prior_montecarlo

April 29, 2019

1 Run and process the prior monte carlo and pick a "truth" realization

```
In [1]: import os
        import shutil
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import flopy
        import pyemu
```

```
flopy is installed in /Users/jeremyw/Dev/gw1876/activities_2day_mfm/notebooks/flopy
```

```
In [2]: t_d = "template"
        pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
```

1.0.1 Decide what pars are uncertain in the truth

We need to decide what our truth looks like - should the pilot points or the grid-scale pars be the source of spatial variability?

```
In [3]: par = pst.parameter_data
        # grid pars
        #should_fix = par.loc[par.pargp.apply(lambda x: "gr" in x),"parnme"]
        # pp pars
        should_fix = par.loc[par.pargp.apply(lambda x: "pp" in x),"parnme"]
        #pst.npar - should_fix.shape[0]
```

```
In [4]: pe = pyemu.ParameterEnsemble.from_binary(pst=pst,filename=os.path.join(t_d,"prior.jcb")
        pe.loc[:,should_fix] = 1.0
        pe.to_csv(os.path.join(t_d,"sweep_in.csv"))
```

```
new binary format detected...
```

1.0.2 run the prior ensemble in parallel locally

```
In [5]: m_d = "master_prior_sweep"
        pyemu.os_utils.start_slaves(t_d,"pestpp-swp","freyberg.pst",num_slaves=20,slave_root="
```

1.0.3 Load the output ensemble and plot a few things

```
In [6]: obs_df = pd.read_csv(os.path.join(m_d,"sweep_out.csv"),index_col=0)
        obs_df.shape
```

```
Out[6]: (200, 4462)
```

drop any failed runs

```
In [7]: obs_df = obs_df.loc[obs_df.failed_flag==0,:]
        obs_df.shape
```

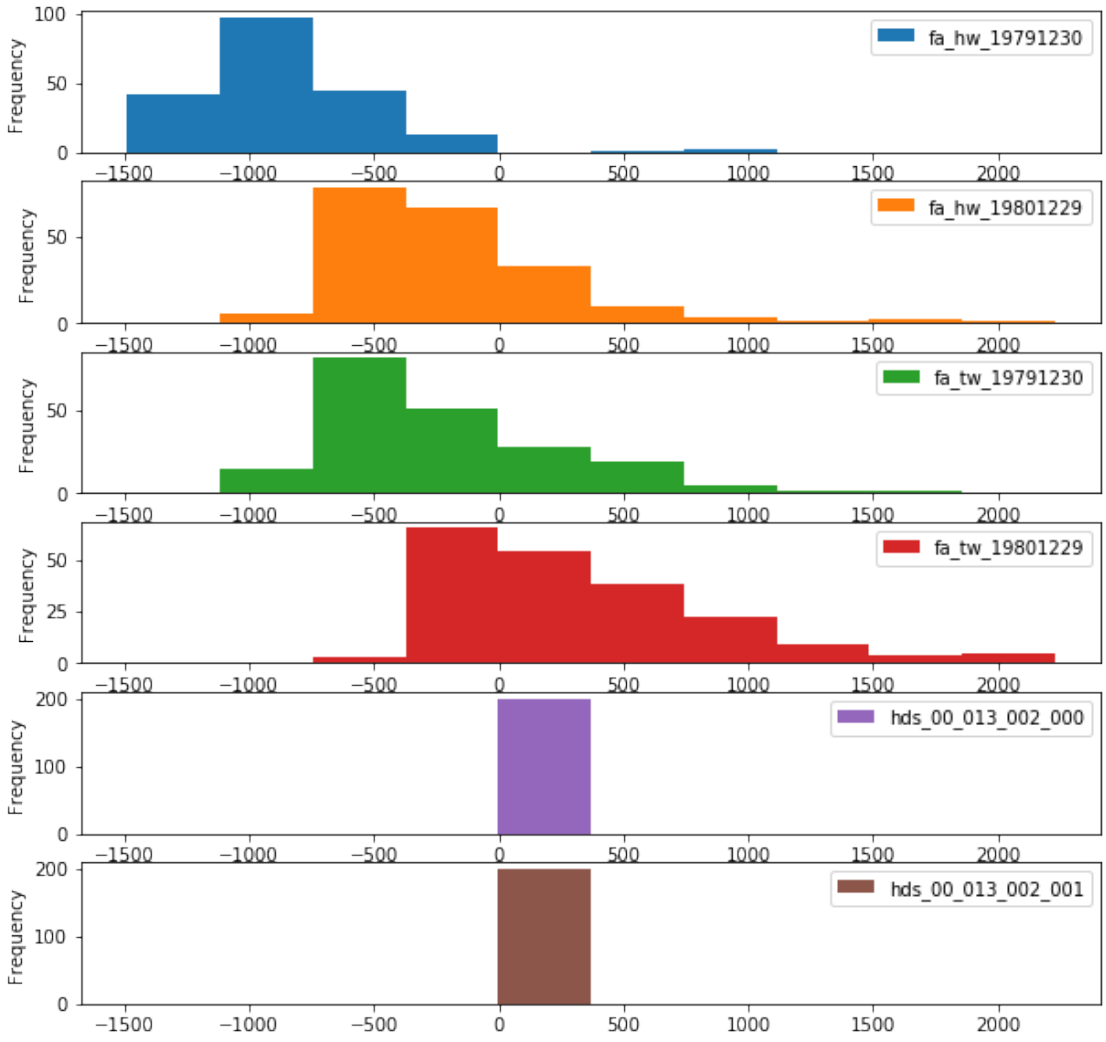
```
Out[7]: (200, 4462)
```

```
In [8]: fnames = pst.pestpp_options["forecasts"].split(',')
        fnames
```

```
Out[8]: ['fa_hw_19791230',
         'fa_hw_19801229',
         'fa_tw_19791230',
         'fa_tw_19801229',
         'hds_00_013_002_000',
         'hds_00_013_002_001']
```

```
In [9]: obs_df.loc[:,fnames].plot(kind="hist",subplots=True,figsize=(10,10),sharex=False)
```

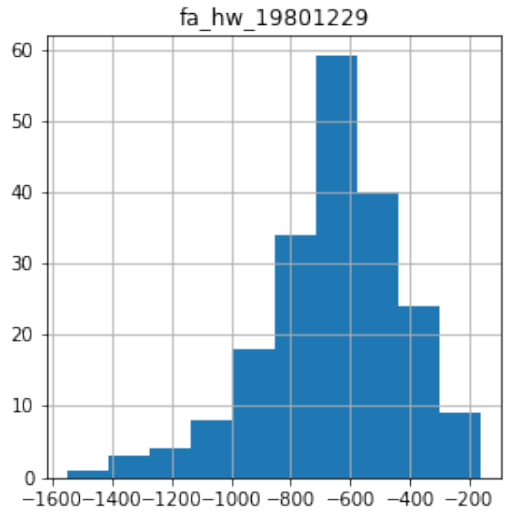
```
Out[9]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x1091e2da0>,
               <matplotlib.axes._subplots.AxesSubplot object at 0x181e73a7f0>,
               <matplotlib.axes._subplots.AxesSubplot object at 0x109209d68>,
               <matplotlib.axes._subplots.AxesSubplot object at 0x10922b320>,
               <matplotlib.axes._subplots.AxesSubplot object at 0x109f16898>,
               <matplotlib.axes._subplots.AxesSubplot object at 0x109ee7e10>],
              dtype=object)
```

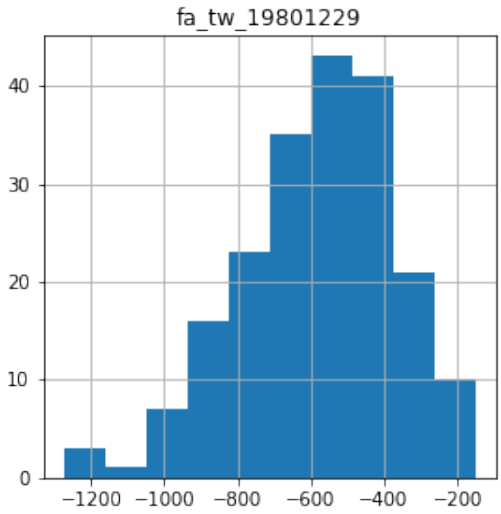


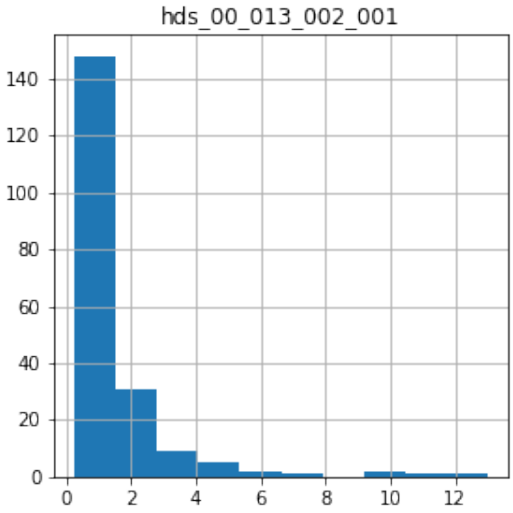
We see that under scenario conditions, many more realizations for the flow to the aquifer in the headwaters are postive (as expected). Lets difference these two:

```
In [10]: sfnames = [f for f in fnames if "1980" in f or "_001" in f]
         hfnames = [f for f in fnames if "1979" in f or "_000" in f]
         diff = obs_df.loc[:,hfnames].values - obs_df.loc[:,sfnames].values
         diff = pd.DataFrame(diff,columns=sfnames)
         diff.hist(figsize=(10,10))
```

```
Out[10]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x181e4540b8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x109270c50>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x109ff41d0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x10a00a748>]],
               dtype=object)
```







We now see that the most extreme scenario yields a large decrease in flow from the aquifer to the headwaters (the most negative value)

1.0.4 setting the "truth"

We just need to replace the observed values (`obsval`) in the control file with the outputs for one of the realizations on `obs_df`. In this way, we now have the nonzero values for history matching, but also the `truth` values for comparing how we are doing with other unobserved quantities. Im going to pick a realization that yields an "average" variability of the observed gw levels:

```
In [11]: # choose the realization with a low historic gw to sw headwater flux
         hist_swgw = obs_df.loc[:,"fa_hw_19791230"].sort_values()
         idx = hist_swgw.index[10]
         idx
```

```
Out[11]: 46
```

```
In [12]: obs_df.loc[idx,pst.nnz_obs_names]
```

```
Out[12]: fo_39_19791230        11925.000000
         hds_00_002_009_000       35.557449
         hds_00_002_015_000       34.771416
         hds_00_003_008_000       35.705761
         hds_00_009_001_000       37.410831
         hds_00_013_010_000       35.260769
         hds_00_015_016_000       34.869968
         hds_00_021_010_000       35.211872
         hds_00_022_015_000       34.578663
         hds_00_024_004_000       36.148964
         hds_00_026_006_000       35.571796
         hds_00_029_015_000       34.557293
         hds_00_033_007_000       34.680103
         hds_00_034_010_000       34.403061
         Name: 46, dtype: float64
```

Lets see how our selected truth does with the swgw forecasts:

```
In [13]: obs_df.loc[idx,fnames]
```

```
Out[13]: fa_hw_19791230       -1289.170400
         fa_hw_19801229        -690.416200
         fa_tw_19791230        -573.743640
         fa_tw_19801229        -160.726405
         hds_00_013_002_000      37.478226
         hds_00_013_002_001      36.462036
         Name: 46, dtype: float64
```

```
In [14]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         obs = pst.observation_data
         obs.loc[:,"obsval"] = obs_df.loc[idx,pst.obs_names]
         obs.loc[obs.obgnme=="calhead","weight"] = 5.0
         obs.loc[obs.obgnme=="calflux","weight"] = 0.035
         obs.weight.value_counts()
```

```
Out[14]: 0.000    4420
         5.000      13
         0.035       1
         Name: weight, dtype: int64
```

Now, it is custom to add noise to the observed values…we will use the classic Gaussian noise…zero mean and standard deviation of 1 over the weight

```
In [15]: # this should give the same standard normal draws each time
         np.random.seed(seed=0)
         snd = np.random.randn(pst.nnz_obs)
         snd
```

```
Out[15]: array([ 1.76405235,  0.40015721,  0.97873798,  2.2408932 ,  1.86755799,
                -0.97727788,  0.95008842, -0.15135721, -0.10321885,  0.4105985 ,
                 0.14404357,  1.45427351,  0.76103773,  0.12167502])

In [16]: noise = snd * 1./obs.loc[pst.nnz_obs_names,"weight"]
         noise

Out[16]: obsnme
         fo_39_19791230        50.401496
         hds_00_002_009_000     0.080031
         hds_00_002_015_000     0.195748
         hds_00_003_008_000     0.448179
         hds_00_009_001_000     0.373512
         hds_00_013_010_000    -0.195456
         hds_00_015_016_000     0.190018
         hds_00_021_010_000    -0.030271
         hds_00_022_015_000    -0.020644
         hds_00_024_004_000     0.082120
         hds_00_026_006_000     0.028809
         hds_00_029_015_000     0.290855
         hds_00_033_007_000     0.152208
         hds_00_034_010_000     0.024335
         Name: weight, dtype: float64
```

Only run this block once!!!

```
In [17]: pst.observation_data.loc[noise.index,"obsval"] += noise
         pst.write(os.path.join(t_d,"freyberg.pst"))
         pyemu.os_utils.run("pestpp-ies freyberg.pst",cwd=t_d)

In [18]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         print(pst.phi)
         pst.res.loc[pst.nnz_obs_names,:]

559.6226991936528
```

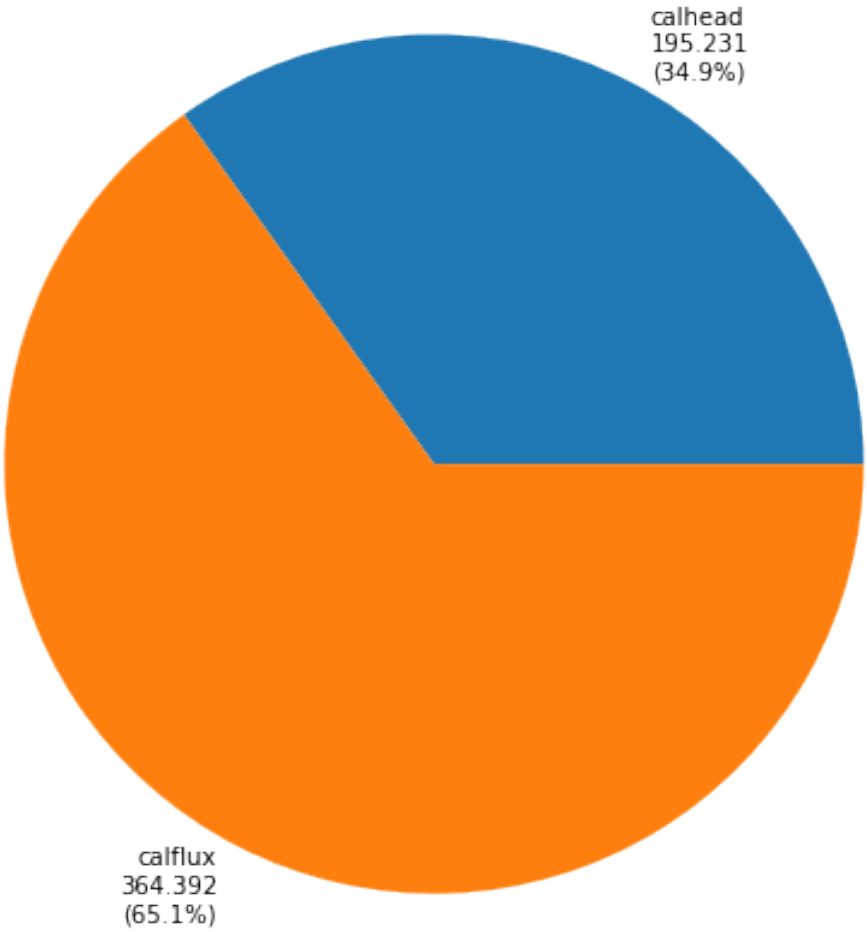
```
Out[18]:                                    name    group      measured      modelled  \
         name
         fo_39_19791230          fo_39_19791230  calflux  11975.401496  11430.000000
         hds_00_002_009_000  hds_00_002_009_000  calhead     35.637481     37.107498
         hds_00_002_015_000  hds_00_002_015_000  calhead     34.967163     35.045185
         hds_00_003_008_000  hds_00_003_008_000  calhead     36.153940     37.397289
         hds_00_009_001_000  hds_00_009_001_000  calhead     37.784343     39.546417
         hds_00_013_010_000  hds_00_013_010_000  calhead     35.065313     35.571774
         hds_00_015_016_000  hds_00_015_016_000  calhead     35.059986     34.835716
         hds_00_021_010_000  hds_00_021_010_000  calhead     35.181601     35.386250
         hds_00_022_015_000  hds_00_022_015_000  calhead     34.558019     34.577492
         hds_00_024_004_000  hds_00_024_004_000  calhead     36.231084     36.760464
```

```
hds_00_026_006_000  hds_00_026_006_000  calhead     35.600605     35.896149
hds_00_029_015_000  hds_00_029_015_000  calhead     34.848148     34.453842
hds_00_033_007_000  hds_00_033_007_000  calhead     34.832311     34.678810
hds_00_034_010_000  hds_00_034_010_000  calhead     34.427396     34.118073
```

name	residual	weight
fo_39_19791230	545.401496	0.035
hds_00_002_009_000	-1.470017	5.000
hds_00_002_015_000	-0.078022	5.000
hds_00_003_008_000	-1.243350	5.000
hds_00_009_001_000	-1.762074	5.000
hds_00_013_010_000	-0.506460	5.000
hds_00_015_016_000	0.224270	5.000
hds_00_021_010_000	-0.204649	5.000
hds_00_022_015_000	-0.019473	5.000
hds_00_024_004_000	-0.529380	5.000
hds_00_026_006_000	-0.295544	5.000
hds_00_029_015_000	0.394305	5.000
hds_00_033_007_000	0.153501	5.000
hds_00_034_010_000	0.309323	5.000

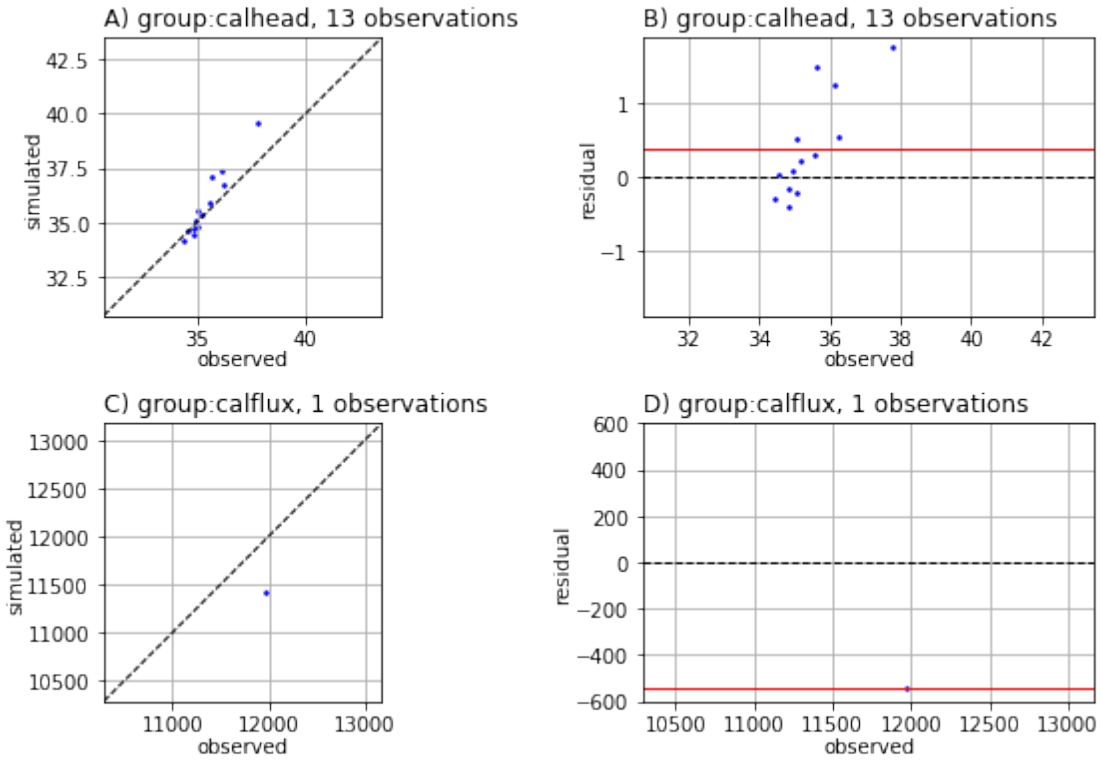
In [19]:
```
pst.phi_components
pst.plot(kind='phi_pie')
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x109f59be0>
```




```
In [20]: figs = pst.plot(kind="1to1")
```

```
<Figure size 576x756 with 0 Axes>
```



Publication ready figs - oh snap!

Just to make sure we have everything working right, we should be able to load the truth parameters, run the model once and have a `phi` equivalent to the noise vector:

```
In [21]: par_df = pd.read_csv(os.path.join(m_d,"sweep_in.csv"),index_col=0)
         pst.parameter_data.loc[:,"parval1"] = par_df.loc[idx,pst.par_names]
         pst.write(os.path.join(m_d,"test.pst"))
         pyemu.os_utils.run("pestpp-ies.exe test.pst",cwd=m_d)
         pst = pyemu.Pst(os.path.join(m_d,"test.pst"))
         print(pst.phi)
         pst.res.loc[pst.nnz_obs_names,:]
```

```
17.528847278524836
```

Out[21]:

	name	group	measured	modelled	residual	weight
name						
fo_39_19791230	fo_39_19791230	calflux	11975.401496	11925.000000	50.401496	0.035
hds_00_002_009_000	hds_00_002_009_000	calhead	35.637481	35.557449	0.080031	5.000
hds_00_002_015_000	hds_00_002_015_000	calhead	34.967163	34.771416	0.195748	5.000
hds_00_003_008_000	hds_00_003_008_000	calhead	36.153940	35.705761	0.448179	5.000
hds_00_009_001_000	hds_00_009_001_000	calhead	37.784343	37.410831	0.373512	5.000
hds_00_013_010_000	hds_00_013_010_000	calhead	35.065313	35.260769	-0.195456	5.000
hds_00_015_016_000	hds_00_015_016_000	calhead	35.059986	34.869968	0.190018	5.000
hds_00_021_010_000	hds_00_021_010_000	calhead	35.181601	35.211872	-0.030271	5.000
hds_00_022_015_000	hds_00_022_015_000	calhead	34.558019	34.578663	-0.020644	5.000
hds_00_024_004_000	hds_00_024_004_000	calhead	36.231084	36.148964	0.082120	5.000
hds_00_026_006_000	hds_00_026_006_000	calhead	35.600605	35.571796	0.028809	5.000
hds_00_029_015_000	hds_00_029_015_000	calhead	34.848148	34.557293	0.290855	5.000
hds_00_033_007_000	hds_00_033_007_000	calhead	34.832311	34.680103	0.152208	5.000
hds_00_034_010_000	hds_00_034_010_000	calhead	34.427396	34.403061	0.024335	5.000

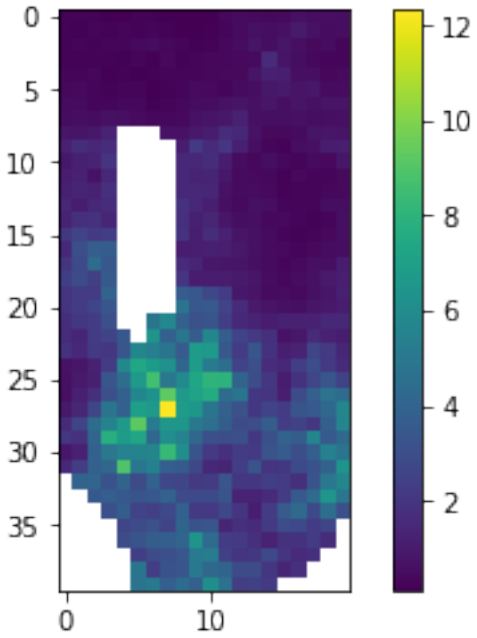
The residual should be exactly the noise values from above.Lets load the model (that was just run using the true pars) and check some things

```
In [22]: m = flopy.modflow.Modflow.load("freyberg.nam",model_ws=m_d)
```

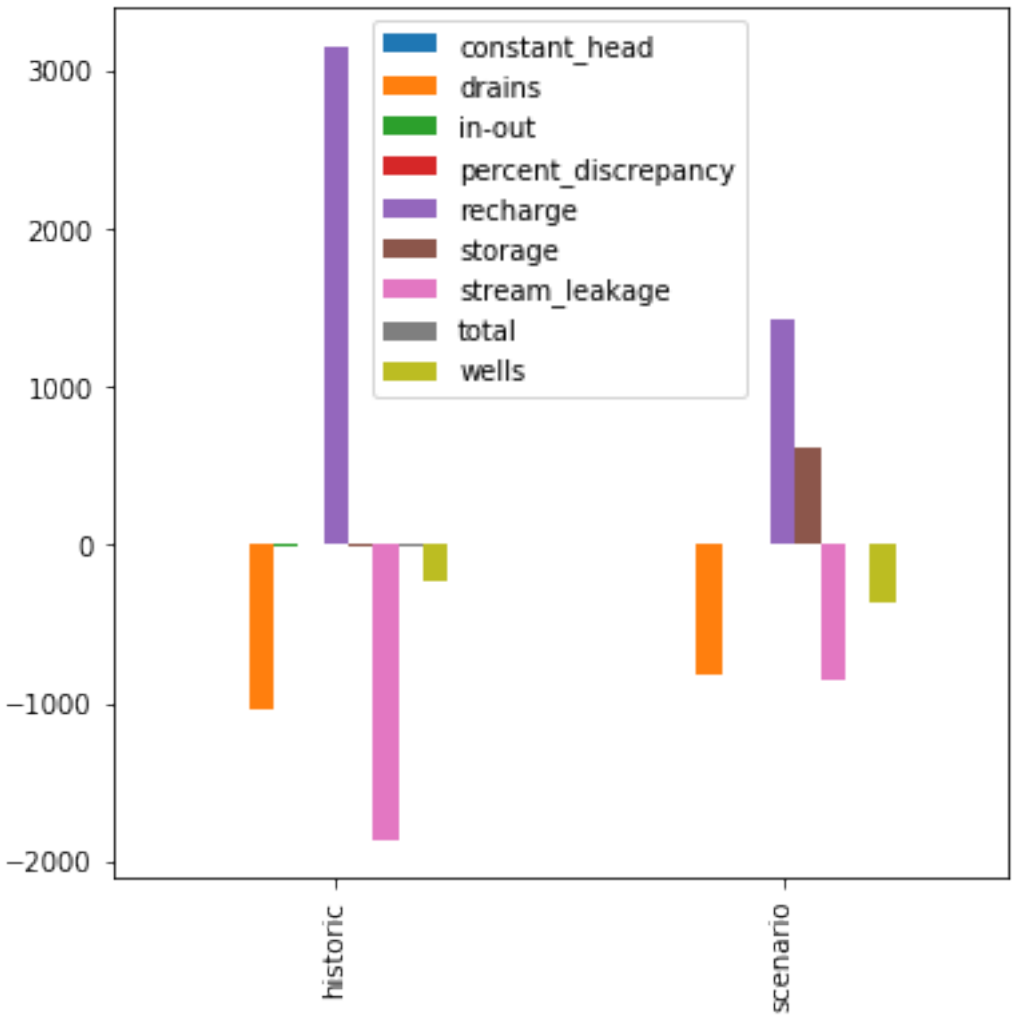
In [23]:
```
a = m.upw.hk[0].array
#a = m.rch.rech[0].array
a = np.ma.masked_where(m.bas6.ibound[0].array==0,a)
print(a.min(),a.max())
c = plt.imshow(a)
plt.colorbar(c)
```

```
0.1282324 12.32662
```

Out[23]: <matplotlib.colorbar.Colorbar at 0x181ffa67f0>



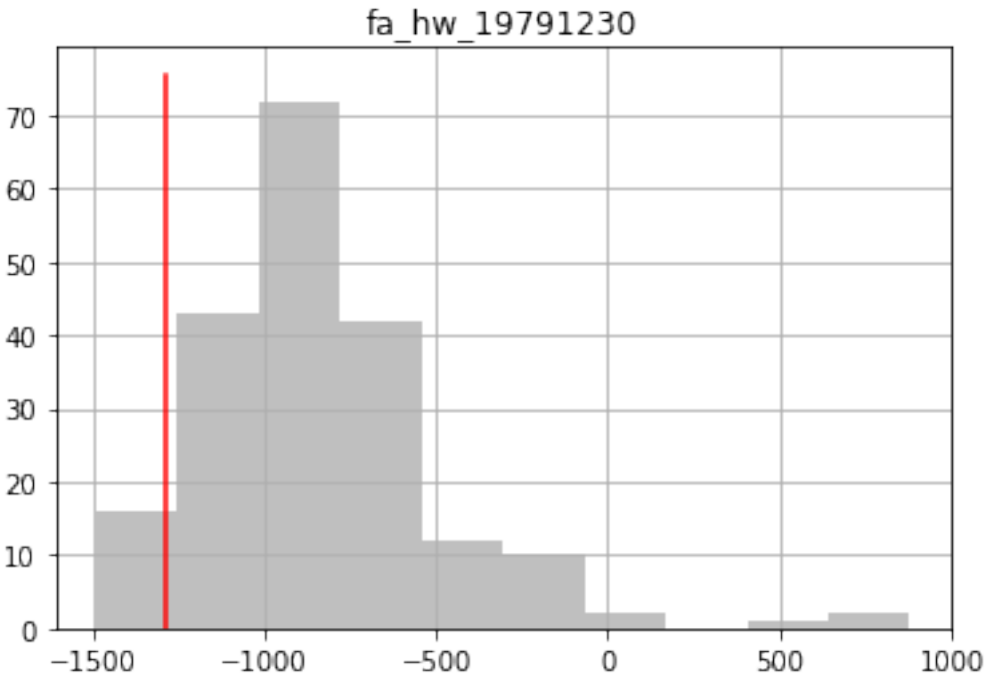
In [24]:
```
lst = flopy.utils.MfListBudget(os.path.join(m_d,"freyberg.list"))
df = lst.get_dataframes(diff=True)[0]
ax = df.plot(kind="bar",figsize=(6,6))
a = ax.set_xticklabels(["historic","scenario"],rotation=90)
```

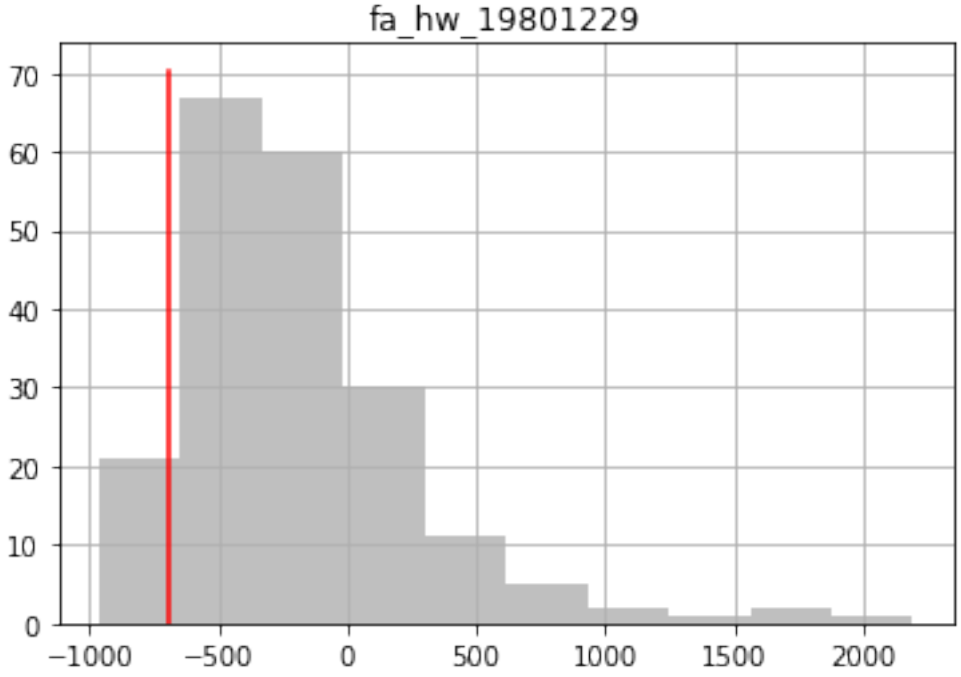


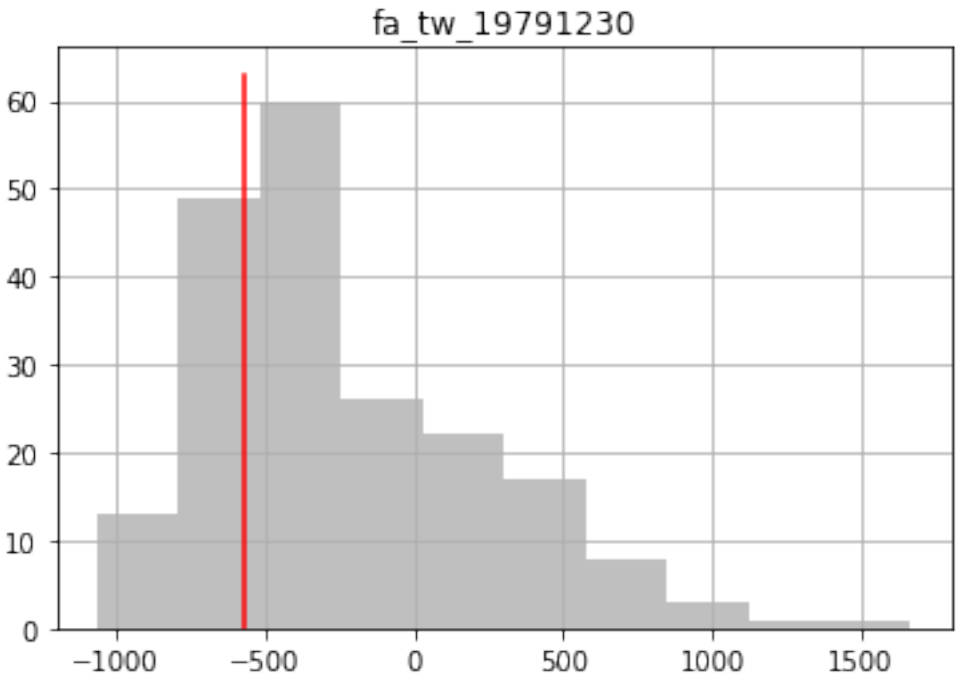
1.0.5 see how our existing observation ensemble compares to the truth

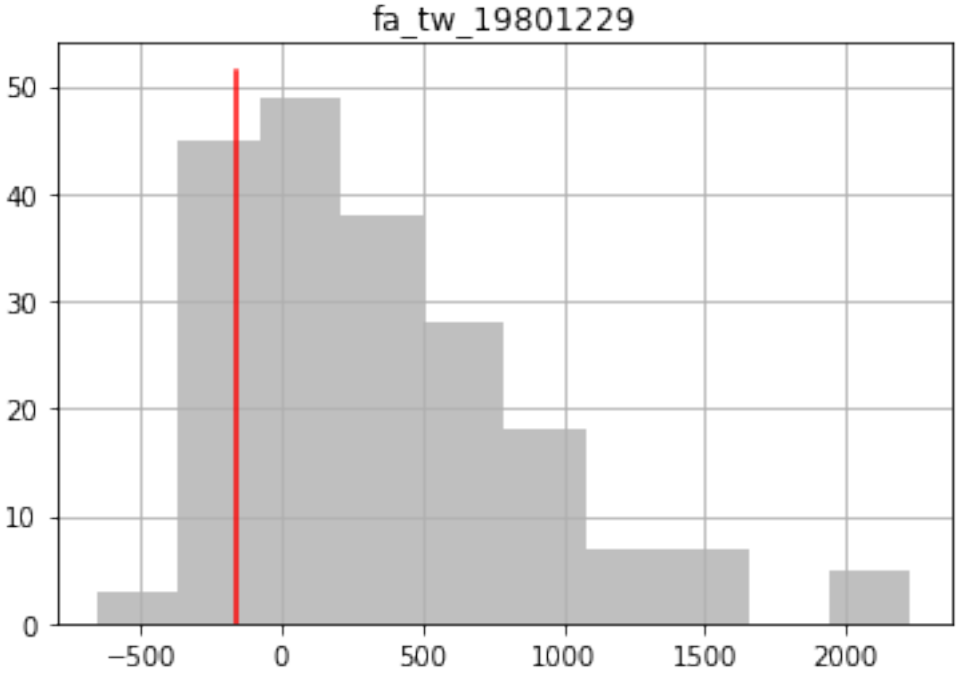
sw-gw outputs:

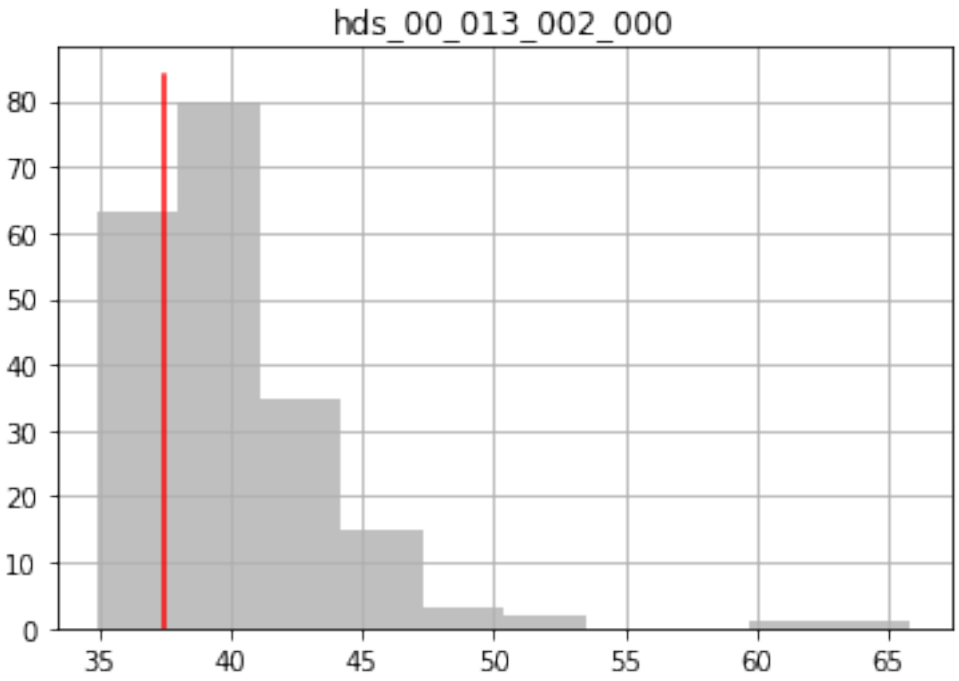
```
In [25]: obs = pst.observation_data

         for forecast in fnames:
             ax = plt.subplot(111)
             obs_df.loc[:,forecast].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
             ax.set_title(forecast)
             plt.show()
```

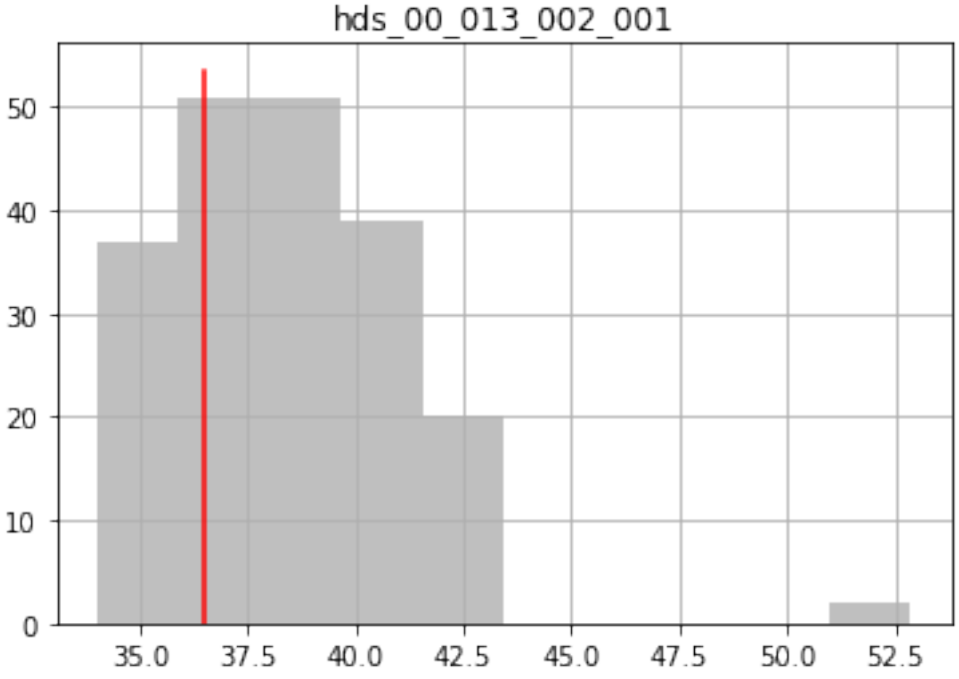

fa_hw_19791230
70
60
50
40
30
20
10
0
−1500
−1000
−500
0
500
1000


fa_hw_19801229
70
60
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000


fa_tw_19791230
60
50
40
30
20
10
0
−1000
−500
0
500
1000
1500

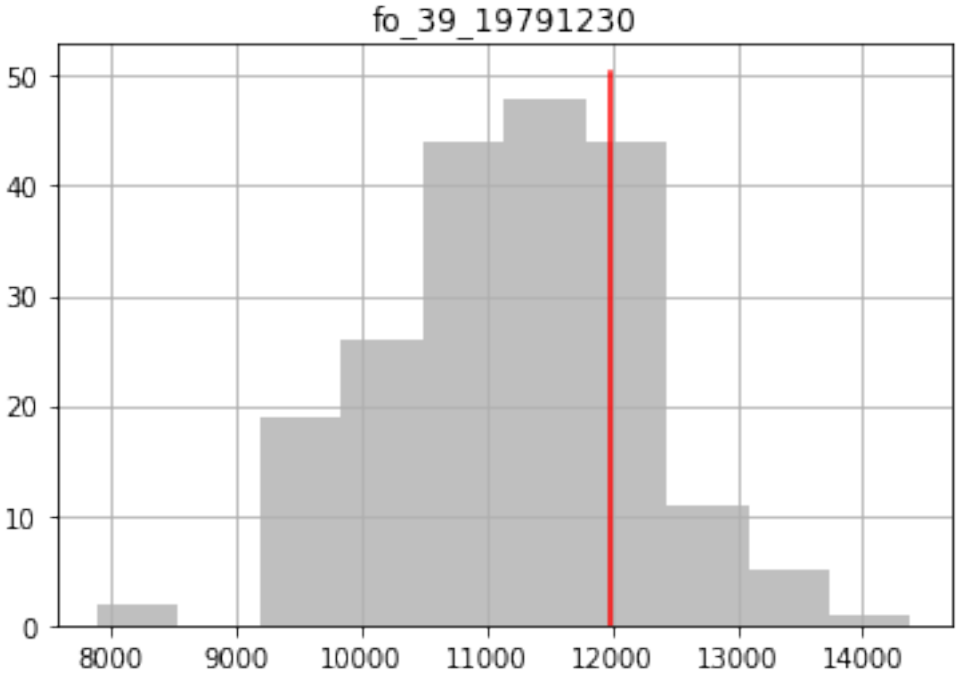

fa_tw_19801229
50
40
30
20
10
0
−500
0
500
1000
1500
2000

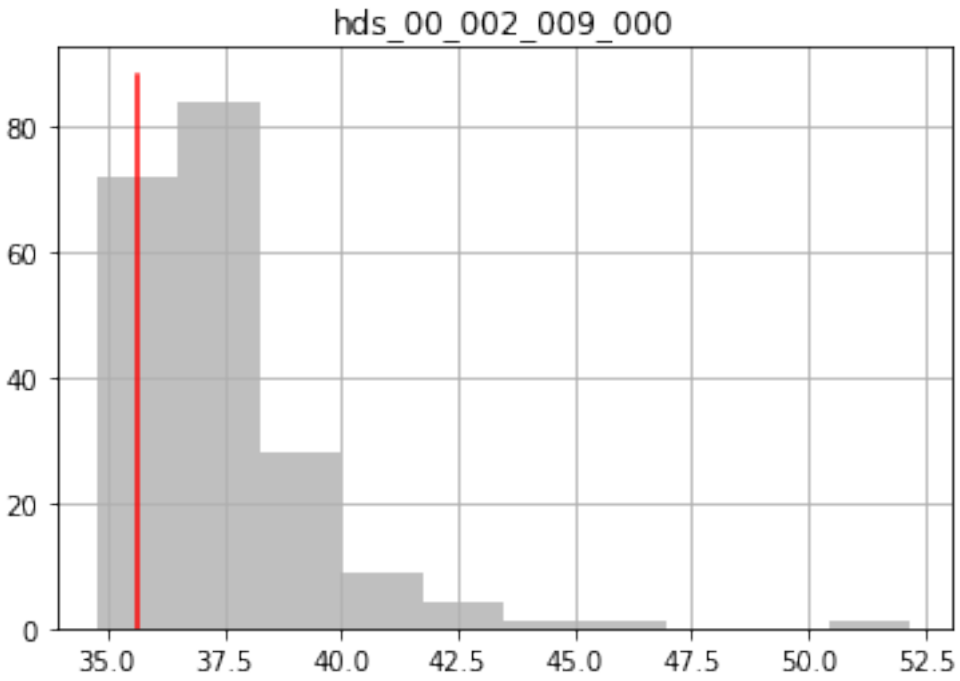


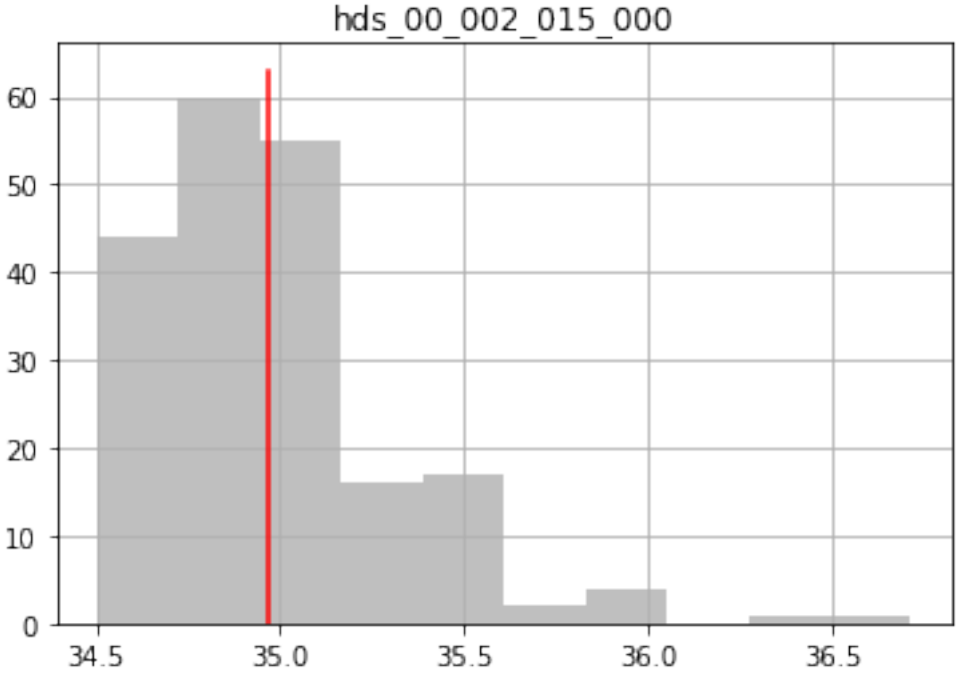


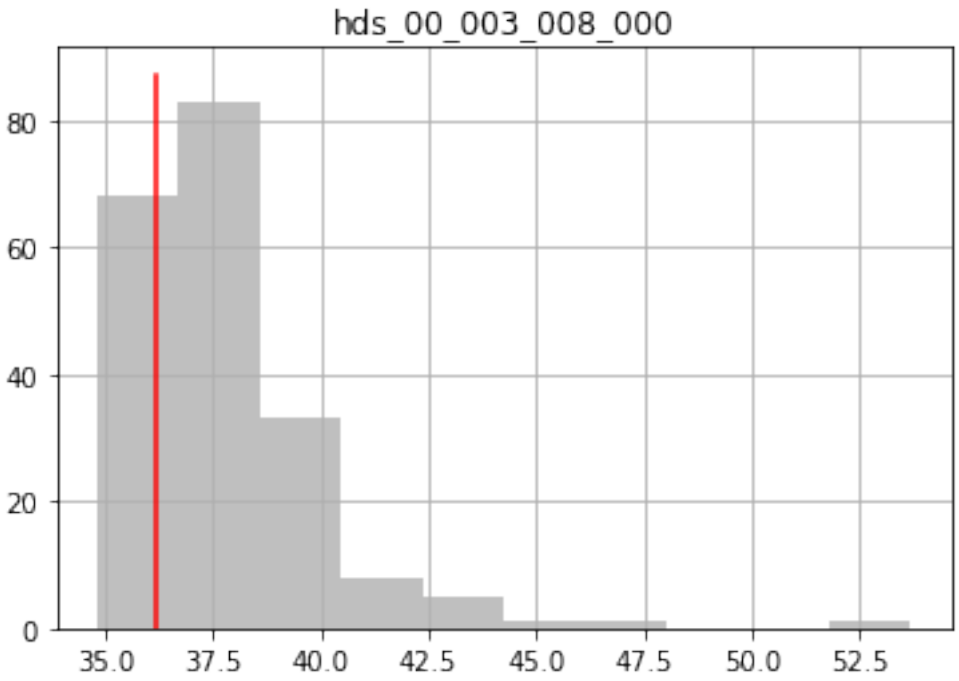
observations:

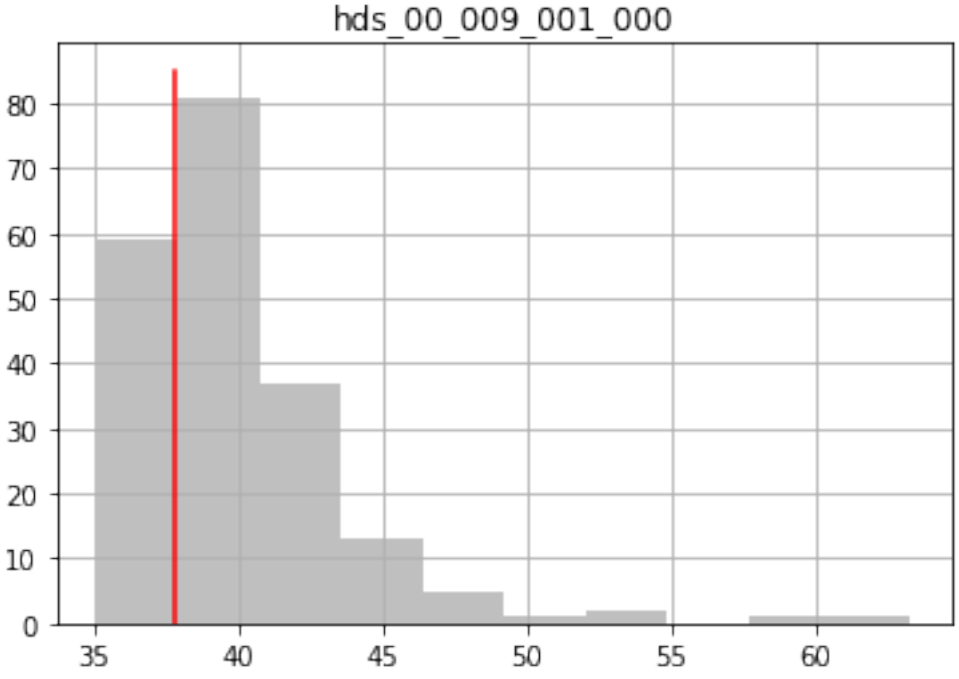
```
In [26]: for oname in pst.nnz_obs_names:
             ax = plt.subplot(111)
             obs_df.loc[:,oname].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[oname,"obsval"],obs.loc[oname,"obsval"]],ax.get_ylim(),"r")
             ax.set_title(oname)
             plt.show()
```

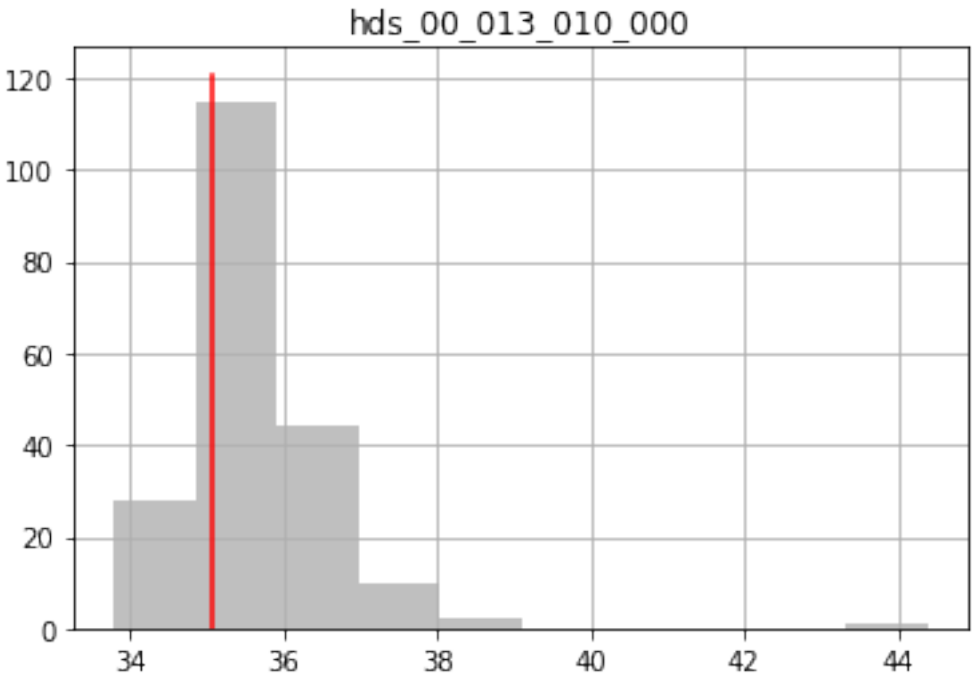


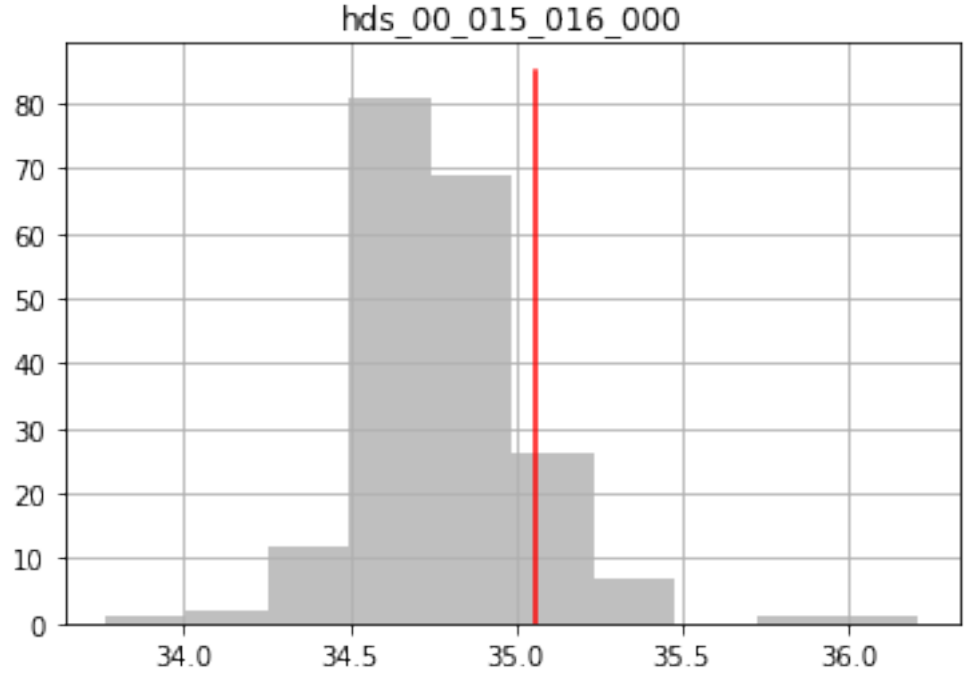

hds_00_002_009_000
80
60
40
20
0
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5

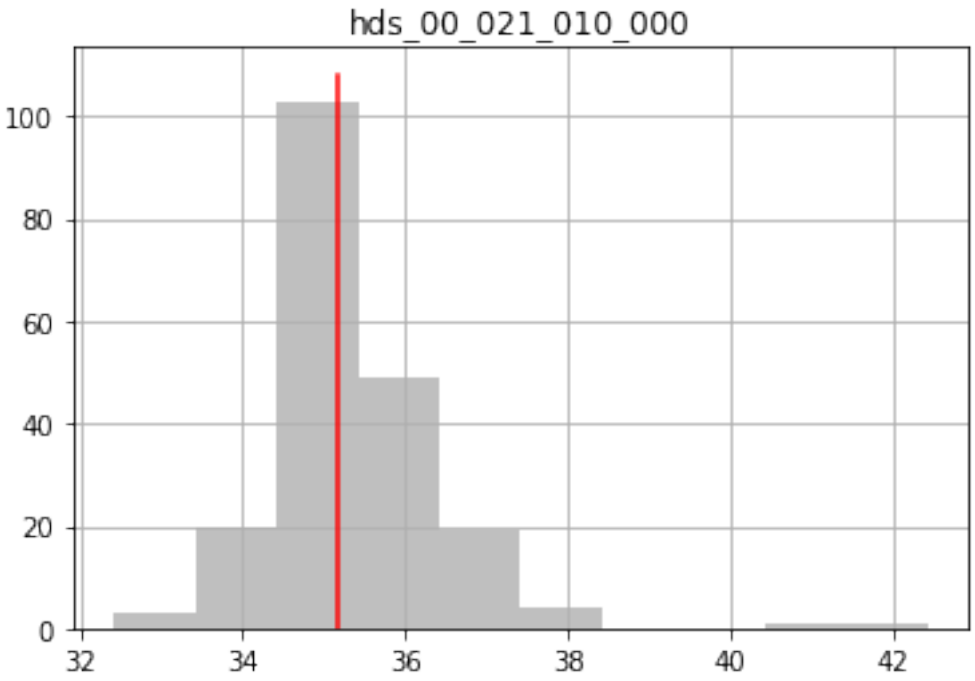

hds_00_002_015_000
60
50
40
30
20
10
0
34.5
35.0
35.5
36.0
36.5

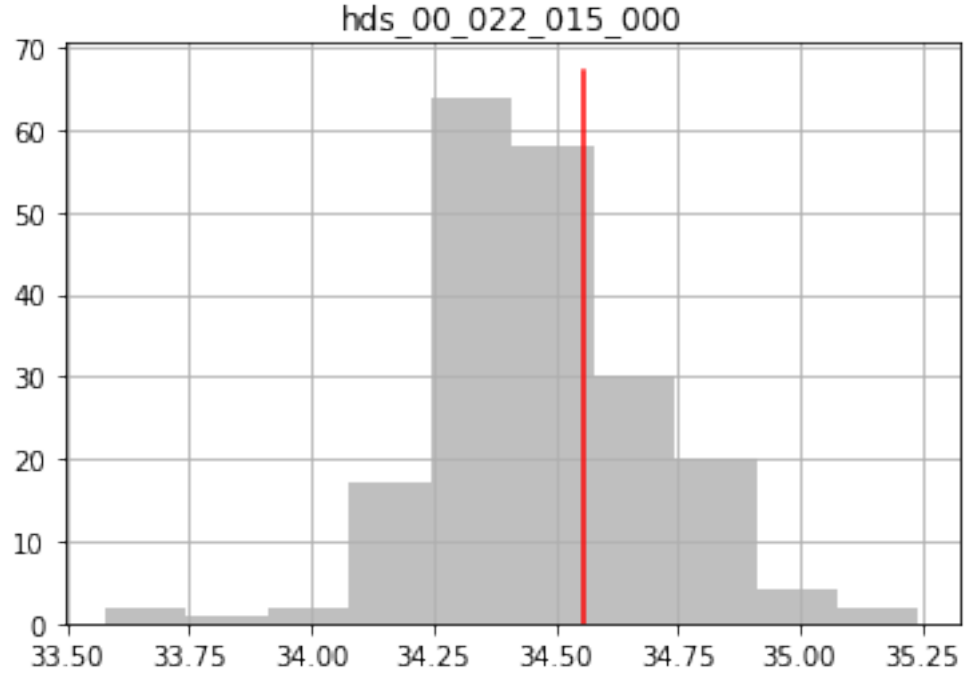

hds_00_003_008_000
80
60
40
20
0
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5

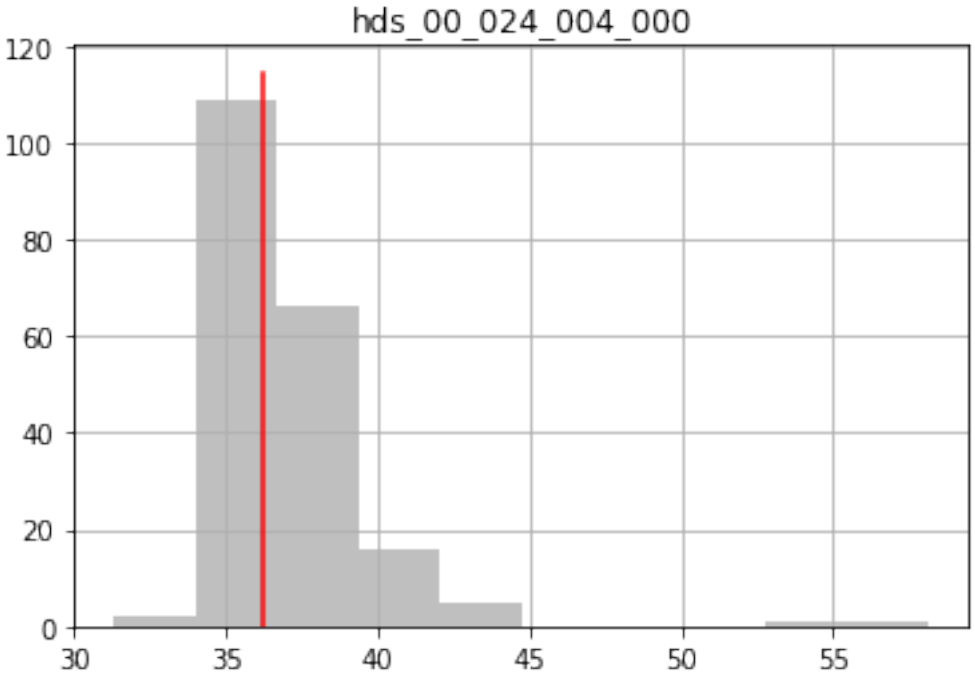

hds_00_009_001_000
80
70
60
50
40
30
20
10
0
35
40
45
50
55
60

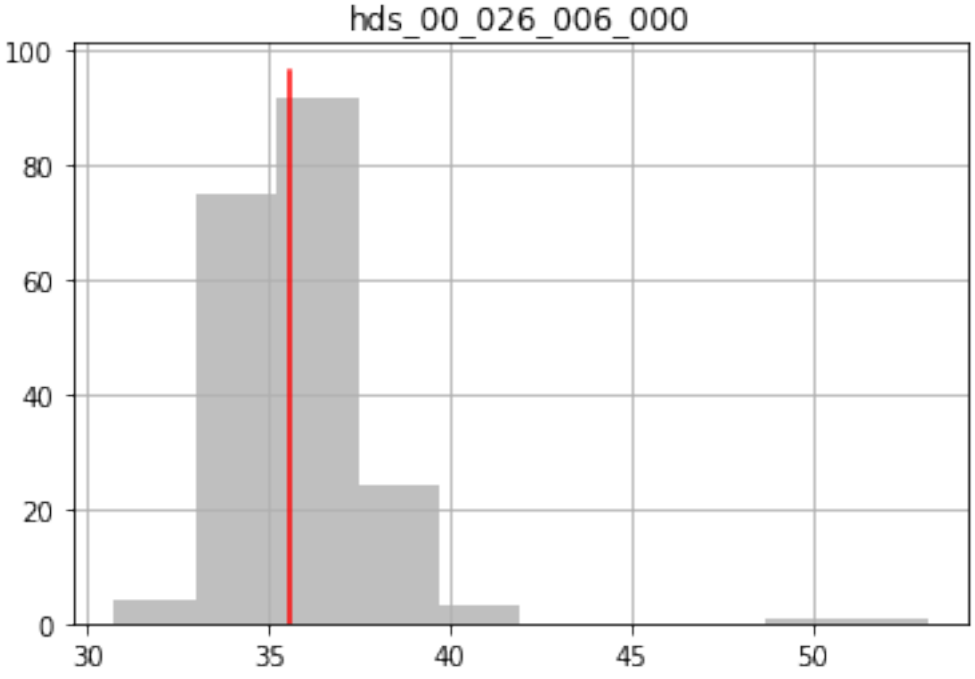

hds_00_013_010_000
120
100
80
60
40
20
0
34
36
38
40
42
44

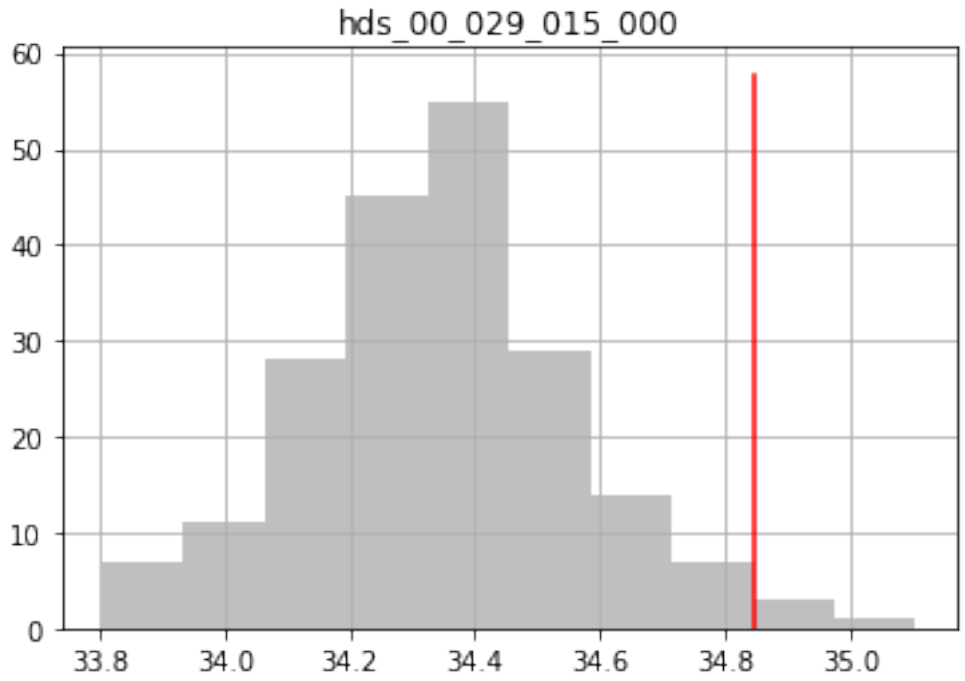

hds_00_015_016_000
80
70
60
50
40
30
20
10
0
34.0
34.5
35.0
35.5
36.0

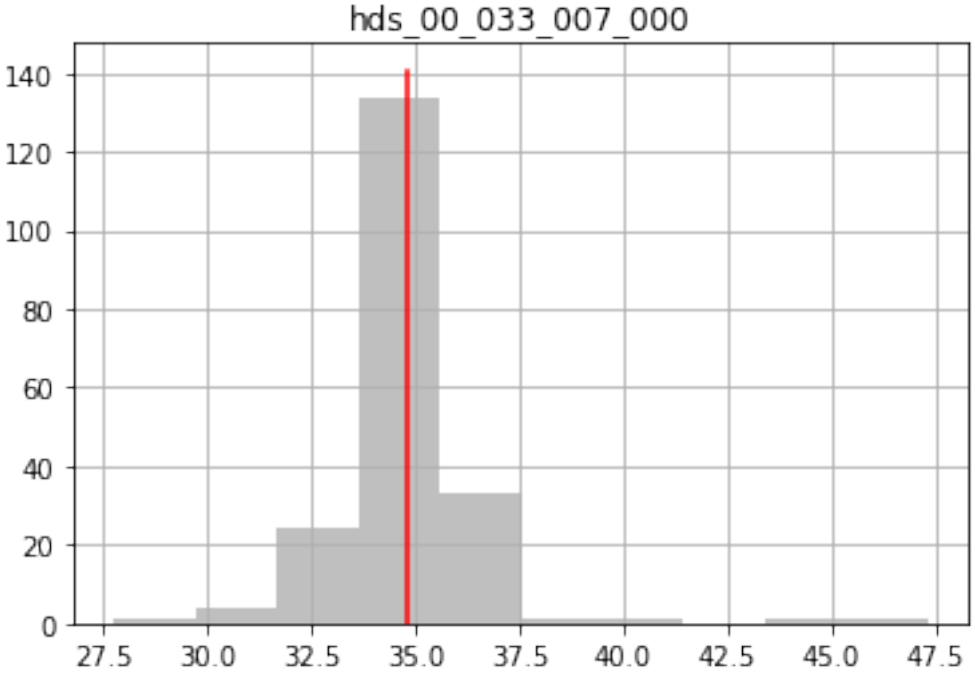

hds_00_021_010_000
100
80
60
40
20
0
32
34
36
38
40
42

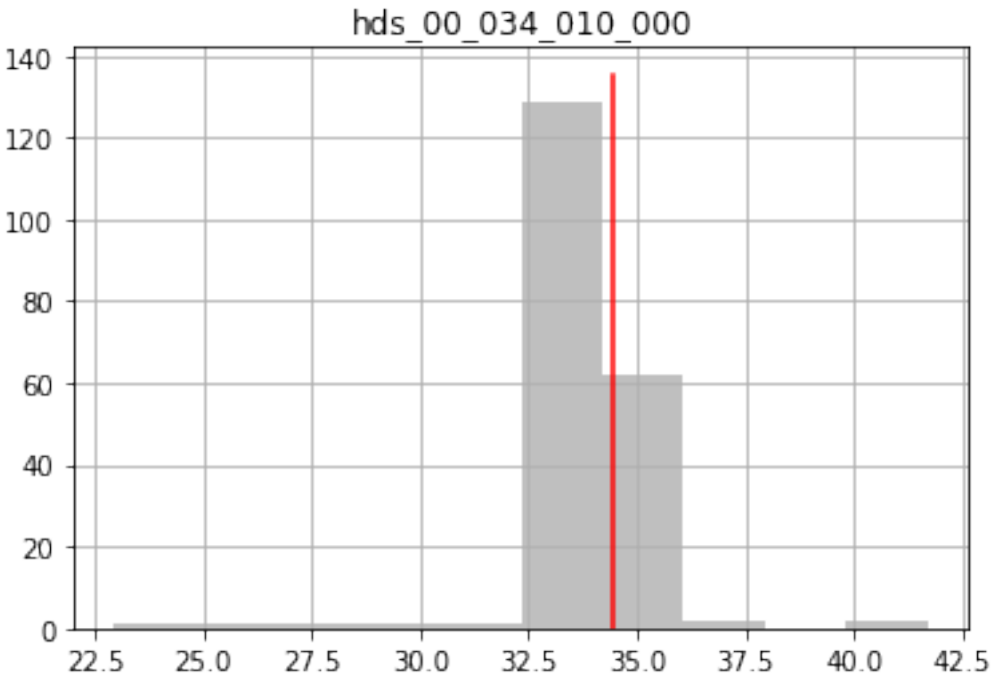

hds_00_022_015_000
70
60
50
40
30
20
10
0
33.50
33.75
34.00
34.25
34.50
34.75
35.00
35.25


hds_00_024_004_000
120
100
80
60
40
20
0
30
35
40
45
50
55


hds_00_026_006_000
100
80
60
40
20
0
30
35
40
45
50


hds_00_029_015_000
60
50
40
30
20
10
0
33.8
34.0
34.2
34.4
34.6
34.8
35.0


hds_00_033_007_000
140
120
100
80
60
40
20
0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5


hds_00_034_010_000
140
120
100
80
60
40
20
0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5