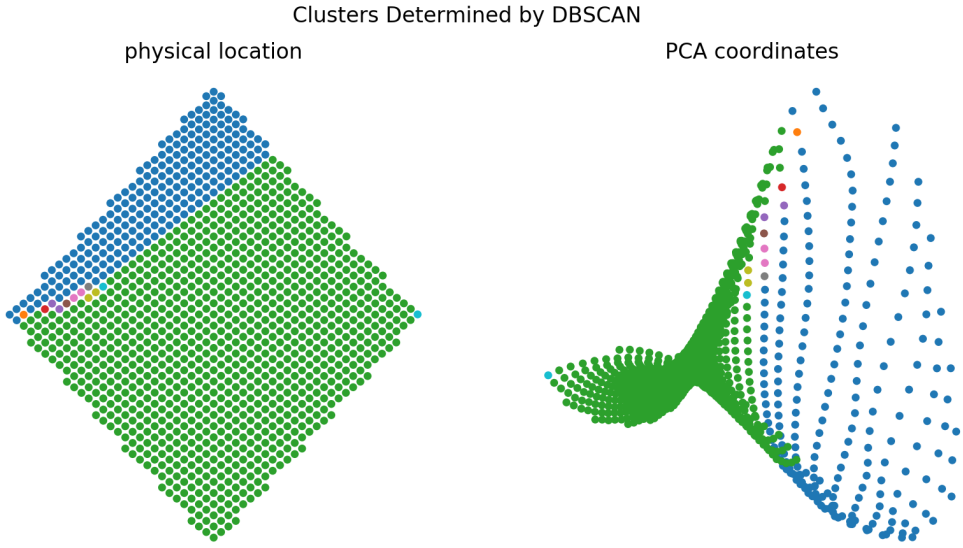


In [28]:
```
clust_db2d = DBSCAN(eps=20, min_samples=1)
clust_db2d.fit(np.array([PV_irrad_sunny_days.iloc[:,42], PV_irrad_sunny_days.iloc[:,48]]).T)
```

Out[28]:
```
DBSCAN(algorithm='auto', eps=20, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=1, n_jobs=None, p=None)
```

In [30]:
```
PV_scatter(PV_irrad_sunny_days.iloc[:,42], PV_irrad_sunny_days.iloc[:,48],
           color=clust_db2d.labels_, cmap='tab10', vmax=clust_db2d.labels_.max(),
           title='Clusters Determined by DBSCAN')
```

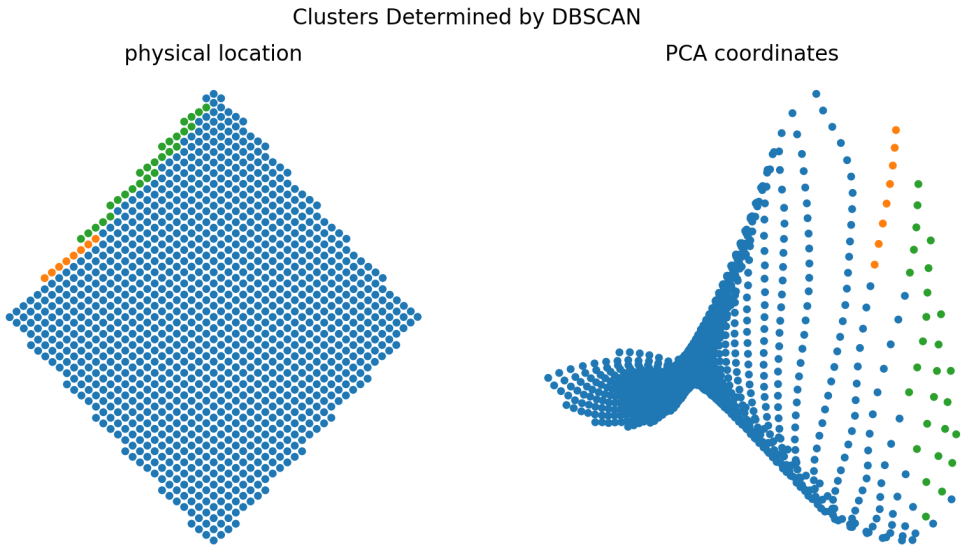
In [49]:
```
PV_double_scatter(clust_db2d.labels_, cmap='tab10', vmax=clust_db2d.labels_.max(), title='Clusters Det
ermined by DBSCAN')
```



In [31]:
```
clust_db = DBSCAN(eps=300, leaf_size=30, min_samples=5)
clust_db.fit(PV_irrad_sunny_days)
```

Out[31]:
```
DBSCAN(algorithm='auto', eps=300, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=5, n_jobs=None, p=None)
```

In [32]:
```
PV_double_scatter(clust_db.labels_, cmap='tab10', vmax=7, title='Clusters Determined by DBSCAN')
```



Optimization

The other way to potenitally create groups is through optimization. The benefit of optimization is that you can include group size constraints as a penalty in the fitness function.

Below are two half-hearted attempts at generating groups with differential evolution optimizer. I could probably get a better result by improving the fitness function. I could probably improve the result by writing a discrete optimizer, or using a package with one. But based on these initial result, I don't think its worth the effort. These optimizations take hours to give a not good result.

Optimization on cluster assignments

First trying a genetic algorithm using a continuous variable between 0-8. The group is asigned by casting to integers using floor to determine groupings. Very much a cludge but worth a try.

In [33]:
```
from scipy.optimize import differential_evolution
```

In [34]:
```
cluster_ids = np.random.randint(0,n_clust,(1175))
bounds = [ (0,7.9999) for i in range(len(cluster_ids))]
```

In [35]:
```
clustering_datapoints = PV_irrad_sunny_days.to_numpy()
```

In [36]:
```
def cost_function(cluster_floats):
    '''cost function for optimization, returns inertia multiplied by a penalty'''
    global clustering_datapoints, n_clust, cluster_minimum, cluster_maximum
    cluster_ids = np.array(np.floor(cluster_floats), dtype=np.int)
    cluster_centroids = calc_cluster_centers(clustering_datapoints, n_clust, cluster_ids)
    cluster_distances = calc_cluster_distances(clustering_datapoints, cluster_centroids)
    cluster_sizes = calc_cluster_sizes(cluster_ids, n_clust)
    inertia = calc_inertia(cluster_distances, cluster_ids)
    if not meets_size_constraints(cluster_sizes, cluster_minimum, cluster_maximum):
        inertia *= 2
    return inertia
```

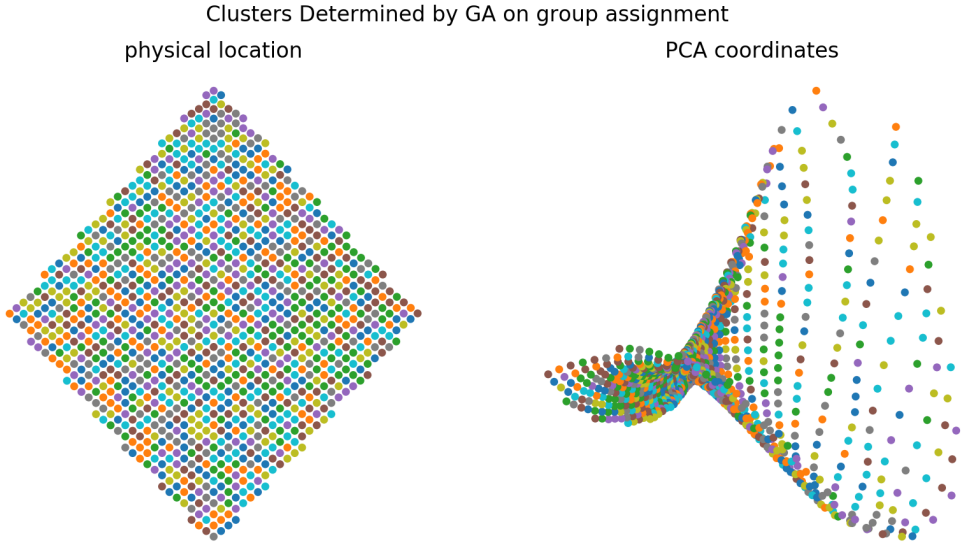
In [37]:
```
%%time
ga_result = differential_evolution(cost_function, bounds, maxiter=1000, popsize=14, workers=7 )
```

```
/usr/local/lib/python3.7/site-packages/scipy/optimize/_differentialevolution.py:494: UserWarning: di
fferential_evolution: the 'workers' keyword has overridden updating='immediate' to updating='deferre
d'
  " updating='deferred'", UserWarning)

CPU times: user 3min 39s, sys: 25.9 s, total: 4min 5s
Wall time: 6min 17s
```

In [38]:
```
ga_result
```

Out[38]:
```
     fun: 580627065.1676188
 message: 'Optimization terminated successfully.'
    nfev: 34076
     nit: 1
 success: True
       x: array([1.65226717, 3.02917196, 0.70802986, ..., 0.70396495, 1.48712337,
       4.73774523])
```

In [39]:
```
PV_double_scatter(np.floor(ga_result.x), cmap='tab10', vmax=7,
           title='Clusters Determined by GA on group assignment'
          )
```



Ha Ha! Yeah, no. The fitness function didn't improve much beyond the random assignment.

Optimization on location of cluster centroids

This attempt instead tries to find cluster centroids with that minimize the cost function. there are 8*73 values to optimize.

In [40]:
```
cl_cent_shape = np.shape(clust_km.cluster_centers_)
ga_centroids = clust_km.cluster_centers_.ravel()
```

In [43]:
```
bounds2 = np.array([[ (min, max) for min, max in zip(PV_irrad_sunny_days.min(), PV_irrad_sunny_days.ma
x()) ]
                    for i in range(cl_cent_shape[0]) ]).reshape(len(ga_centroids),2)
```

In [44]:
```
def cost_function2(ga_centroids_flat):
    '''cost function for optimization, returns inertia multiplied by a penalty'''
    global clustering_datapoints, n_clust, cluster_minimum, cluster_maximum, cl_cent_shape
    ga_centroids = ga_centroids_flat.reshape(cl_cent_shape)
    ga_distances = calc_cluster_distances(clustering_datapoints, ga_centroids)
    ga_ids = assign_to_closest(ga_distances)
    ga_sizes = calc_cluster_sizes(ga_ids, n_clust)
    inertia = calc_inertia(ga_distances, ga_ids)
    if not meets_size_constraints(ga_sizes, cluster_minimum, cluster_maximum):
        inertia = inertia * 2
    return inertia
```

In [45]:
```
%%time
ga2_result = differential_evolution(cost_function2, bounds2, maxiter=100000, popsize=14, workers=7)
```

```
CPU times: user 1h 2min 38s, sys: 6min 8s, total: 1h 8min 47s
Wall time: 6h 14s
```

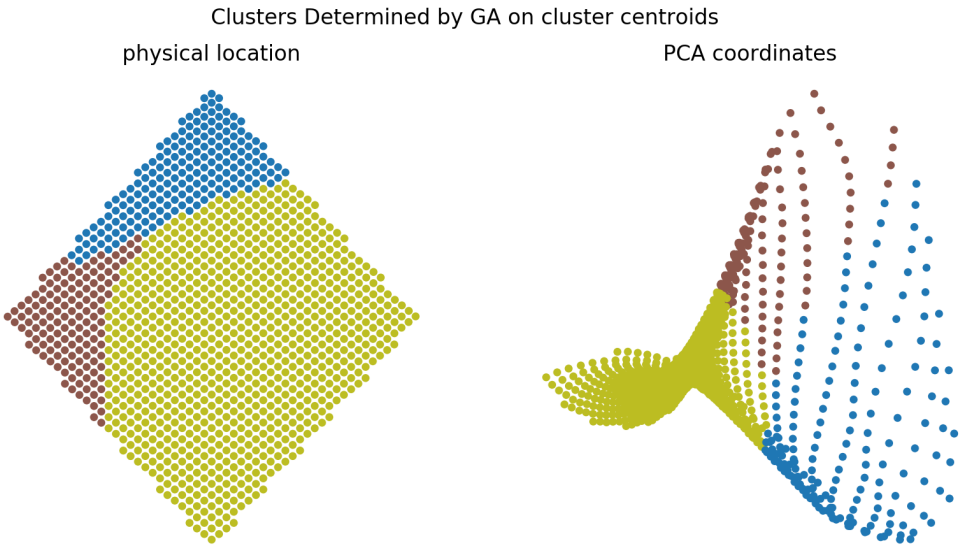
In [46]:
```
ga2_result
```

Out[46]:
```
      fun: 579735657.819385
  message: 'Optimization terminated successfully.'
     nfev: 2559116
      nit: 310
  success: True
        x: array([ 411.3844197 ,  603.37649203,  593.95326208,  613.22383566,
         497.8370716 ,  290.50912182,  123.33831052,  515.55741815,
         720.48270597,  755.75503544,  744.02251019,  414.26986193,
          69.54773627,   65.51172232,   45.3269311 ,  234.8244983 ,
         505.65358583,  685.07120865,  862.19597622,  849.31272721,
         878.27864039,  714.79943199,  501.49962434,  200.80696655,
          84.38609673,  117.8144436 ,  239.0933588 ,  465.75738377,
         691.65381829,  821.86002318,  898.74114475,  961.6457097 ,
         898.20241094,  831.74547086,  518.83767953,  230.74284773,
         130.06683952,  125.53919988,  245.27633209,  475.52478948,
         707.45845673,  837.47830876,  975.5117338 ,  963.30345186,
         969.77724033,  815.5513627 ,  576.38036095,  332.30905509,
         117.37409557,   75.69961318,   89.5301162 ,  317.63288505,
         693.02501868,  830.08274186,  960.56605825,  927.40884281,
         931.00405739,  776.07431979,  595.40847855,  336.68733673,
         105.84875541,   98.92465941,   78.35348548,   24.69640614,
         233.0384003 ,  455.47961672,  480.13020968,  535.92087258,
         531.4675718 ,  376.81972575,  159.0472692 ,  107.12104173,
          25.6358569 ,  369.05250378,  441.13838213,  391.02682584,
         387.92516812,  351.3562583 ,  506.43094921,  209.49362846,
         414.32621899,  559.81372516,  568.47686939,  575.27186665,
         354.32235041,   92.3084187 ,  174.64174319,   41.56240198,
         179.50400414,  327.08702492,  640.22311503,  942.08771862,
         768.9912187 ,  762.16540135,  744.41847487,  343.83782056,
         448.45901131,  193.48860267,   98.85138755,  219.9525682 ,
         262.52955394,  516.39027446,  888.3277739 ,  950.4261031 ,
         930.88297864,  876.97290297,  930.37694221,  658.82139322,
         538.52992945,  406.07297025,   86.19186775,  114.91545573,
         561.22974474,  662.16004307,  802.55276607,  821.51215036,
         959.34157342, 1005.32305744,  907.95472214,  514.71423476,
         160.18967186,  371.82465712,  216.36539189,   86.18588484,
         323.00926647,  605.79875346,  712.35785331,  858.25134825,
         909.29046107,  915.01565022,  787.46013354,  670.72937999,
         290.88951201,  347.62215916,  200.02423337,   91.00988237,
          60.77430805,   99.96799885,  470.40132585,  434.77732254,
         218.66815796,  385.15857283,  239.19748143,  435.12495303,
         120.15936021,   46.7546064 ,  402.98555151,  518.48778872,
         403.18921725,  382.92962958,  385.7807007 ,  436.06532758,
         253.98020272,  434.03234683,  786.25243839,  768.03419787,
         414.85007337,  479.70943931,  534.55037462,  206.85048365,
          26.12304411,  173.58547169,  428.62480808,  566.46460782,
         893.22186232,  787.23969981,  716.05544558,  713.08969263,
         407.18164681,  327.49957473,  184.39071901,  114.71457455,
         212.41073706,  390.77064668,  773.08367563,  688.32251954,
         910.33984521,  816.67415008,  771.53862528,  798.6234567 ,
         843.55768727,  227.29376125,  334.39889724,  201.85777236,
         288.10356542,  306.44763806,  419.2502874 ,  797.60712361,
         933.75120059,  948.67053169,  948.75849112,  772.19273633,
         473.25492878,  490.58842864,  315.82028269,  277.37781283,
          84.24579473,  262.66590626,  429.01207739,  741.56956715,
         922.46835957,  961.89672536,  873.40352124,  854.31318982,
         854.18876444,  306.20190496,  237.72189533,  190.37018527,
          62.48584092,   64.25065519,  283.69597513,  390.53271405,
         480.23577619,  344.27822   ,  499.84924138,  227.21862673,
         230.99542525,  337.67520195,   35.87060266,  401.0073017 ,
         541.38051524,  290.96178186,  296.1080391 ,  359.39585536,
         405.66996676,  173.92473877,  472.38180137,  646.71938246,
         512.21102108,  711.45019143,  708.37805847,  253.62221674,
         154.22903659,   32.06072331,  191.14495778,  303.30039173,
         595.98561965,  723.61613469,  808.94125032,  738.73830729,
         560.61771927,  137.09767312,  334.93692811,  414.84485607,
          73.13204374,  168.50346819,  146.83637664,  366.47692271,
         742.08683059,  897.39096224,  964.63775353,  850.11370423,
         802.11737023,  833.47052988,  550.21492014,  371.72745585,
         248.70778938,  199.64438564,  293.11160301,  585.61915767,
         835.589021  ,  898.86000941,  854.87182102,  900.97090019,
         872.93614374,  656.08530964,  406.87391972,  538.18465147,
         319.63848505,   94.45021395,  263.12038788,  527.81157325,
         636.3489249 ,  874.17981462,  879.60019447,  906.50869486,
         818.83591526,  419.98901381,  649.32520385,  446.23968237,
         119.0469015 ,   71.78982667,   15.87488172,  309.79682024,
```

```
525.35390205,  362.1878062 ,  455.25428924,  639.99377349,
318.03965115,  451.91309949,  239.65540724,   51.93532233,
288.7784224 ,  376.23552294,  332.28817253,  317.26448117,
395.45548717,  253.58300554,  145.83556334,  419.78140517,
589.1813571 ,  576.22787074,  529.21194998,  362.92994819,
156.77172307,   87.7710615 ,   38.89383997,  110.23929618,
433.4283357 ,  542.16496545,  739.63579776,  738.71703853,
701.91158981,  618.90071627,  444.99688145,  314.0934092 ,
138.43669197,   98.29561978,  158.11418179,  362.34142589,
520.7744608 ,  677.36753792,  823.91445214,  831.60492783,
800.14962779,  758.13731279,  630.03666136,  426.94714328,
207.96590115,  127.37073344,  271.3963325 ,  475.5682339 ,
668.87814234,  716.18687845,  820.84263435, 1006.16594982,
927.09180024,  806.68337211,  667.24377757,  542.43121309,
322.32512672,  140.29436698,   83.07011546,  350.24763803,
650.55101839,  741.96113851,  837.75050822,  900.67786381,
904.84734476,  762.11302267,  711.78668856,  529.40644604,
285.31508586,  216.98513134,   81.95983079,   52.41522272,
113.98321223,  180.24634203,  326.46783576,  324.5653068 ,
351.6710774 ,  273.64388071,  189.04980909,   98.14490488,
 56.8853113 ,  353.96830608,  525.93967392,  500.42125207,
555.04978238,  508.7671925 ,  322.70079498,   80.14987971,
417.08029704,  684.64507783,  446.39698395,  558.70200221,
215.31166201,  113.78570319,  189.28181732,   52.42274135,
167.56757019,  463.38251407,  726.23775054,  729.07147588,
809.86509967,  734.84875346,  693.02991937,  562.9632521 ,
330.9777699 ,  265.91936459,   96.41560896,  181.37508268,
478.80460645,  610.11528238,  799.20706545,  869.45529484,
925.33760567,  820.73705332,  776.01337628,  640.52504865,
689.72942681,  562.53298661,  184.85320742,  203.01729178,
230.04565532,  517.79930395,  816.25360475,  919.40168807,
976.55413345,  969.53141112,  828.28186395,  493.49717933,
725.23643424,  332.77756353,  144.32239619,   89.24865485,
323.22072028,  662.20716322,  870.63434228,  865.08082879,
886.8951827 ,  943.82234642,  805.6198258 ,  246.61647614,
442.03520531,  367.01441234,  193.66581696,   85.03882723,
 38.38251007,  161.29341935,  355.05898041,  532.06860546,
592.198609  ,  477.93225241,  247.22878556,  400.73392422,
354.02993085,   40.78745608,  307.10344126,  403.93787327,
499.80705098,  539.50277597,  507.47009879,  417.73835735,
220.84198513,  416.91875046,  642.20642326,  714.34343188,
694.51056217,  585.21439419,  395.75825137,  183.86919893,
 21.39717502,  159.29378516,  390.83437552,  559.89121462,
746.34771369,  801.71444826,  843.48240418,  768.50165085,
650.82148849,  505.62260106,  328.53522123,  110.28888161,
132.52200468,  318.86402167,  528.93787969,  708.14043901,
823.70695018,  925.42967305,  948.06952798,  873.45046374,
736.20334929,  604.74462484,  356.09049888,  145.03160695,
188.16814374,  392.37333762,  604.04606103,  770.13605059,
880.9086852 ,  968.96334583,  973.92270255,  907.33667178,
817.77343941,  642.5585396 ,  493.77818251,  239.57073504,
 79.81283159,  304.95767042,  559.33051746,  736.49209154,
875.36942249,  937.06444527,  969.58774615,  886.59998255,
815.63706993,  650.39768581,  485.18681427,  252.36605493,
 85.72048874,   20.93410925,  116.31079786,  269.66640708,
404.17712198,  458.58080015,  488.82767706,  432.28613197,
348.31182826,  208.68953846,   31.42006249,  384.95551902,
546.28374784,  468.86500858,  498.35793764,  223.66003698,
213.34121074,  283.6945613 ,  440.65987943,  710.88571349,
727.9236575 ,  628.97179231,  252.03484628,  448.17602903,
361.11181735,   63.88848846,  221.3648557 ,  445.35701347,
604.65308628,  602.27167246,  843.75382775,  754.0065655 ,
734.66116066,  821.30679568,  444.92381772,  282.90848099,
 87.56932349,  134.23774602,  191.22461856,  691.32652281,
658.90639654,  842.89441906,  816.48608315,  844.19988723,
870.84455698,  748.4411412 ,   86.22029556,  471.22884083,
282.37120785,  117.60750269,  250.88613602,  623.33949081,
752.03303514,  778.38349519,  924.26817097,  929.75916814,
939.47723599,  430.28265311,  706.51444353,  261.62726063,
251.12510521,   92.54978433,  269.14945576,  384.87370255,
668.63260205,  819.80002138,  870.42587076,  864.18546473,
863.65825878,  439.60247961,  396.90284968,  311.24066463,
222.38998211,   65.67149438,   41.28101677,  172.35444931,
422.86322638,  513.60409684,  503.36307791,  526.31431279,
448.88560874,  258.93261742,   53.83704212,   42.93654662])
```

In [47]:
```
ga_distances = calc_cluster_distances(clustering_datapoints, ga2_result.x.reshape(cl_cent_shape))
ga_ids = assign_to_closest(ga_distances)
ga_sizes = calc_cluster_sizes(ga_ids, n_clust)
inertia = calc_inertia(ga_distances, cluster_ids)
cost = cost_function2(ga2_result.x)
print(inertia, cost)
PV_double_scatter(ga_ids, cmap='tab10', vmax=7,
           title='Clusters Determined by GA on cluster centroids'
          )
```

```
1643462134.3456652 579735657.819385
```



In [ ]:
```

```