

Real Micro Crystals - Data Engineering & Exploration 3

explore larger data set

Michael Janus, June 2018

Use the functions on a real (small) data set.

For explanation and how to usage functions, see the notebook `imgutils_test_and_explain.ipynb`

1. Import the used modules, including the one with test functions:

```
In [86]: import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

import matplotlib.pyplot as plt
from scipy import stats

import imgutils
import imgutils_test as tst
```

```
In [87]: # Re-run this cell if you altered imgutils or imgutils_test
import importlib
importlib.reload(imgutils)
importlib.reload(tst)
```

```
Out[87]: <module 'imgutils_test' from 'C:\\JADS\\SW\\Grad Proj\\realxtals1\\sources\\imgutils_test.py'>
```

1. Get image files

```
In [88]: df_imgfiles = imgutils.scanimgdir('../data/Crystals_Apr_12/Tileset6', '.tif')
print(df_imgfiles)
```

```

                                filename
0  ..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...
1  ..\data\Crystals_Apr_12\Tileset6\Tile_001-002-...
2  ..\data\Crystals_Apr_12\Tileset6\Tile_001-003-...
3  ..\data\Crystals_Apr_12\Tileset6\Tile_001-004-...
4  ..\data\Crystals_Apr_12\Tileset6\Tile_001-005-...
5  ..\data\Crystals_Apr_12\Tileset6\Tile_002-001-...
6  ..\data\Crystals_Apr_12\Tileset6\Tile_002-002-...
7  ..\data\Crystals_Apr_12\Tileset6\Tile_002-003-...
8  ..\data\Crystals_Apr_12\Tileset6\Tile_002-004-...
9  ..\data\Crystals_Apr_12\Tileset6\Tile_002-005-...
10 ..\data\Crystals_Apr_12\Tileset6\Tile_003-001-...
11 ..\data\Crystals_Apr_12\Tileset6\Tile_003-002-...
12 ..\data\Crystals_Apr_12\Tileset6\Tile_003-003-...
13 ..\data\Crystals_Apr_12\Tileset6\Tile_003-004-...
14 ..\data\Crystals_Apr_12\Tileset6\Tile_003-005-...
15 ..\data\Crystals_Apr_12\Tileset6\Tile_004-001-...
16 ..\data\Crystals_Apr_12\Tileset6\Tile_004-002-...
17 ..\data\Crystals_Apr_12\Tileset6\Tile_004-003-...
18 ..\data\Crystals_Apr_12\Tileset6\Tile_004-004-...
19 ..\data\Crystals_Apr_12\Tileset6\Tile_004-005-...
20 ..\data\Crystals_Apr_12\Tileset6\Tile_005-001-...
21 ..\data\Crystals_Apr_12\Tileset6\Tile_005-002-...
22 ..\data\Crystals_Apr_12\Tileset6\Tile_005-003-...
23 ..\data\Crystals_Apr_12\Tileset6\Tile_005-004-...
24 ..\data\Crystals_Apr_12\Tileset6\Tile_005-005-...
25 ..\data\Crystals_Apr_12\Tileset6\Tile_006-001-...
26 ..\data\Crystals_Apr_12\Tileset6\Tile_006-002-...
27 ..\data\Crystals_Apr_12\Tileset6\Tile_006-003-...
28 ..\data\Crystals_Apr_12\Tileset6\Tile_006-004-...
29 ..\data\Crystals_Apr_12\Tileset6\Tile_006-005-...
```

2. Get Image Slice Statistics

This set contains many images. Let's slice those up in 10 by 10

And also apply the statistics on each slice.

```
In [91]: statfuncs = imgutils.statfuncs_5numsummary()
df = imgutils.slicestats(list(df_imgfiles['filename']), 10, 10, statfuncs)
print("records: ", df.shape[0])
df.head()
```

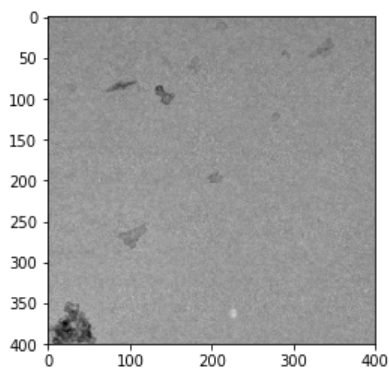
records: 3000

Out[91]:

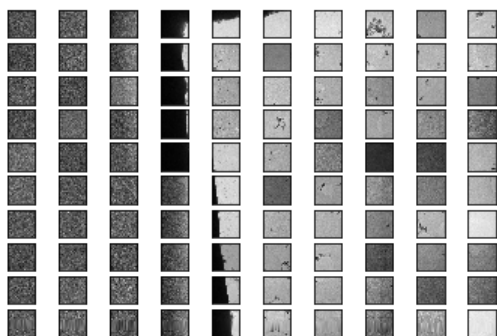
	filename	s_y	s_x	n_y	n_x	alias	img_min	img_quartile1	img_median	img_quartile3	img_max
0	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	0	10	10	img0_0-0	313.0	526.0	573.0	624.0	1110.0
1	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	1	10	10	img0_0-1	335.0	548.0	594.0	643.0	1022.0
2	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	2	10	10	img0_0-2	363.0	628.0	705.0	793.0	1401.0
3	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	3	10	10	img0_0-3	614.0	1100.0	1427.0	2095.0	13357.0
4	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	4	10	10	img0_0-4	1314.0	2569.0	11250.0	11589.0	13435.0

visualize some images

```
In [92]: #get single slice:
sliceimg = imgutils.getimgslice(df, 8)
imgutils.showimg(sliceimg)
```



```
In [93]: # show first image sliced up:
imgname = df_imgfiles.iloc[0]['filename']
imgs, dummy = imgutils.getimgslices_fromdf(df, imgname)
imgutils.showimgs(imgs)
```



```
In [94]: df.isnull().values.any()
```

Out[94]: False

Normalize the statistics using 'standarization'

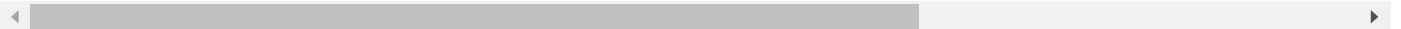
```
In [95]: stat_names = imgutils.stat_names(statfuncs)
print(stat_names)

['img_min', 'img_quartile1', 'img_median', 'img_quartile3', 'img_max']
```

```
In [96]: imgutils.normalize(df, stat_names)
df.head()
```

Out[96]:

	filename	s_y	s_x	n_y	n_x	alias	img_min	img_quartile1	img_median	img_quartile3	img_max	
0	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	0	10	10	img0_0-0	313.0	526.0	573.0	624.0	1110.0	-
1	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	1	10	10	img0_0-1	335.0	548.0	594.0	643.0	1022.0	-
2	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	2	10	10	img0_0-2	363.0	628.0	705.0	793.0	1401.0	-
3	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	3	10	10	img0_0-3	614.0	1100.0	1427.0	2095.0	13357.0	-
4	..\data\Crystals_Apr_12\Tileset6\Tile_001-001-...	0	4	10	10	img0_0-4	1314.0	2569.0	11250.0	11589.0	13435.0	-



```
In [97]: df.isnull().values.any()
```

Out[97]: False

```
In [98]: stat_normnames = imgutils.normalized_names(stat_names)
print(stat_normnames)

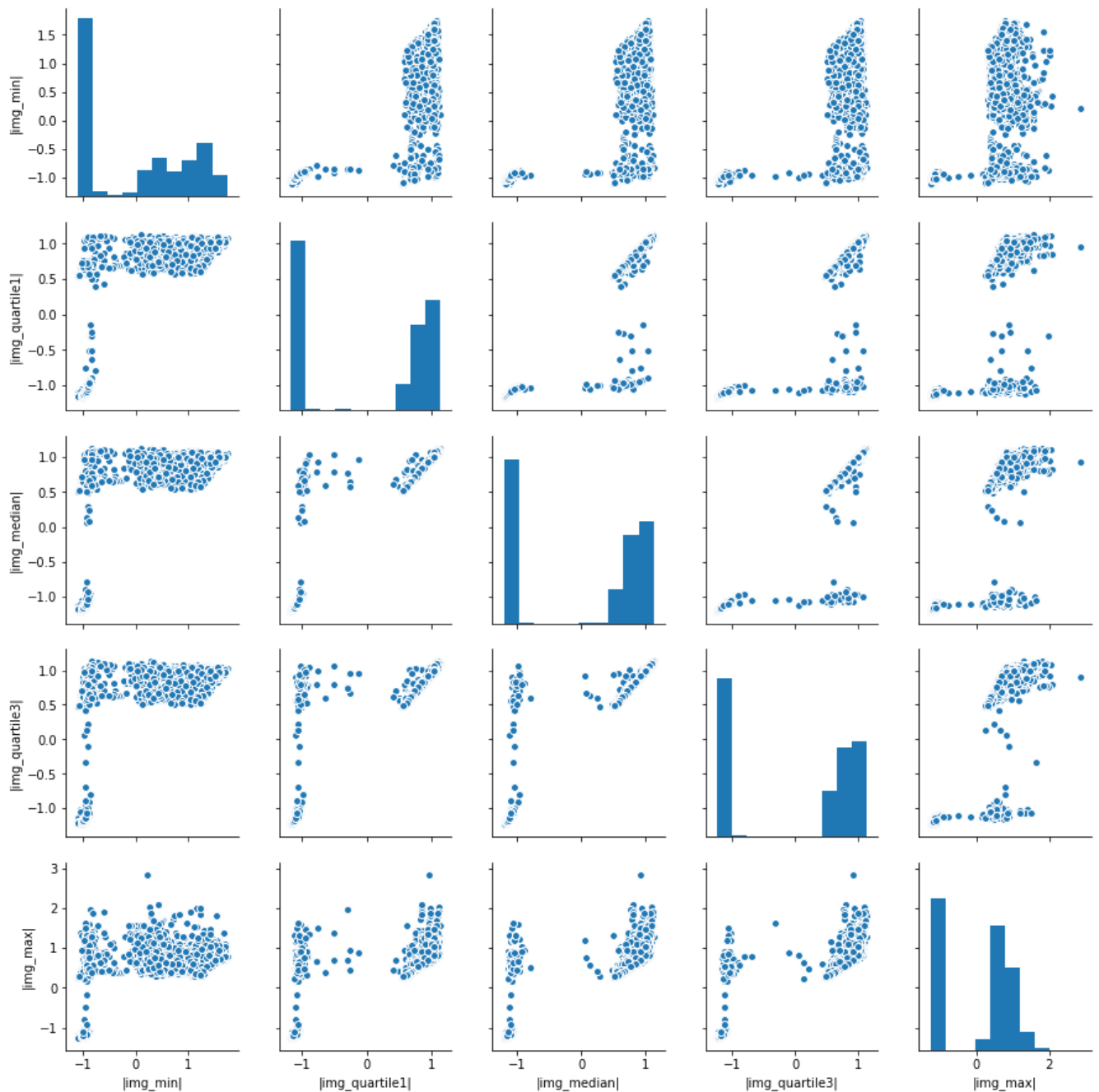
['|img_min|', '|img_quartile1|', '|img_median|', '|img_quartile3|', '|img_max|']
```

3. Check some combinations for patterns

(using the seaborn pairplot)

```
In [99]: import seaborn as sb
```

```
In [100]: %matplotlib inline
sb.pairplot(df, vars=stat_normnames)
#sb.pairplot(df, vars=['|img_mean|', '|img_min|', '|img_std|'])
plt.show()
```

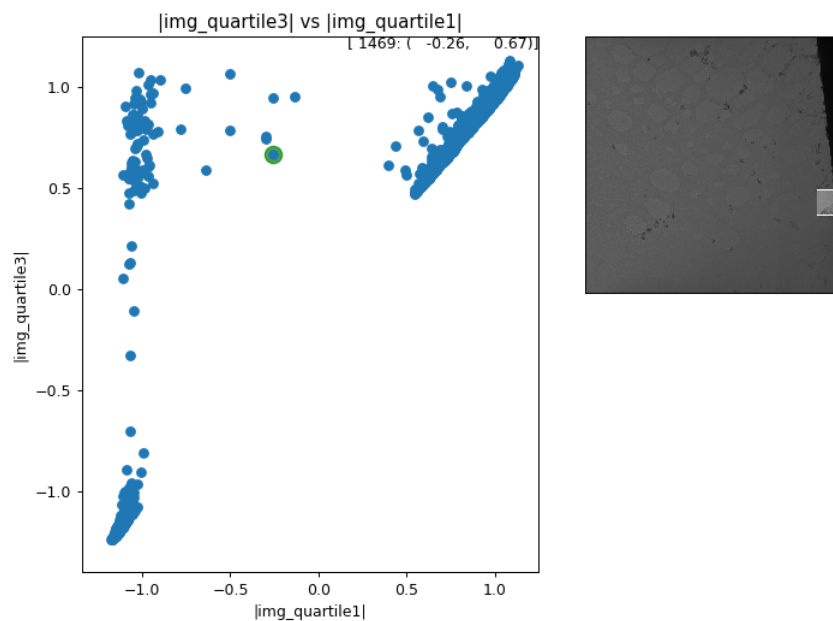


4. Inspect interactively

Let's inspect some combinations that have 'signs of clustering' in the interactive graph

```
In [104]: %matplotlib notebook
```

```
In [105]: imgutils.plotwithimg(df, '|img_quartile1|', '|img_quartile3|', imgutils.highlightingslice, True)
```

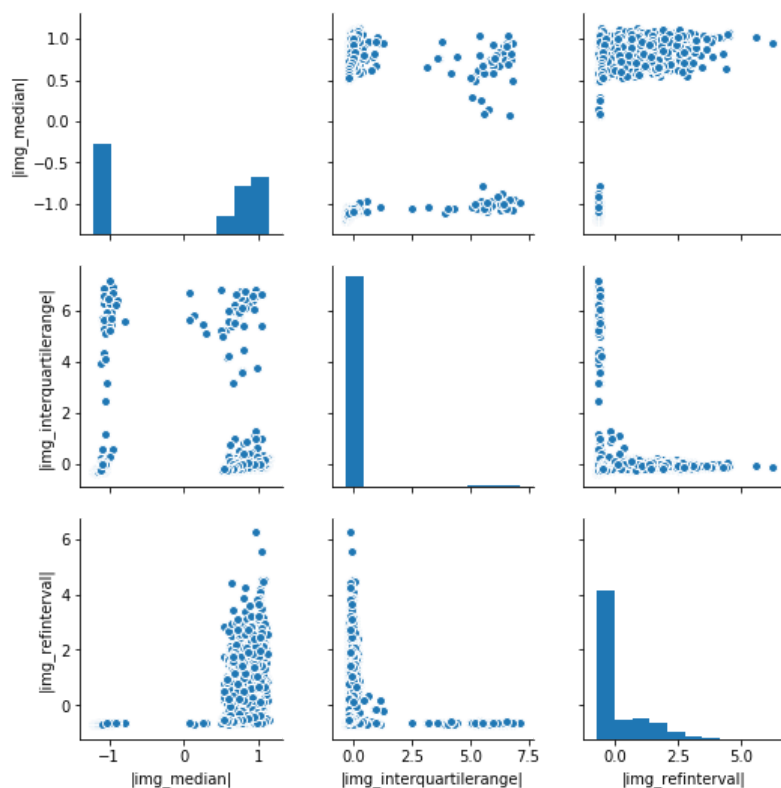


the black parts are easy identified, but the crystals are harder to get out

```
In [106]: #Try other stats:
statfuncs = imgutils.statfuncs_boxandwhisker()
stat_names = imgutils.stat_names(statfuncs)
stat_normnames = imgutils.normalized_names(stat_names)
df = imgutils.slicestats(list(df_imgfiles['filename']), 10, 10, statfuncs)
imgutils.normalize(df, stat_names)

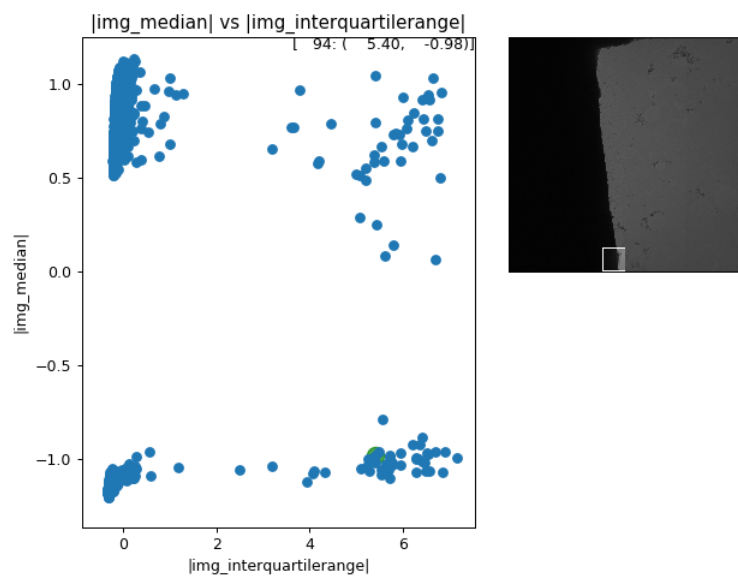
%matplotlib inline
sb.pairplot(df, vars=stat_normnames)
```

```
Out[106]: <seaborn.axisgrid.PairGrid at 0xe93dc18>
```



and check interactively

```
In [107]: %matplotlib notebook
imgutils.plotwithimg(df, '|img_interquartilerange|', '|img_median|', imgutils.highlightingslice, True)
```



looks like 4 nice clusters, but all seem to have black grid, so stats are too coarse

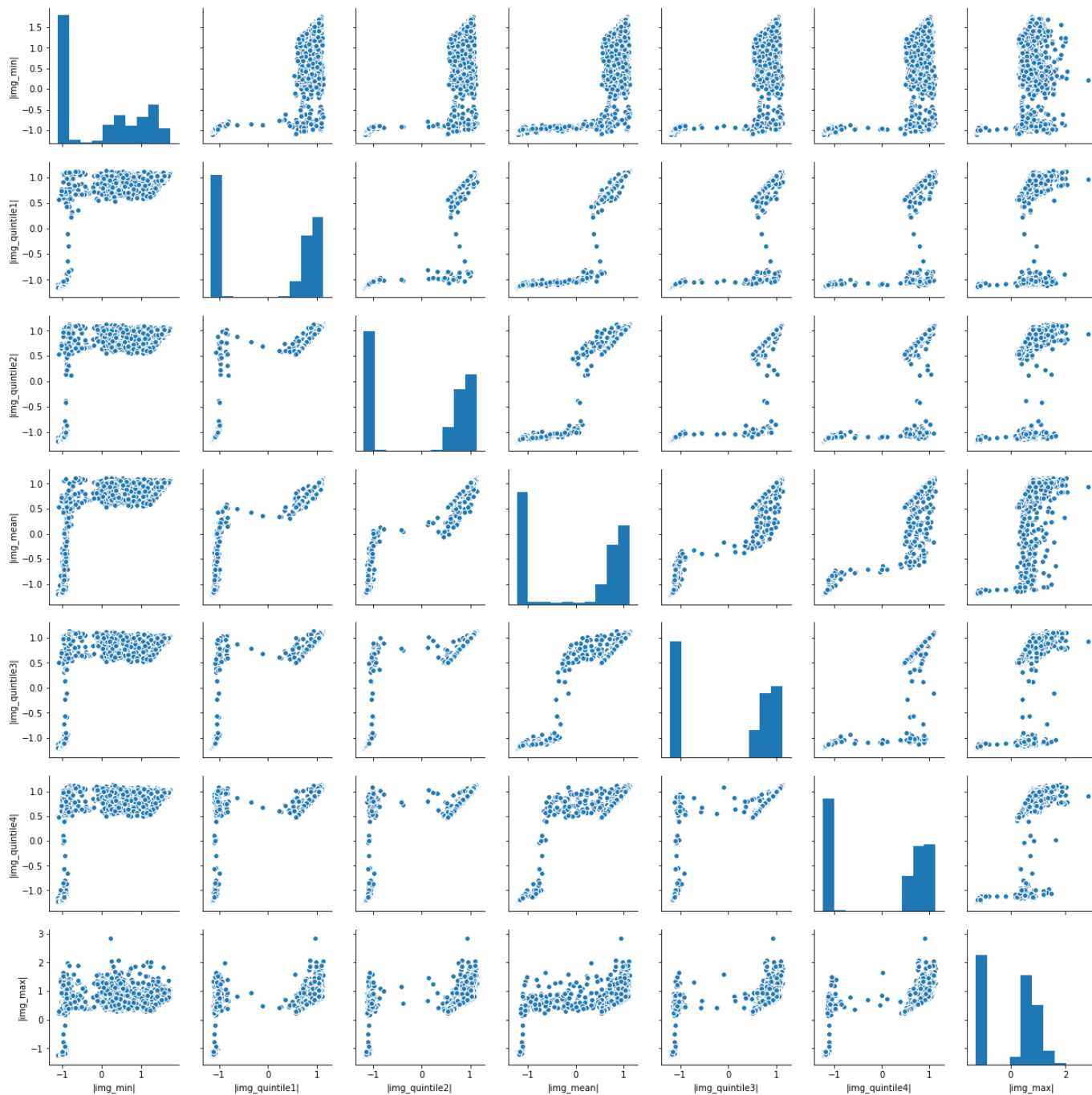
```

In [108]: #Try other stats:
%matplotlib inline
statfuncs = imgutils.statfuncs_7numsummary()
stat_names = imgutils.stat_names(statfuncs)
stat_normnames = imgutils.normalized_names(stat_names)
df = imgutils.slicestats(list(df_imgfiles['filename']), 10, 10, statfuncs)
imgutils.normalize(df, stat_names)

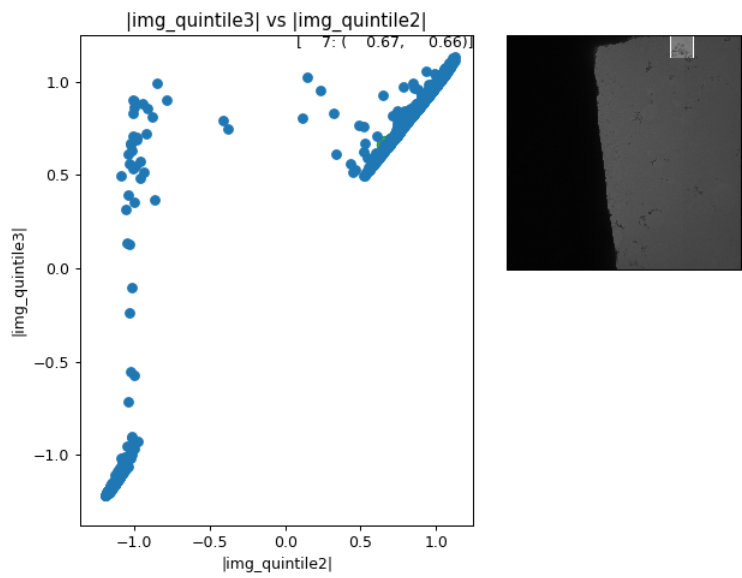
sb.pairplot(df, vars=stat_normnames)

```

Out[108]: <seaborn.axisgrid.PairGrid at 0x195454a8>



```
In [109]: %matplotlib notebook
imgutils.plotwithimg(df, '|img_quintile2|', '|img_quintile3|', imgutils.highlightingslice, True)
```



Hmmm, looks like mean and standard deviation are missing parts

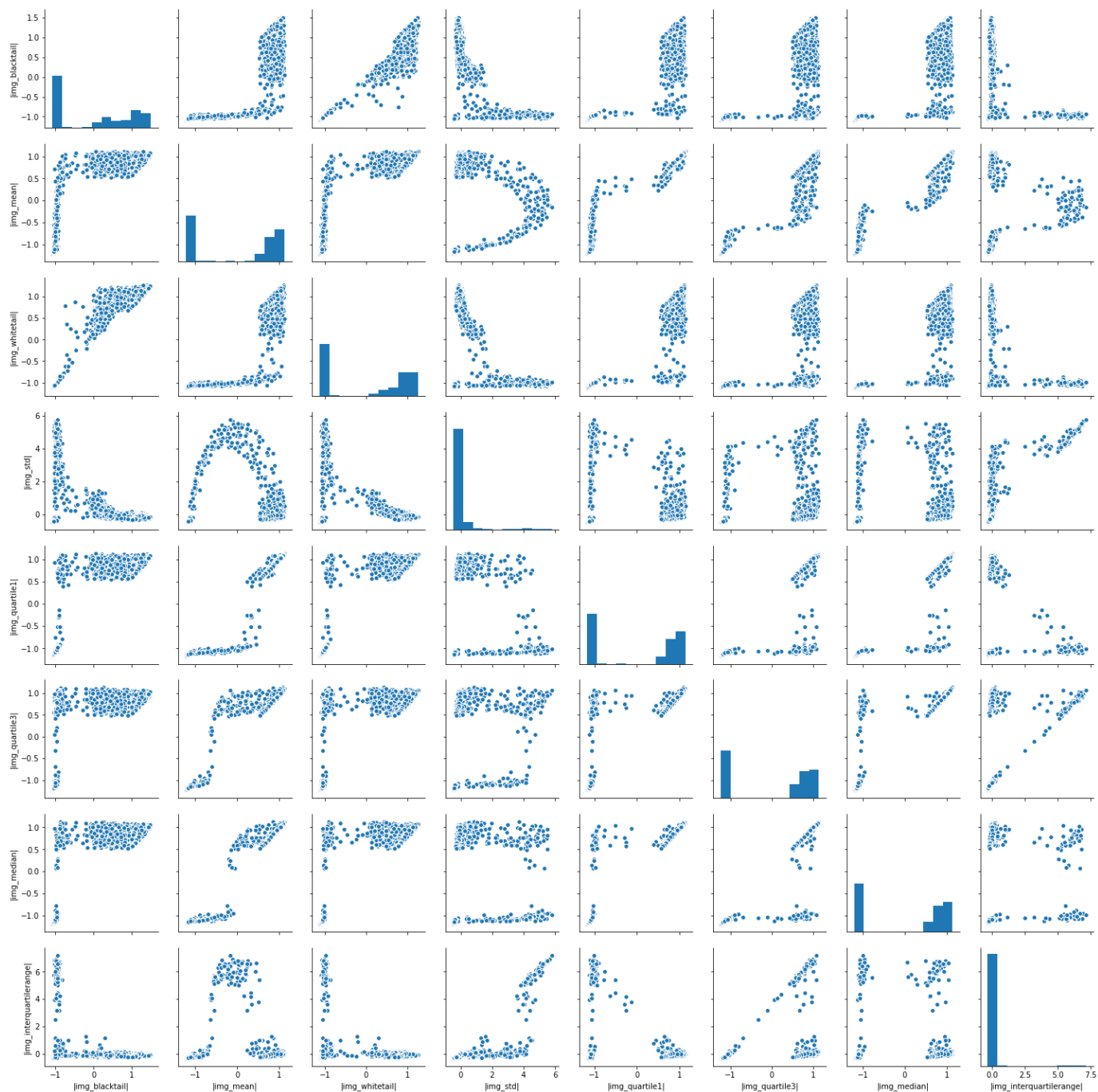

```

In [110]: # added one that is a mix of quartile stats and common stats
%matplotlib inline
statfuncs = imgutils.statfuncs_selection1()
stat_names = imgutils.stat_names(statfuncs)
stat_normnames = imgutils.normalized_names(stat_names)
df = imgutils.slicestats(list(df_imgfiles['filename']), 10, 10, statfuncs)
imgutils.normalize(df, stat_names)

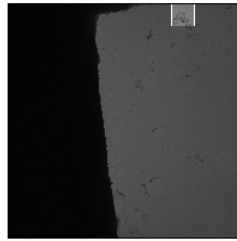
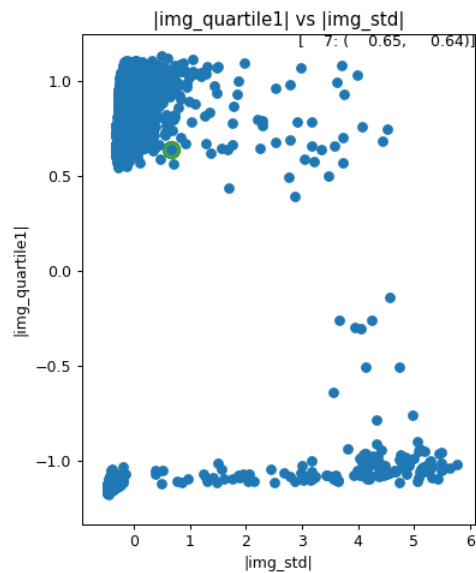
sb.pairplot(df, vars=stat_normnames)

```

Out[110]: <seaborn.axisgrid.PairGrid at 0xe62fac8>

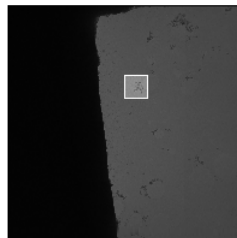
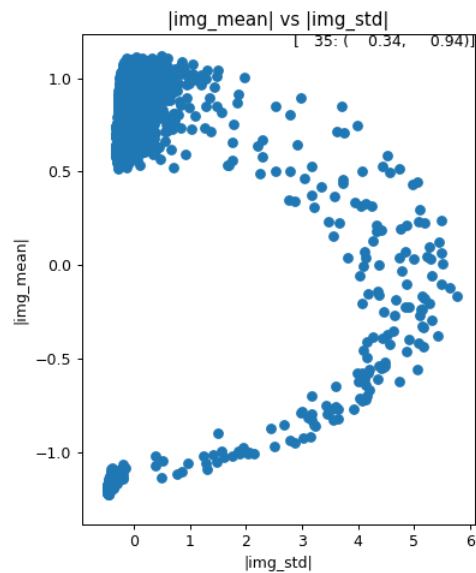


```
In [111]: # try some to get separatable one
%matplotlib notebook
# imgutils.plotwithimg(df, '|img_std|', '|img_interquartilerange|', imgutils.highlightingslice, True)
imgutils.plotwithimg(df, '|img_std|', '|img_quartile1|', imgutils.highlightingslice, True)
```



quartile 1 can separate the black bars from the others. Let's try some more and then plot some heatmaps

```
In [112]: %matplotlib notebook
#imgutils.plotwithimg(df, '|img_interquartilerange|', '|img_quartile3|', imgutils.highlightingslice, True)
imgutils.plotwithimg(df, '|img_std|', '|img_mean|', imgutils.highlightingslice, True)
```



the real particles are 'hidden' in the cluster top-left, i.e. high mean but also some variance. I can try as a 'separator' the summation of both.

5. Heatmaps

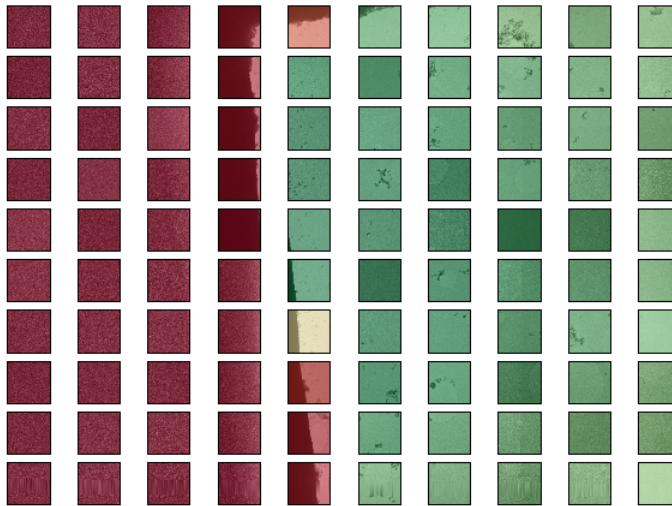
Let's do an attempt to create a score for a heatmap. Looks like `|img_std|` is most infromative

```
In [113]: imgname = df_imgfiles.iloc[0]['filename']
print(imgname)

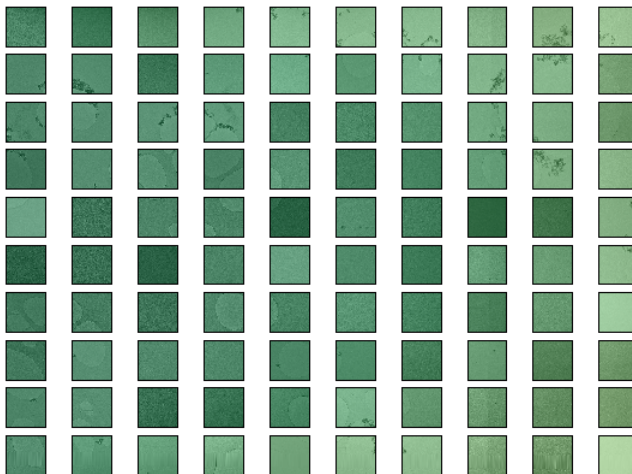
..\data\Crystals_Apr_12\Tileset6\Tile_001-001-000_0-000.tif
```

```
In [114]: # Looks like here that if quartile 2 needs to be > 0 and quartile 1 < -1
df['score'] = df['img_quartile1']
df['|score|'] = imgutils.norm_minmax(df, 'score')
```

```
In [115]: imgname = df_imgfiles.iloc[0]['filename']
imgs, heats = imgutils.getimgslices_fromdf(df, imgname, '|score|')
imgutils.showheatmap(imgs, heats, cmapname='RdYlGn', opacity=0.5, heatdepend_opacity = False)
```



```
In [116]: imgname = df_imgfiles.iloc[2]['filename']
imgs, heats = imgutils.getimgslices_fromdf(df, imgname, '|score|')
imgutils.showheatmap(imgs, heats, cmapname='RdYlGn', opacity=0.5, heatdepend_opacity = False)
```



```
In [117]: imgname = df_imgfiles.iloc[1]['filename']
          imgs, heats = imgutils.getimgslices_fromdf(df, imgname, '|score|')
          imgutils.showheatmap(imgs, heats, cmapname='RdYlGn', opacity=0.5, heatdepend_opacity = False)
```

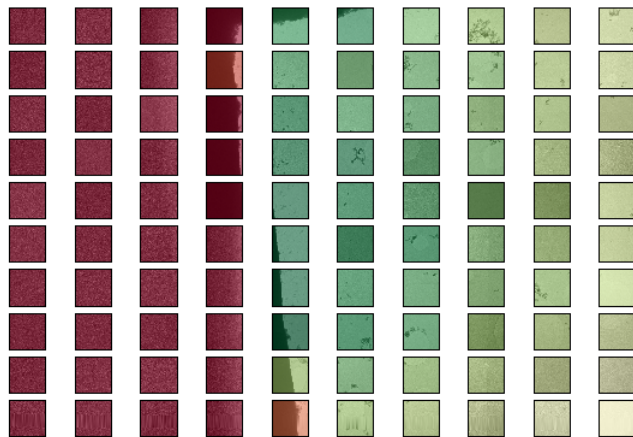


Hmm

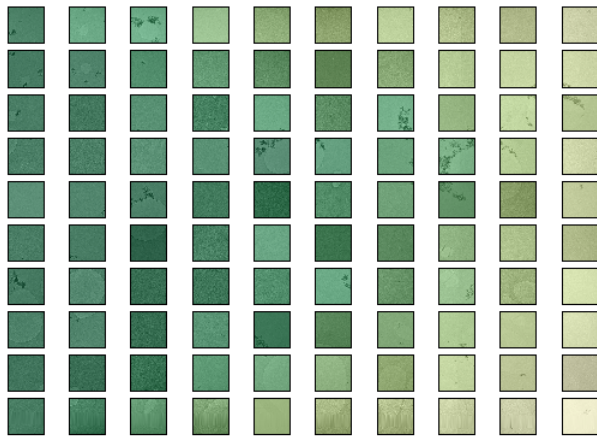
try the other score:

```
In [118]: # Looks like here that if quartile 2 needs to be > 0 and quartile 1 < -1
          df['score2'] = df['img_mean'] + df['img_std']
          df['|score2|'] = imgutils.norm_standardize(df, 'score2')
```

```
In [119]: imgname = df_imgfiles.iloc[0]['filename']
          imgs, heats = imgutils.getimgslices_fromdf(df, imgname, '|score2|')
          imgutils.showheatmap(imgs, heats, cmapname='RdYlGn', opacity=0.5, heatdepend_opacity = False)
```



```
In [120]: imgname = df_imgfiles.iloc[1]['filename']
          imgs, heats = imgutils.getimgslices_fromdf(df, imgname, '|score2|')
          imgutils.showheatmap(imgs, heats, cmapname='RdYlGn', opacity=0.5, heatdepend_opacity = False)
```

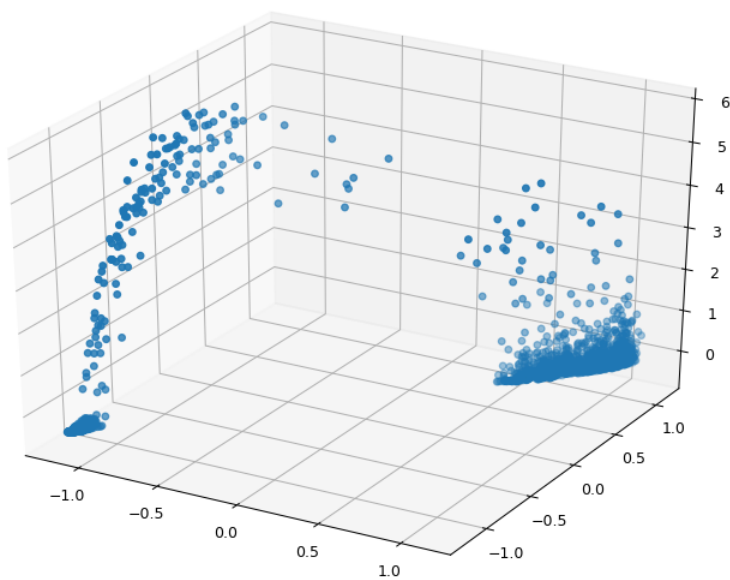


I really think this needs 3 dimensions!

```
In [123]: from mpl_toolkits.mplot3d import Axes3D
          %matplotlib notebook
```

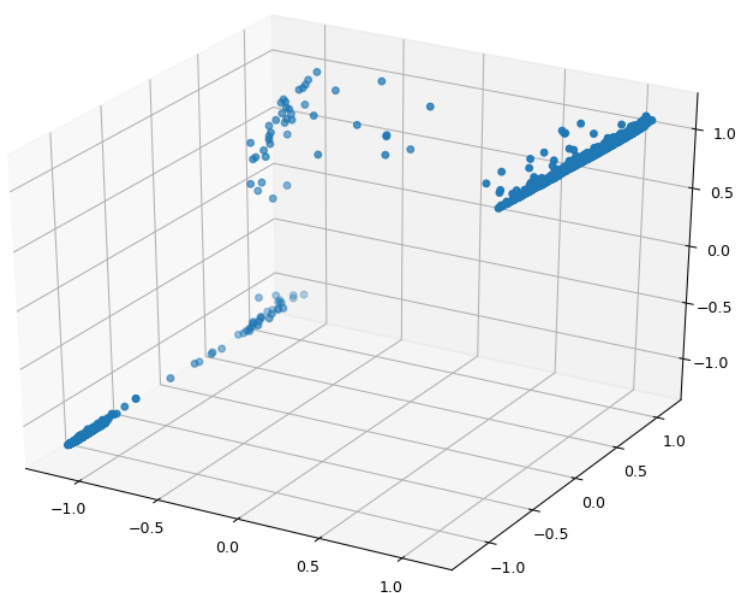
```
In [124]: fig = plt.figure()
          ax = Axes3D(fig)

          ax.scatter(df['|img_quartile1|'], df['|img_mean|'], df['|img_std|'])
          plt.show()
```



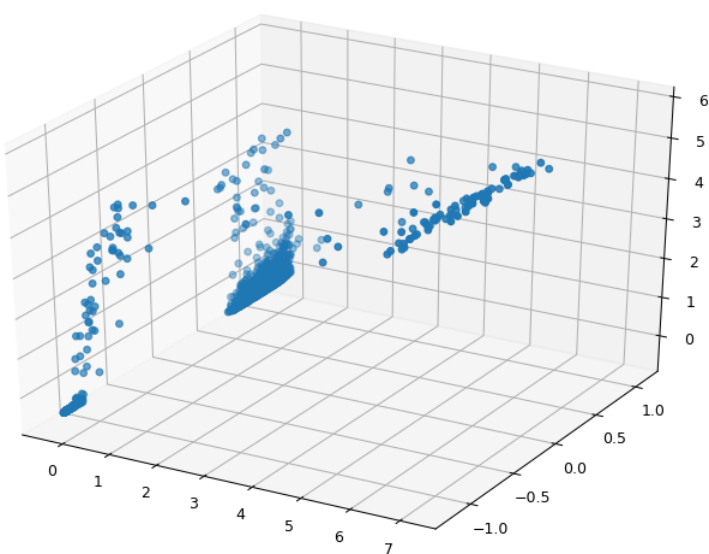
```
In [125]: fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(df['|img_quartile1|'], df['|img_quartile3|'], df['|img_median|'])
plt.show()
```



```
In [126]: fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(df['|img_interquartilerange|'], df['|img_mean|'], df['|img_std|'])
plt.show()
```



6. Conclusions & Remarks

- need multi-dimension analyses and e.g. PCA; let's export these values for that!
- consider pre-filtering the image to take out the noise (99% range)
- consider re-scaling images two 2k x 2k or 1k x 1k for performance

7. Next steps

- Export this data set for multi dimension visualization