The Backdrop

I make an assumption that we are in very early stages of engagement with 3D Technologies. Even if false, it helps explain my thought process behind my analysis and recommendations.

First Why, Then How, Finally What (now)

Amazon's Leadership Principles are and always have been the guiding force behind any work I do. First "Why", then "How" and finally "What now" approach I took (from How great leaders inspire actions by Simon Sinek), plays well with Amazon's Leadership Principles. And hence the format.

Why (The Purpose)

I would like 3D Technologies to truly believe our team has their company's best interest as **our** highest priority. We need to win the trust the key players in the engagement by delivering continuous results.

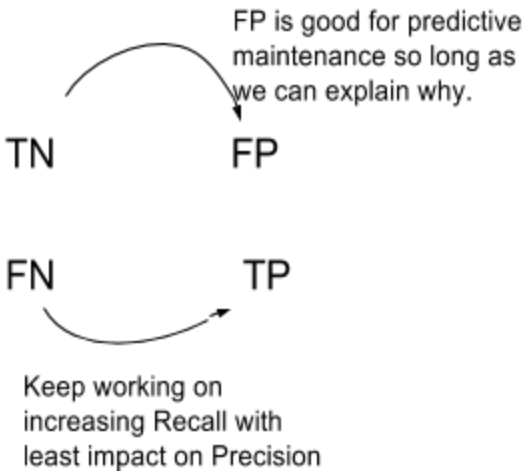
How (The Strategy)

- **By understanding the scale of the problem.**
    - Current cost of failure per month or year. And per device.
    - Cost of repairing/replacing the device.
    - This derives: (Note: These numbers will evolve over time as we collect/analyze more data)
        - Savings in replacing a device that will truly fail.
        - Cost of replacing a device that may not fail.
        - And hence, at what prediction probability of failure does it make sense to drive a decision. (Nash's equilibrium ("Utitlity"))
        - Eventually leads to most important questions: Money saved using our services per year.
    - Note: I implemented many fancy algorithms in AT&T without doing above (**unsexy and boring**) analysis. My repeated requests to do basic BI even after showing sample interactive reports using d3.js/Kibana were shot down. Apart from boosting the sexiness of my resume, those implementations were not worth much despite costing AT&T millions of dollars in implementations. The few algorithms that really worked were based on interactive BI reports and very little math(KMeans Clustering, Change Point Detection, and other Linear models). Only exception was using Syntax parsing using AT&T Watson, which is a deep learning framework developed by AT&T and putting this in context in the BI reports (d3.js/dc.js based)
- **By studying existing maintenance practices by customer**. Eg: Previous models that were historically used.
    - Eg: After basic data analysis/wrangling, and initial finding,
        - Site visit to the plant, discussing with Floor/Line managers/technitians
        - (Based on my experience in automation of Dell Desktop assembly plant in Nashville, TN in 2005)
- **Above two help the team be engaged and personally relate to the problem.**
- **By "iteratively" delivering actionable intelligence from the data we collect.**
    - First iteration, approximately 6-8 weeks, and hopefully in 4-6 week cycles after that.
    - This approch keeps customers engaged continuously.

- **By engaging with customer and internal operations teams**, "continuously":
    - To the best of our ability, explain the type of failures and how far ahead we can predict a failure.
        - I expect this kind of communication to give better clues for feature engineering.
        - Many times, customer knows their equipment better than we do. (Might lead to better features)
    - Improve data quality.
    - Improve story telling techniques by improved "interactive" visualizations (Kibana, custom d3.js, ZoomData etc)
        - Evolve backend stores to improve interactiveness (Eg: "orc" in hdfs, Redshift, Elastic, Druid etc)
    - Hypothesis testing over time to analyze reliability of our models.
        - Eg: Given our model predicted failure, number of days to actual failure.
    - Reduce "device data output → data ingest → ML decision → Customer Action cycle time.

What (Current Insights and Proposed Actions)

- Every insight delivered and action proposed is derived based on my thought process behind the "Why" and the "How".
- Consider this as the first step in a journey.
    - This is where we are (Insights)
    - This is what we propose that our customer and our internal teams must do.
- Overall Approach: Using Confusion matrix



Step 1: Basic Exploratory Data Analysis

In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Basic per device metrics and Data Quality

In [6]:
```
df = pd.read_csv("../data/device_failure.csv")
df.loc[:,'date'] = pd.to_datetime(df['date'])
```

In [7]:
```
df.head()
```

Out[7]:

	date	device	failure	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attribute7	attribute8	attribute9
0	2015-01-01	S1F01085	0	215630672	56	0	52	6	407438	0	0	7
1	2015-01-01	S1F0166B	0	61370680	0	3	0	6	403174	0	0	0
2	2015-01-01	S1F01E6Y	0	173295968	0	0	0	12	237394	0	0	0
3	2015-01-01	S1F01JE0	0	79694024	0	0	0	6	410186	0	0	0
4	2015-01-01	S1F01R2B	0	135970480	0	0	0	15	313173	0	0	3

In [10]:
```
df.isnull().describe()
```

Out[10]:

	date	device	failure	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attribute7	attribute8	attribu
count	124494	124494	124494	124494	124494	124494	124494	124494	124494	124494	124494	12449
mean	0	0	0	0	0	0	0	0	0	0	0	0
std	0	0	0	0	0	0	0	0	0	0	0	0
min	False	False	False	False	False	False	False	False	False	False	False	False
25%	0	0	0	0	0	0	0	0	0	0	0	0
50%	0	0	0	0	0	0	0	0	0	0	0	0
75%	0	0	0	0	0	0	0	0	0	0	0	0
max	False	False	False	False	False	False	False	False	False	False	False	False

In [11]:
```
df["date"].describe()
```

Out[11]:
```
count                  124494
unique                    304
top       2015-01-03 00:00:00
freq                     1163
first     2015-01-01 00:00:00
last      2015-11-02 00:00:00
Name: date, dtype: object
```

Good. At least no null values

In [9]:
```
total_obs = df.shape[0]
unique_dev = df.device.unique().shape[0]
print ("Total Observations        : %s"%total_obs)
print ("Unique Devices            : %s"%unique_dev)
print ("Unique devices that failed: %d"%df[df['failure'] == 1].device.unique().size)
print ("Avg number of observations per device: %f"%(float(total_obs)/unique_dev))
```

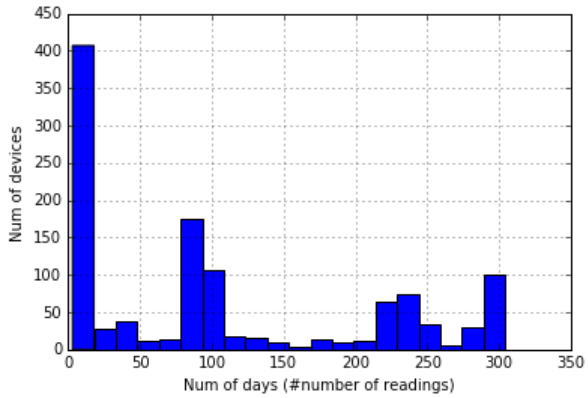
```
Total Observations        : 124494
Unique Devices            : 1168
Unique devices that failed: 106
Avg number of observations per device: 106.587329
```

Distribution of number of readings for devices

NumDevices/day that were observed --> points to systemic ingestion/sampling problem

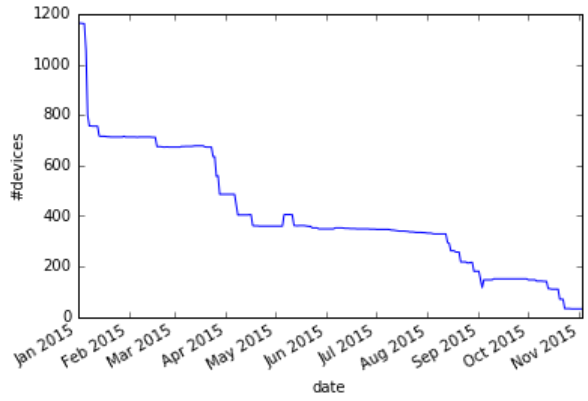
In [15]:
```
df.groupby(["device"]).count()["attribute1"].hist(bins=20)
plt.xlabel("Num of days (#number of readings)")
plt.ylabel("Num of devices")
```

Out[15]: <matplotlib.text.Text at 0x7efef8508d50>



In [16]:
```
df.groupby(["date"]).count()["device"].plot.line()
plt.ylabel("#devices")
```

Out[16]: <matplotlib.text.Text at 0x7efef83d85d0>



See if there are gaps in dates for sensor data. See how bad it is

In [19]:
```
df_date_min = df.groupby("device").min()["date"]
df_date_max = df.groupby("device").max()["date"]
df_date_count = df.groupby("device").count()["date"]
df2 = pd.concat([df_date_min, df_date_max, df_date_count], axis=1)
df2.columns = ["start", "end", "xcount"]
df2.loc[:,"diff_dates"] = (df2.end - df2.start).dt.days+1
df2.loc[:,'deltas'] = df2.diff_dates - df2.xcount
df2.loc[:,'missing'] = df2.deltas.map(lambda x: x != 0)
df2.groupby("missing").count()["start"]
```

Out[19]:
```
missing
False    997
True     171
Name: start, dtype: int64
```

At least majority of cases, readings are continuous

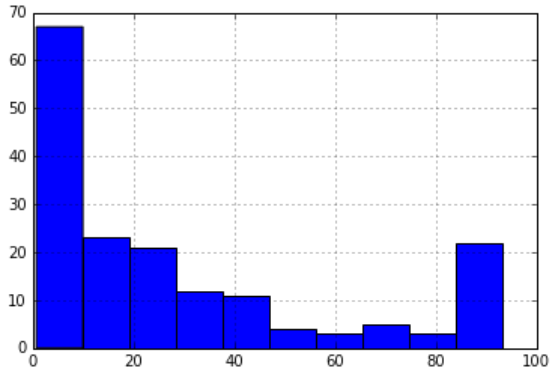
For missing data, how many days are we missing?

In [21]:
```
missing = df2[df2["missing"] == True]
missing.loc[:,"miss_pct"] = (missing["deltas"]/missing["diff_dates"])*100
missing.miss_pct.describe()
```

Out[21]:
```
count    171.000000
mean      27.568382
std       30.822808
min        0.653595
25%        0.701754
50%       15.116279
75%       43.494856
max       93.129771
Name: miss_pct, dtype: float64
```

In [22]:
```
missing.miss_pct.hist()
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7efef830b890>
```



Summary

We have data issues, but I think we have enough data to get started

Work with internal and external operations in parallel

In [ ]:
```

```

Step 2: Time Series Explorator Data Analysis ¶

Get a feel for how time series data looks like for various features

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [3]:
```
df = pd.read_csv("../data/device_failure.csv")
df.columns = ['date', 'device', 'failure', 'a1', 'a2','a3','a4','a5','a6','a7','a8','a9']
fcols = ['a1', 'a2','a3','a4','a5','a6','a7','a8','a9']
df.loc[:,'date'] = pd.to_datetime(df['date'])
```

Ideally, would save this off in seperate data stores for faster retrieval

Like in orc/parquet format (hdfs) to retain format types (especially dates), float/int stored very efficiently

In [4]:
```
failed_devs = pd.DataFrame(df[df['failure'] == 1].device.unique())
failed_devs.columns = ["device"]
failed_devs_hist = pd.merge(df, failed_devs, on=["device"])

good_devs = pd.DataFrame(list(set(df.device.unique()) - set(failed_devs["device"])))
good_devs.columns = ["device"]
good_devs_hist = pd.merge(df, good_devs, on=["device"])
```

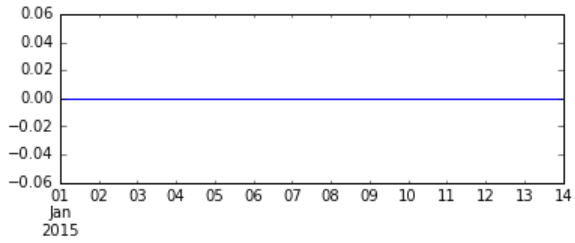
Explore how good vs bad devices data looks for various features

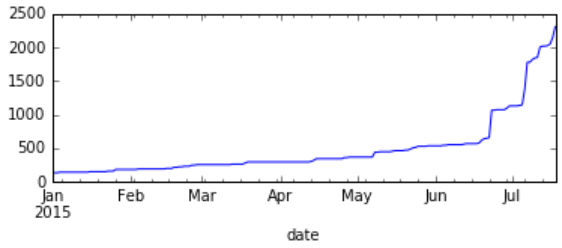
Just priliminary analysis. See which transformations make sense so we can build modules.

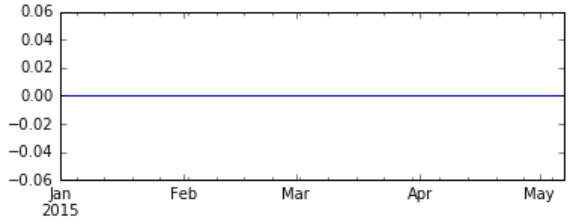
In [5]:
```
def plot_history(tdf, feature, devname):
    fdev = tdf[tdf["device"] == devname]
    fdev.set_index("date", inplace=True)
    fdev[feature].plot()

def plot_sample_history(tdf, dev_list_df, sample_cnt, feature):
    #Get a sample of devices and their history
    sample_dev_df = dev_list_df.sample(sample_cnt)
    sample_dev_hist = pd.merge(tdf, sample_dev_df, on=["device"])
    for device in sample_dev_df["device"]:
        fig, axs = plt.subplots(1)
        fig.set_size_inches(6,2)
        plot_history(sample_dev_hist, feature, device)
```

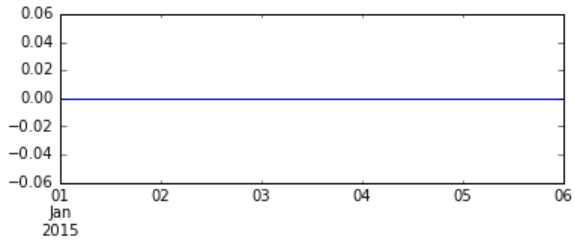
In [6]:
```
plot_sample_history(failed_devs_hist, failed_devs, 3, "a2")
```

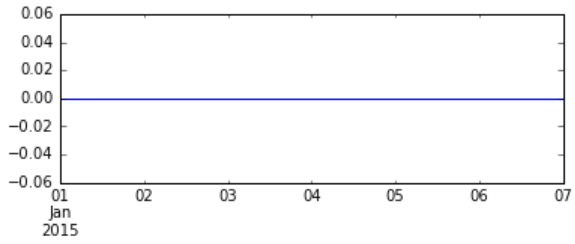


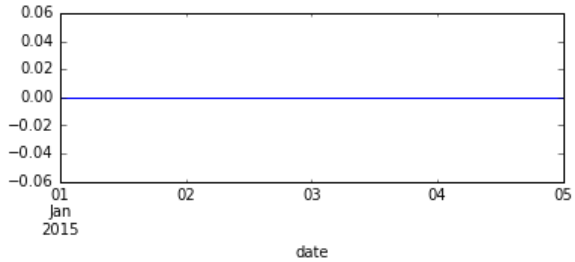




In [7]:
```
plot_sample_history(good_devs_hist, good_devs, 3, "a2")
```







As Good Vs Bad analysys for "a2" shows

Failures happen quickly within days, once we start seeing signal on a2

Most of good devices don't show any signal activity in a2

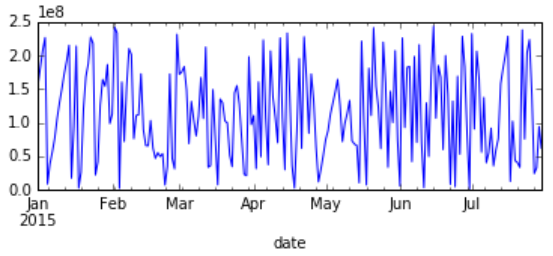
Range of values varies widely. So, may have to use natural logs, their differentials and derive features from these

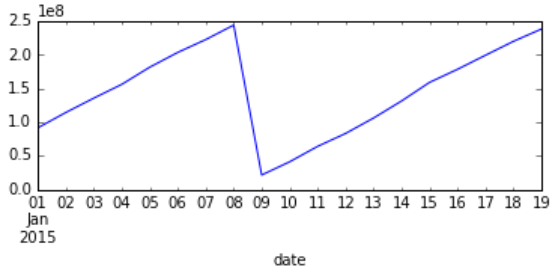
Note: I have taken samples in 10 many times to come to this conclusion (not just for a2, but all features)

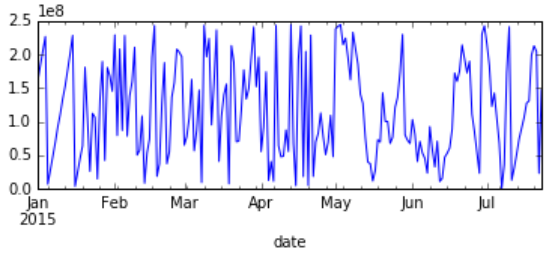
a1: Cannot seem to find anything different between good and bad devices.

Try other approach like using slope and bias of regression lines as features. TBD later

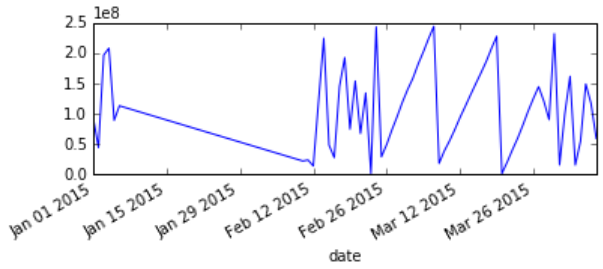
```
In [8]: plot_sample_history(failed_devs_hist, failed_devs, 3, "a1")
```

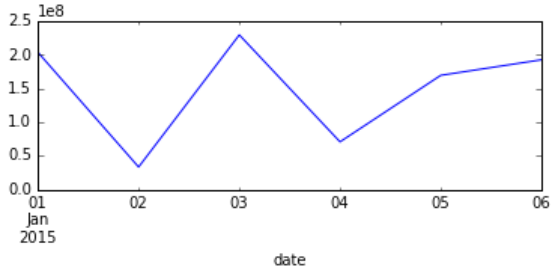


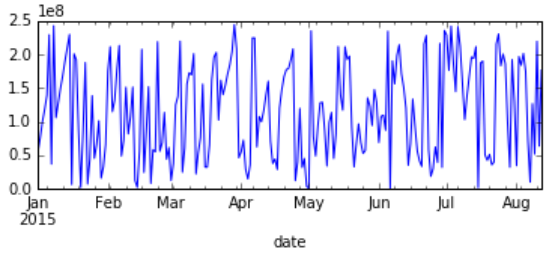




In [9]:
```
plot_sample_history(good_devs_hist, good_devs, 3, "a1")
```







a7 and a8 are the same feature!

In [10]:
```
df[df["a7"] != df["a8"]]
```

Out[10]:

	date	device	failure	a1	a2	a3	a4	a5	a6	a7	a8	a9

Summary after going through each feature

a2, a7 and a4 show promising difference between good vs bad like shown above. Focus first

a5: Don't see much signal here. But confirm.

a6: This looks like a counter that just keeps going up. Not expecting much, but confirm

a1: May have to try different approach. Good vs bad show no difference with current features

a8: same as a7. Ignore

In [ ]:
```
```

Step 3: a2 Feature Analysis

Use this feature to build module of most common functions/graphs used

/utils/DataAggregation.py: Dataframe Convenience functions

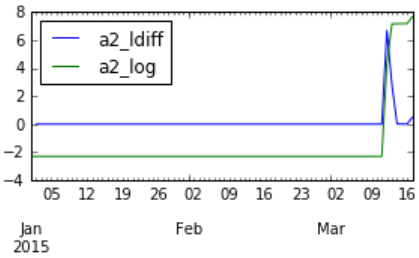
/utils/AlgoUtils.py : Algorithm Convenience functions

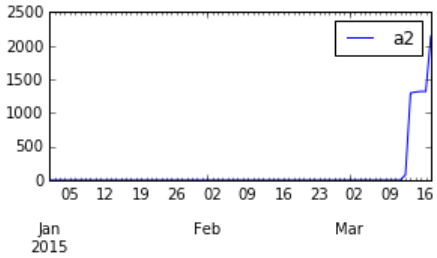
We will use this template and python modules to analyse other features

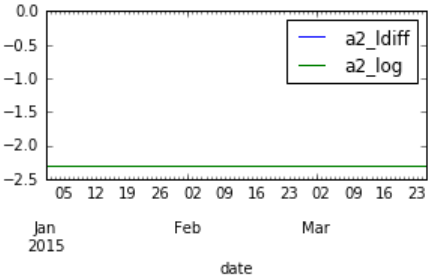
In [1]:
```
import pandas as pd
import sys
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
sys.path.append('../utils')
import DataAggregation as da
import AlgoUtils as au
cmap_bold = ListedColormap(['#00FF00','#FF0000'])
```

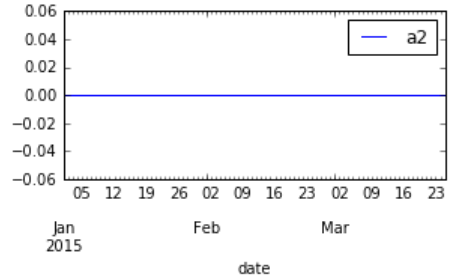
In [2]:
```
dd = da.GetFrames("../data/device_failure.csv", "a2")
```

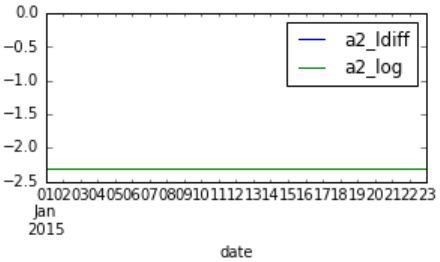
In [13]:
```
dd.plot_sample_history(dd.failed_devs["device"],10)
```

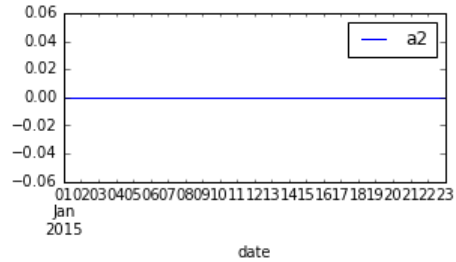

a2_ldiff
a2_log
Jan
2015
Feb
Mar
date

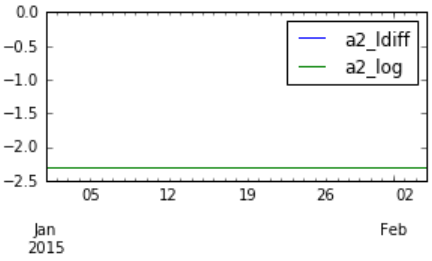

a2
Jan
2015
Feb
Mar
date

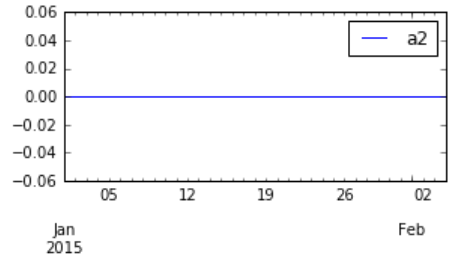

a2_ldiff
a2_log
Jan
2015
Feb
Mar
date

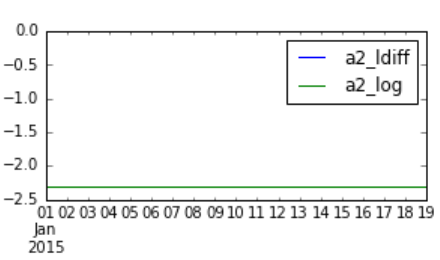

a2
Jan
2015
Feb
Mar
date

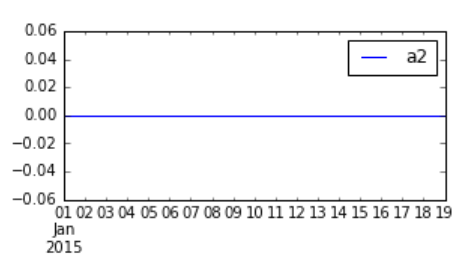

a2_ldiff
a2_log
Jan
2015
date

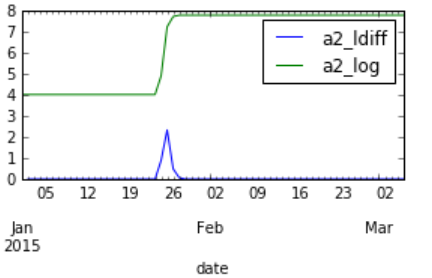

a2
Jan
2015
date

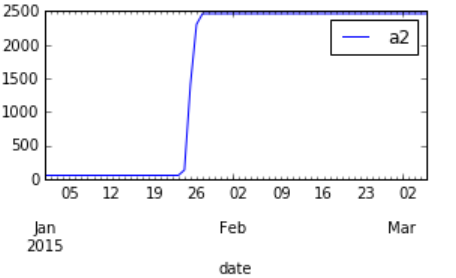

a2_ldiff
a2_log
Jan
2015
Feb
date

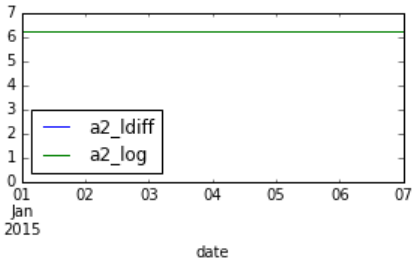

a2
Jan
2015
Feb
date

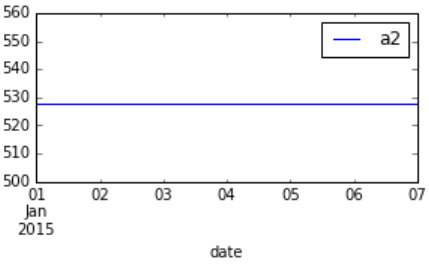

a2_ldiff
a2_log
Jan
2015
date

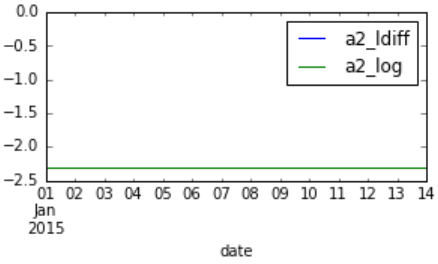

a2
Jan
2015
date

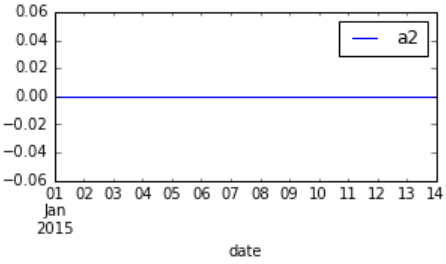

a2_ldiff
a2_log
Jan
2015
Feb
Mar
date

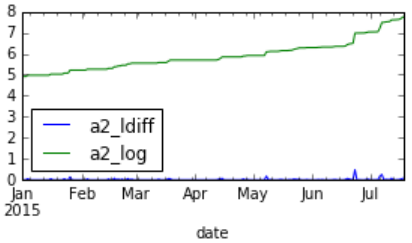

a2
Jan
2015
Feb
Mar
date

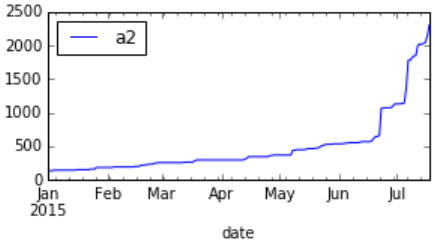

a2_ldiff
a2_log
01
Jan
2015
02
03
04
05
06
07
date

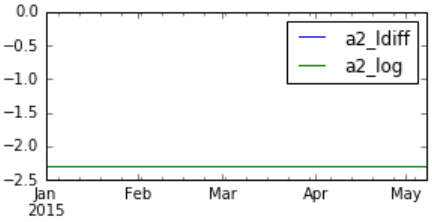

a2
01
Jan
2015
02
03
04
05
06
07
date

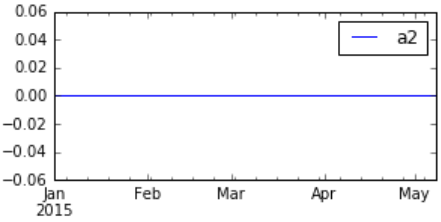

a2_ldiff
a2_log
01
Jan
2015
02
03
04
05
06
07
08
09
10
11
12
13
14
date


a2
01
Jan
2015
02
03
04
05
06
07
08
09
10
11
12
13
14
date

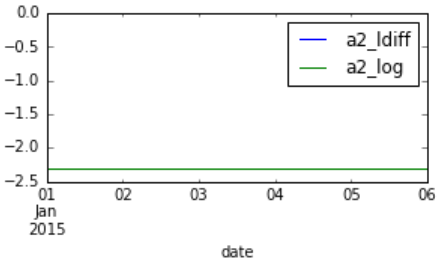

a2_ldiff
a2_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

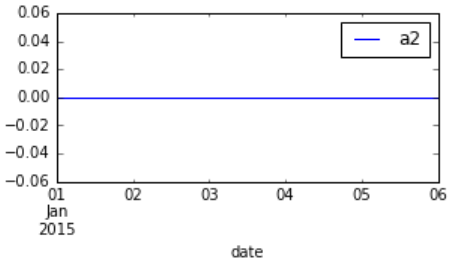

a2
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

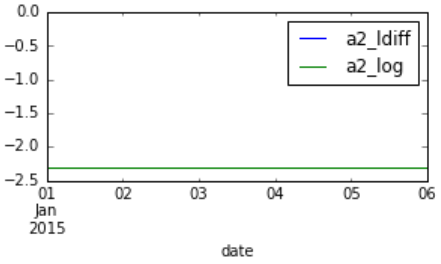

a2_ldiff
a2_log
Jan
2015
Feb
Mar
Apr
May
date

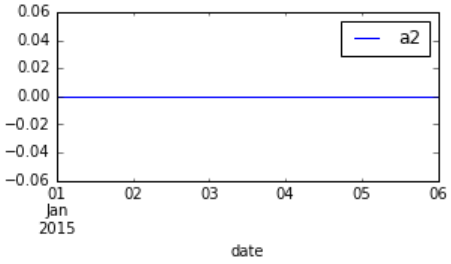

a2
Jan
2015
Feb
Mar
Apr
May
date

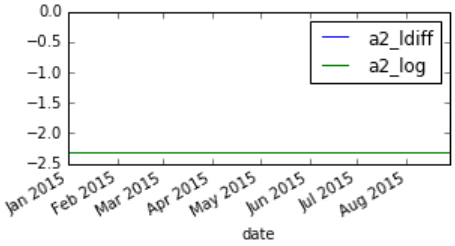
In [14]:
```
dd.plot_sample_history(dd.good_devs["device"],10)
```

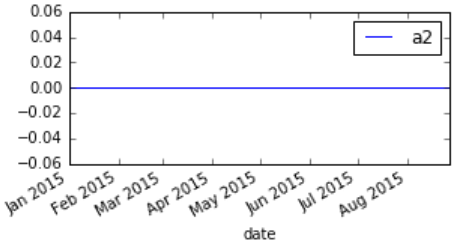

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
02
03
04
05
06
Jan
2015
date

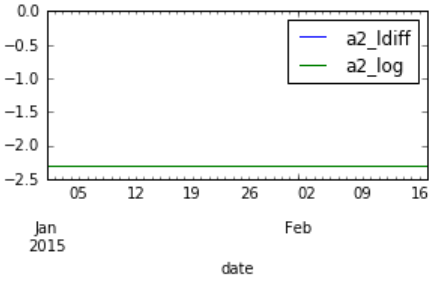

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
02
03
04
05
06
Jan
2015
date

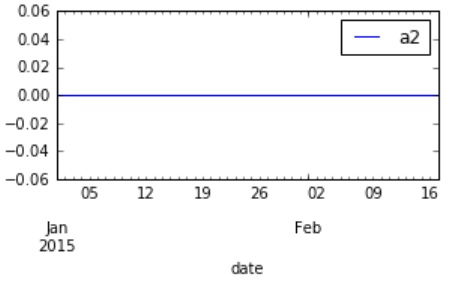

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
02
03
04
05
06
Jan
2015
date

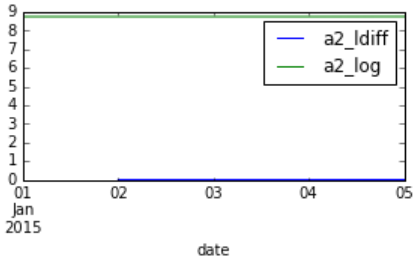

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
02
03
04
05
06
Jan
2015
date

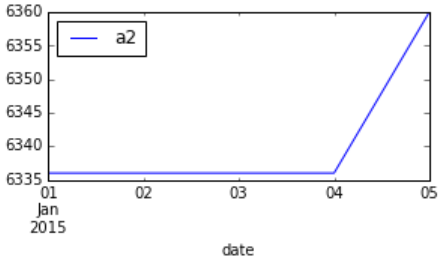

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
date

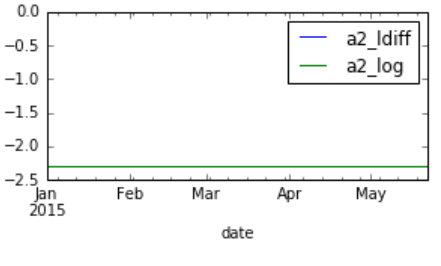

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
date

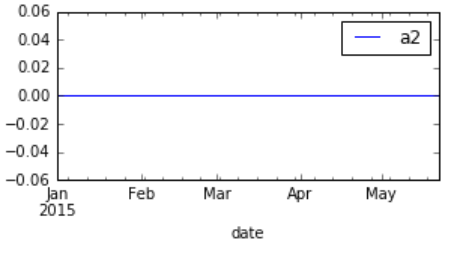

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05
12
19
26
02
09
16
Jan
2015
Feb
date

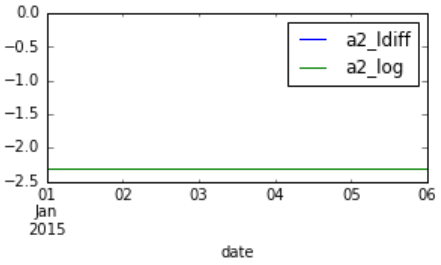

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05
12
19
26
02
09
16
Jan
2015
Feb
date

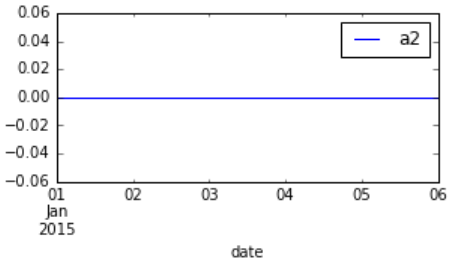

a2_ldiff
a2_log
9
8
7
6
5
4
3
2
1
0
01
02
03
04
05
Jan
2015
date

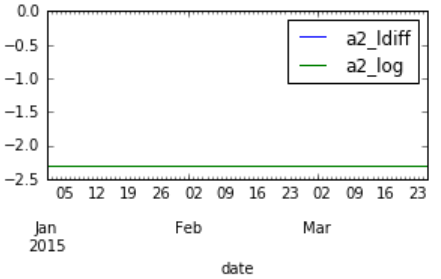

a2
6360
6355
6350
6345
6340
6335
01
02
03
04
05
Jan
2015
date

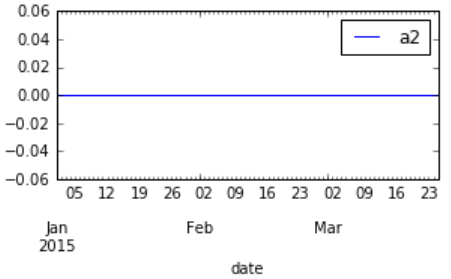

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
Feb
Mar
Apr
May
2015
date

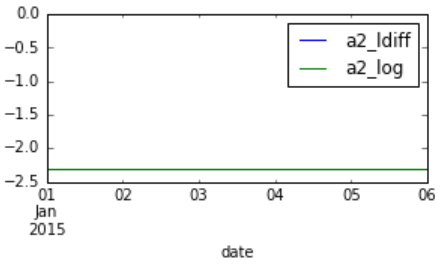

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
Feb
Mar
Apr
May
2015
date

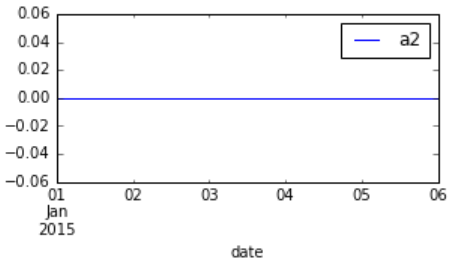

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
02
03
04
05
06
Jan
2015
date

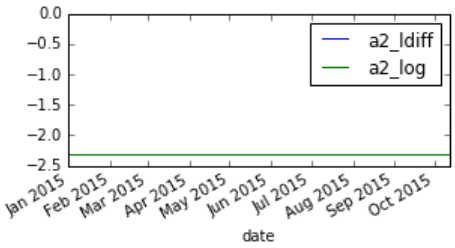

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
02
03
04
05
06
Jan
2015
date

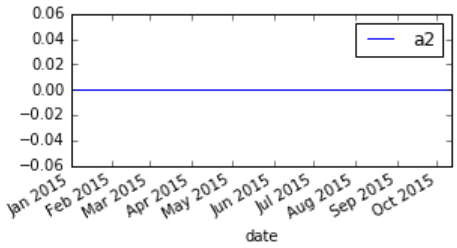

a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05
12
19
26
02
09
16
23
02
09
16
23
Jan
2015
Feb
Mar
date


a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05
12
19
26
02
09
16
23
02
09
16
23
Jan
2015
Feb
Mar
date


a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
02
03
04
05
06
Jan
2015
date

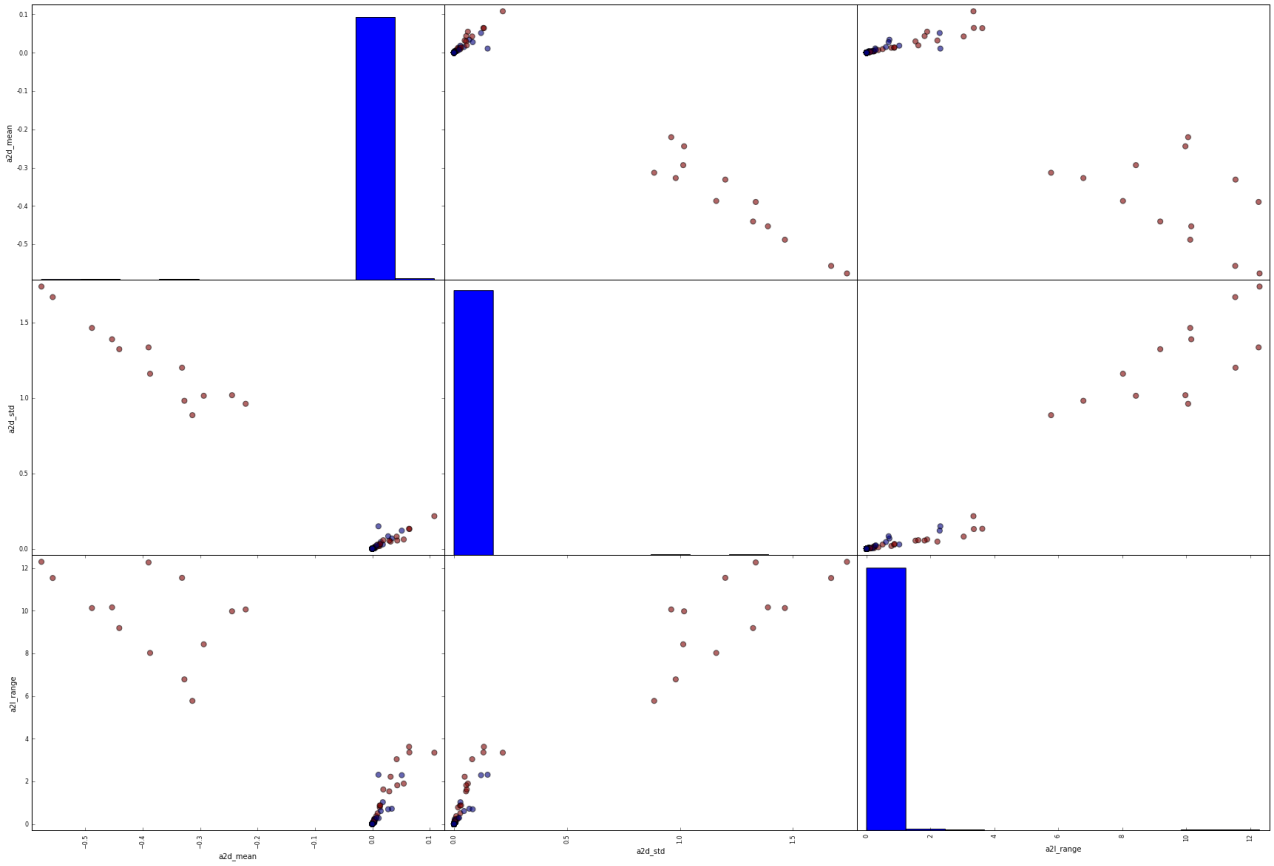

a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
02
03
04
05
06
Jan
2015
date


a2_ldiff
a2_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
date


a2
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
date

In [15]:
```
sfeature = dd.sfeature
fcols_tmp = [sfeature +"d_mean", sfeature +"d_std", sfeature +"l_range"]
df_sfeature = dd.df_sfeature
pd.scatter_matrix(df_sfeature[fcols_tmp], figsize=(30,20), s=200, c=df_sfeature["failure"], alpha=0.6)
```

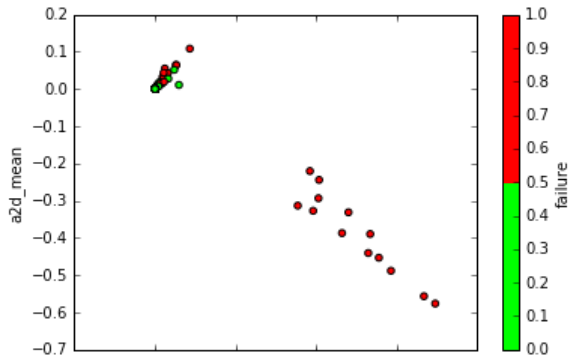
Out[15]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0d4fe62a50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0d50c61510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0d502907d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0d5004c190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0d50b6e650>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0d50990e50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0d5088cf90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0d4ffa9e50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0d50918f10>]], dtype=object)
```



Nice! See seperation between good and bad devices.

In [21]:
```
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_mean", c="failure", colormap=cmap_bold)
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0d50d08b10>
```



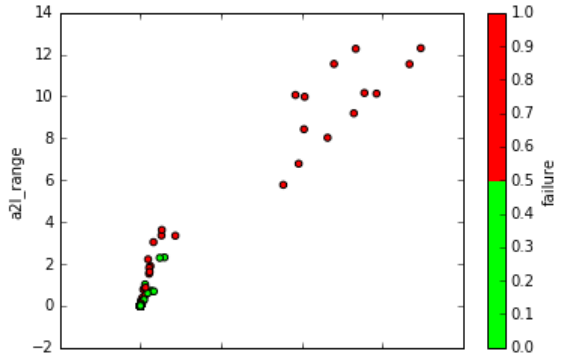
NICE!! a2d_std vs a2d_mean

+ve correlation for good devices and -ve correlation for bad devices!

Quadratic Discriminant Analysis should really pick this up well

In [18]:
```
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"l_range", c="failure", colormap=cmap_
bold)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0d50435fd0>



NICE!! a2d_std vs a2l_range

High correlation of bad devices and failure. Very good.

QDA or Linear Regression should pick this up

I really think QDA will come up on top, but try other algorithms just in case

In [3]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [4]:
```
df_sfeature = dd.df_sfeature
sfeature = dd.sfeature
df_results = au.run_algo_analysis(df_sfeature, sfeature, fcols, algos_str, algos_dd)
```

```
----------
LogisticRegression:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.64
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.91
Cross-val-score(f1)       = 0.91
----------
LogisticRegressionB:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.27
Cross-val-score(precision)= 0.78
Cross-val-score(f1)       = 0.78
----------
KNeighborsClassifier:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.77
Cross-val-score(f1)       = 0.77
----------
LinearDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.67
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.20
Cross-val-score(precision)= 0.86
Cross-val-score(f1)       = 0.86
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.79
Cross-val-score(f1)       = 0.79
----------
LogisticRegression:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.21
Cross-val-score(precision)= 0.91
Cross-val-score(f1)       = 0.91
----------
LogisticRegressionB:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.27
Cross-val-score(precision)= 0.77
Cross-val-score(f1)       = 0.77
----------
KNeighborsClassifier:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.77
Cross-val-score(f1)       = 0.77
----------
LinearDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.21
Cross-val-score(precision)= 0.90
Cross-val-score(f1)       = 0.90
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.25
Cross-val-score(precision)= 0.83
Cross-val-score(f1)       = 0.83
----------
LogisticRegression:d_std:l_range
Cross-val-score(roc_auc) = 0.67
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.21
Cross-val-score(precision)= 0.90
```

```
Cross-val-score(f1)       = 0.90
----------
LogisticRegressionB:d_std:l_range
Cross-val-score(roc_auc) = 0.67
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.29
Cross-val-score(precision)= 0.78
Cross-val-score(f1)       = 0.78
----------
KNeighborsClassifier:d_std:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.77
Cross-val-score(f1)       = 0.77
----------
LinearDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.67
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.21
Cross-val-score(precision)= 0.93
Cross-val-score(f1)       = 0.93
----------
QuadraticDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.25
Cross-val-score(precision)= 0.85
Cross-val-score(f1)       = 0.85
----------
LogisticRegression:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.21
Cross-val-score(precision)= 0.93
Cross-val-score(f1)       = 0.93
----------
LogisticRegressionB:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.29
Cross-val-score(precision)= 0.79
Cross-val-score(f1)       = 0.79
----------
KNeighborsClassifier:l_range
Cross-val-score(roc_auc) = 0.65
```

```
/home/vagrant/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Undefine
dMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/vagrant/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Undefine
dMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.77
Cross-val-score(f1)       = 0.77
----------
LinearDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.16
Cross-val-score(precision)= 0.87
Cross-val-score(f1)       = 0.87
----------
QuadraticDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.63
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.24
Cross-val-score(precision)= 0.85
Cross-val-score(f1)       = 0.85
----------
LogisticRegression:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.16
Cross-val-score(precision)= 0.75
Cross-val-score(f1)       = 0.75
----------
LogisticRegressionB:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.27
Cross-val-score(precision)= 0.72
Cross-val-score(f1)       = 0.72
----------
KNeighborsClassifier:d_std
Cross-val-score(roc_auc) = 0.63
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.20
Cross-val-score(precision)= 0.77
Cross-val-score(f1)       = 0.77
----------
LinearDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.12
Cross-val-score(precision)= 0.90
Cross-val-score(f1)       = 0.90
----------
QuadraticDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.63
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.85
Cross-val-score(f1)       = 0.85
----------
LogisticRegression:d_max
Cross-val-score(roc_auc) = 0.64
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.10
Cross-val-score(precision)= 0.53
Cross-val-score(f1)       = 0.53
----------
LogisticRegressionB:d_max
Cross-val-score(roc_auc) = 0.64
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.23
Cross-val-score(precision)= 0.61
Cross-val-score(f1)       = 0.61
----------
KNeighborsClassifier:d_max
Cross-val-score(roc_auc) = 0.62
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.19
Cross-val-score(precision)= 0.63
Cross-val-score(f1)       = 0.63
----------
LinearDiscriminantAnalysis:d_max
```

```
Cross-val-score(roc_auc) = 0.64
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.13
Cross-val-score(precision)= 0.68
Cross-val-score(f1)       = 0.68
----------
QuadraticDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.60
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.18
Cross-val-score(precision)= 0.67
Cross-val-score(f1)       = 0.67
```

In [5]:
```
df_results
```

Out[5]:

	fcols	algo	recall	precision	f1	roc_auc	accuracy
11	d_std:l_range	LogisticRegressionB	0.29	0.78	0.41	0.67	0.92
16	l_range	LogisticRegressionB	0.29	0.79	0.41	0.66	0.93
1	d_mean:d_std:d_max:l_range	LogisticRegressionB	0.27	0.78	0.39	0.65	0.93
6	d_mean:d_std:l_range	LogisticRegressionB	0.27	0.77	0.39	0.65	0.92
21	d_std	LogisticRegressionB	0.27	0.72	0.38	0.66	0.92
4	d_mean:d_std:d_max:l_range	QuadraticDiscriminantAnalysis	0.26	0.79	0.38	0.66	0.92
14	d_std:l_range	QuadraticDiscriminantAnalysis	0.25	0.85	0.38	0.65	0.93
9	d_mean:d_std:l_range	QuadraticDiscriminantAnalysis	0.25	0.83	0.37	0.65	0.93
19	l_range	QuadraticDiscriminantAnalysis	0.24	0.85	0.37	0.63	0.93
0	d_mean:d_std:d_max:l_range	LogisticRegression	0.23	0.91	0.35	0.64	0.93
2	d_mean:d_std:d_max:l_range	KNeighborsClassifier	0.23	0.77	0.34	0.65	0.92
7	d_mean:d_std:l_range	KNeighborsClassifier	0.23	0.77	0.34	0.65	0.92
12	d_std:l_range	KNeighborsClassifier	0.23	0.77	0.34	0.65	0.92
17	l_range	KNeighborsClassifier	0.23	0.77	0.34	0.65	0.92
24	d_std	QuadraticDiscriminantAnalysis	0.23	0.85	0.34	0.63	0.92
26	d_max	LogisticRegressionB	0.23	0.61	0.33	0.64	0.92
10	d_std:l_range	LogisticRegression	0.21	0.90	0.33	0.67	0.93
13	d_std:l_range	LinearDiscriminantAnalysis	0.21	0.93	0.33	0.67	0.93
8	d_mean:d_std:l_range	LinearDiscriminantAnalysis	0.21	0.90	0.33	0.66	0.93
15	l_range	LogisticRegression	0.21	0.93	0.33	0.66	0.93
5	d_mean:d_std:l_range	LogisticRegression	0.21	0.91	0.33	0.65	0.93
3	d_mean:d_std:d_max:l_range	LinearDiscriminantAnalysis	0.20	0.86	0.31	0.67	0.92
22	d_std	KNeighborsClassifier	0.20	0.77	0.30	0.63	0.92
27	d_max	KNeighborsClassifier	0.19	0.63	0.28	0.62	0.92
29	d_max	QuadraticDiscriminantAnalysis	0.18	0.67	0.28	0.60	0.92
18	l_range	LinearDiscriminantAnalysis	0.16	0.87	0.26	0.66	0.92
20	d_std	LogisticRegression	0.16	0.75	0.26	0.66	0.92
28	d_max	LinearDiscriminantAnalysis	0.13	0.68	0.21	0.64	0.92
23	d_std	LinearDiscriminantAnalysis	0.12	0.90	0.21	0.66	0.92
25	d_max	LogisticRegression	0.10	0.53	0.16	0.64	0.92

As expected, QuadraticDiscriminantAnalysis(QDA) (Row index 14 above) produces fairly precise results for with close enough recall rate.

Lets see why Recall(Sensitivity) is so low

In [11]:
```
sfeature = dd.sfeature
algo_str = "QuadraticDiscriminantAnalysis"
fcols = [sfeature + x for x in "d_mean:d_std:l_range".split(":")]
```

In [18]:
```
analysisdf = au.do_clf_validate(df_sfeature, algo_str,algos_dd[algo_str], fcols, "failure")
```

```
Accuracy = 0.94
Confusion Matrix
[[267   4]
 [ 14   7]]
recall_sensitivity = 0.33
precision          = 0.64
f1                 = 0.44
```

In [13]:
```
mispredictdf = analysisdf[analysisdf["failure"] != analysisdf["y_pred"]]
mispredictdf.reset_index(inplace=True)
mispredictdf.columns = ["device", "a2d_std", "failure", "y_pred"]
```

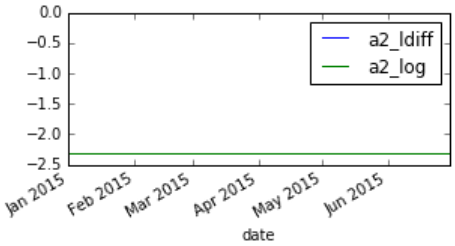
In [14]:
```
mispredictdf
```

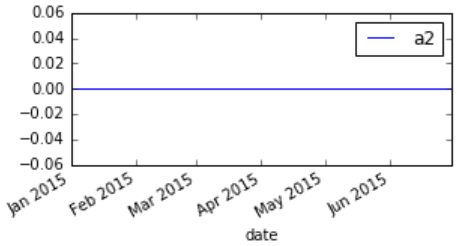
Out[14]:

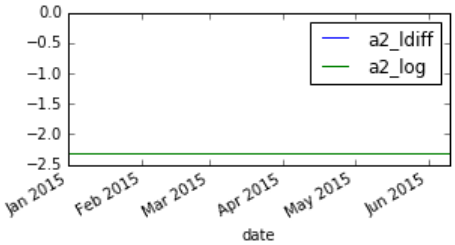
	device	a2d_std	failure	y_pred
0	Z1F1VQFY	0.000000	1	0
1	W1F0Z3KR	0.000000	1	0
2	W1F1BGAB	0.082773	0	1
3	S1F0S4T6	0.000000	1	0
4	Z1F0Q8RT	0.022388	0	1
5	Z1F0MRPJ	0.000000	1	0
6	Z1F130LH	0.000000	1	0
7	S1F0PJJW	0.000000	1	0
8	Z1F0P5D9	0.000000	1	0
9	W1F1B0KF	0.028372	0	1
10	S1F10E6M	0.000000	1	0
11	S1F0CTDN	0.000000	1	0
12	S1F0T2LA	0.000000	1	0
13	W1F0X4FC	0.000000	1	0
14	W1F0NZZZ	0.000000	1	0
15	S1F0TQEJ	0.148782	0	1
16	W1F13SRV	0.000367	1	0
17	W1F0P114	0.000000	1	0

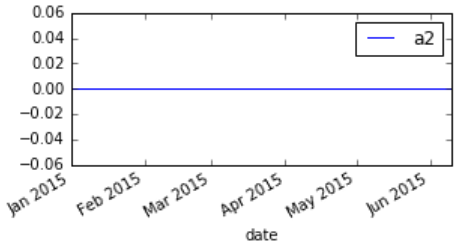
In [15]:
```
mispredict_devs = pd.DataFrame(mispredictdf.device.unique())
mispredict_devs.columns = ["device"]
```

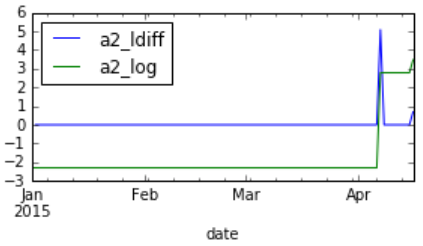
In [17]:
```
dd.plot_sample_history(mispredict_devs["device"],0)
```

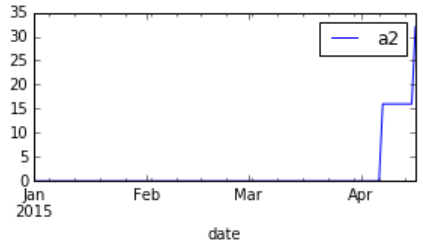

a2_ldiff
a2_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

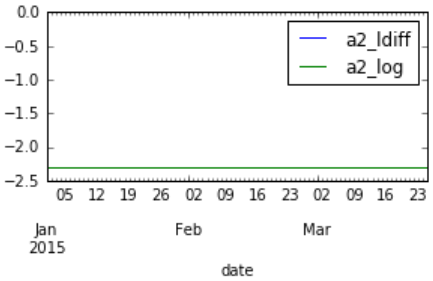

a2
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

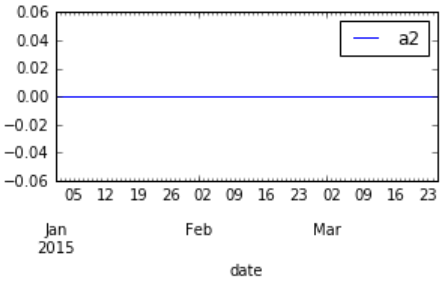

a2_ldiff
a2_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

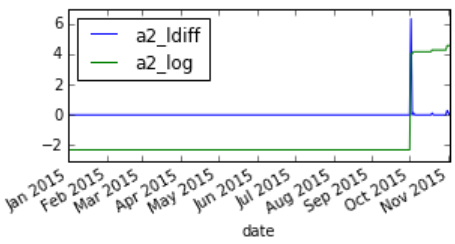

a2
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

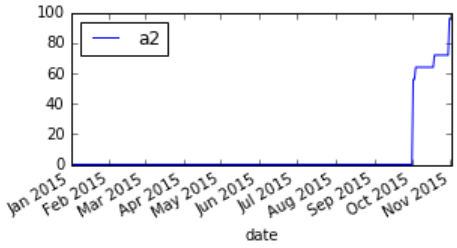

a2_ldiff
a2_log
Jan 2015
Feb
Mar
Apr
date

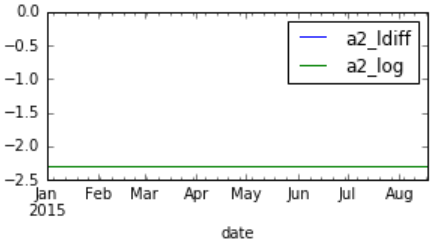

a2
Jan 2015
Feb
Mar
Apr
date

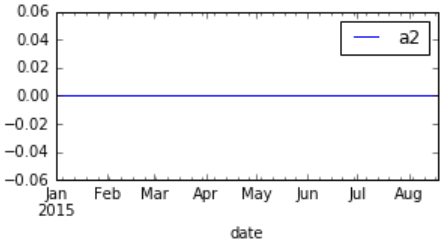

a2_ldiff
a2_log
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

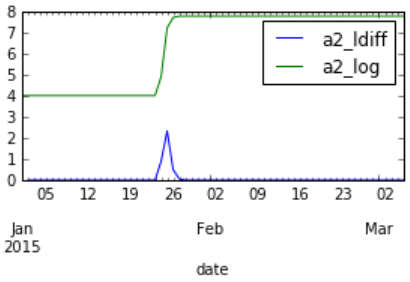

a2
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

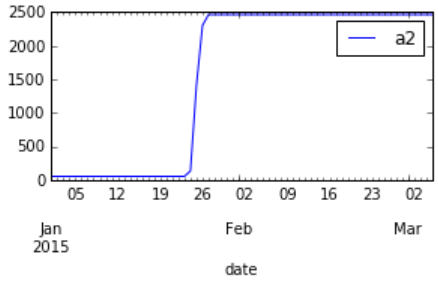

a2_ldiff
a2_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
date

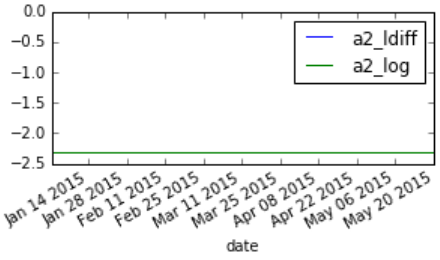

a2
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
date

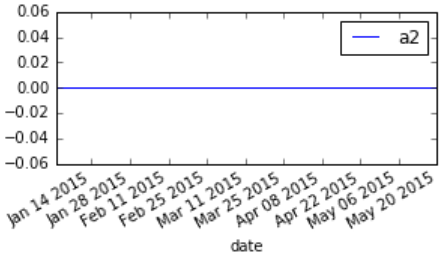

a2_ldiff
a2_log
Jan 2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

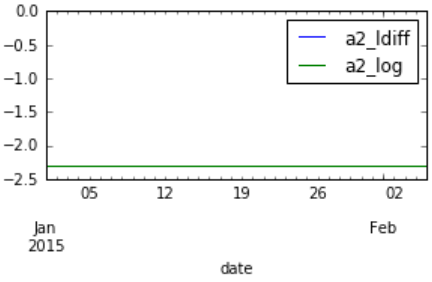

a2
Jan 2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

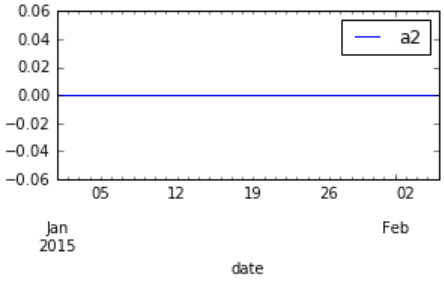

a2_ldiff
a2_log
Jan
2015
Feb
Mar
date

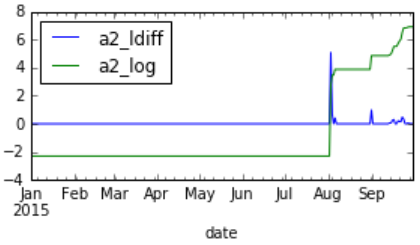

a2
Jan
2015
Feb
Mar
date

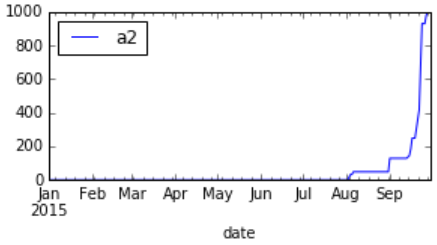

a2_ldiff
a2_log
date

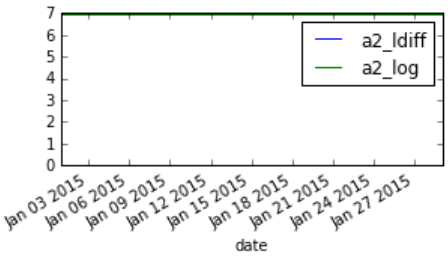

a2
date

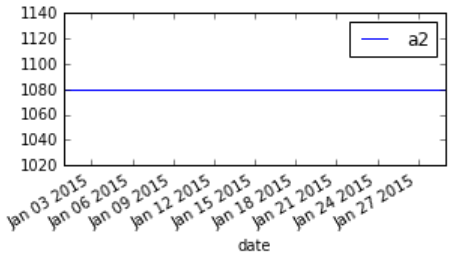

a2_ldiff
a2_log
Jan
2015
Feb
date

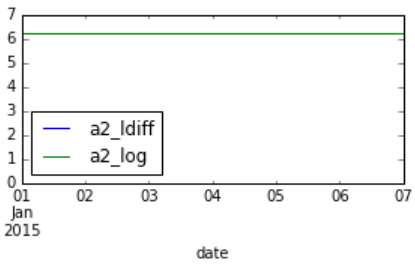

a2
Jan
2015
Feb
date

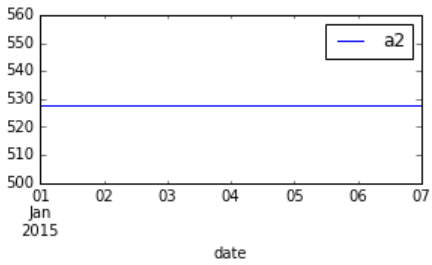

a2_ldiff
a2_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
date

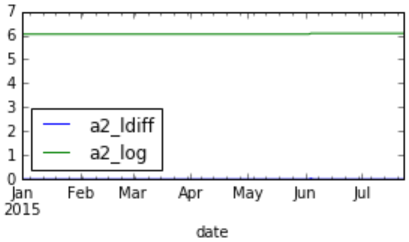
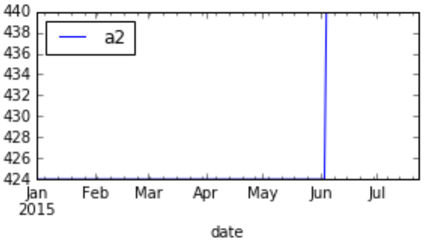
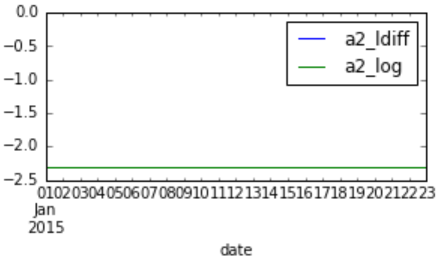
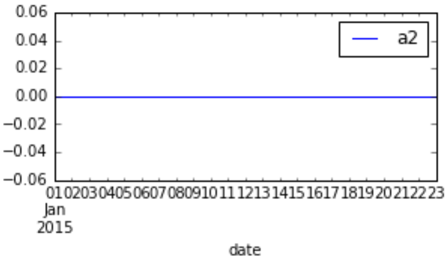
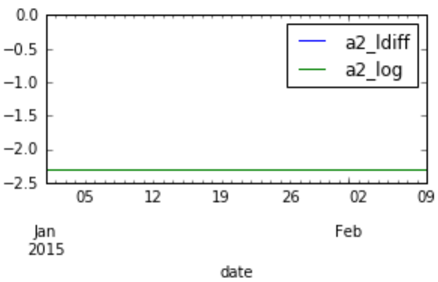
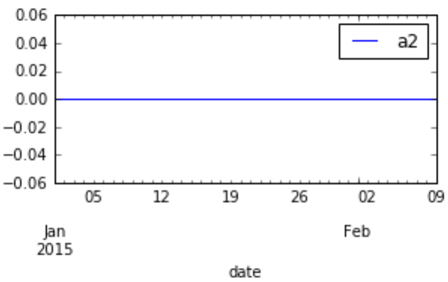
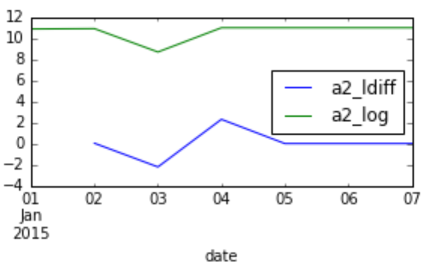
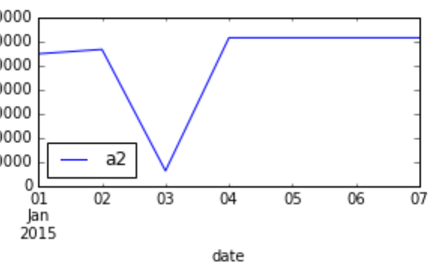
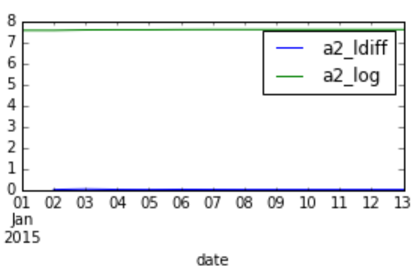
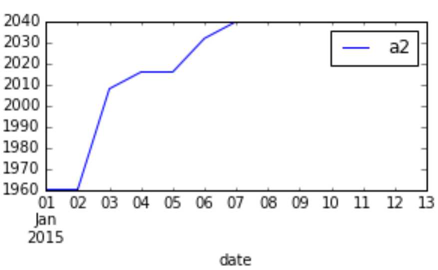
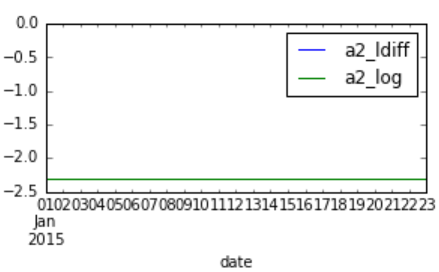
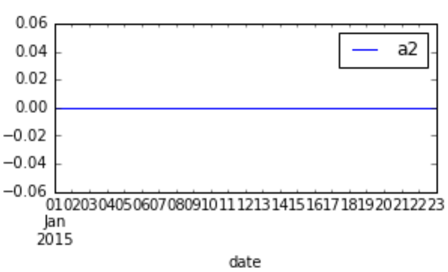

a2
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
date


a2_ldiff
a2_log
Jan 03 2015
Jan 06 2015
Jan 09 2015
Jan 12 2015
Jan 15 2015
Jan 18 2015
Jan 21 2015
Jan 24 2015
Jan 27 2015
date


a2
Jan 03 2015
Jan 06 2015
Jan 09 2015
Jan 12 2015
Jan 15 2015
Jan 18 2015
Jan 21 2015
Jan 24 2015
Jan 27 2015
date


a2_ldiff
a2_log
01
Jan
2015
02
03
04
05
06
07
date


a2
01
Jan
2015
02
03
04
05
06
07
date


a2_ldiff
a2_log
date

a2
date

a2_ldiff
a2_log
date

a2
date

a2_ldiff
a2_log
date

a2
date

a2_ldiff
a2_log
date

a2
date

a2_ldiff
a2_log
date

a2
date

a2_ldiff
a2_log
date

a2
date

Summary

This is really good. If there is signal in a2, we're capturing it

Take even False Positives seriously!

Reason recall rate is very low is because there is just no signal in a2 for these failed cases. Need to look for other features for failures

In [ ]:

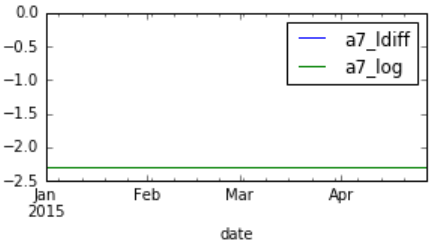
Step 4: a7 Feature Analysis

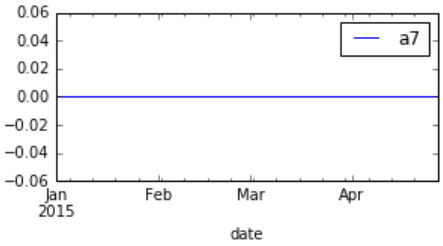
Use modules built for a2 (evolve if necessary)

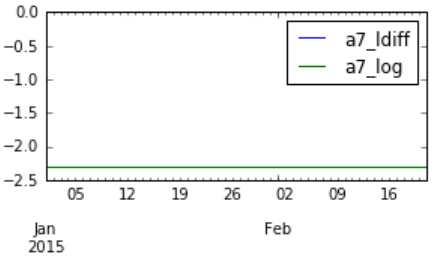
In [1]:
```
import pandas as pd
import sys
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
sys.path.append('../utils')
import DataAggregation as da
import AlgoUtils as au
cmap_bold = ListedColormap(['#00FF00','#FF0000'])
```

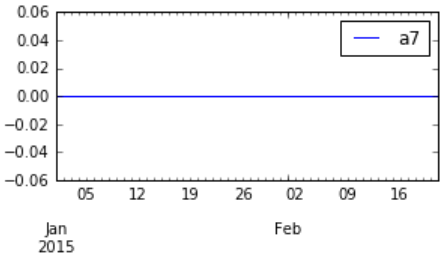
In [2]:
```
dd = da.GetFrames("../data/device_failure.csv", "a7")
```

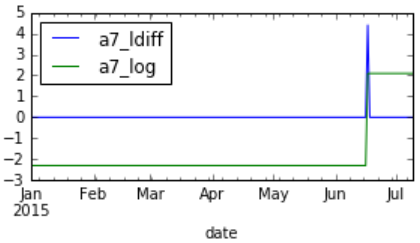
In [46]:
```
dd.plot_sample_history(dd.failed_devs["device"],10)
```

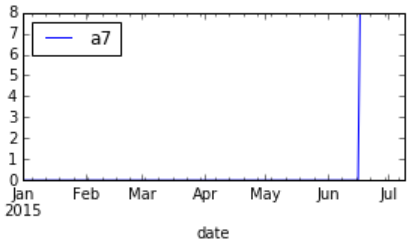

a7_ldiff
a7_log
Jan
2015
Feb
Mar
Apr
date

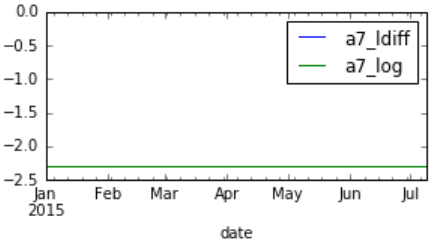

a7
Jan
2015
Feb
Mar
Apr
date

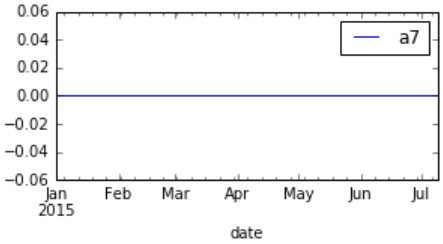

a7_ldiff
a7_log
05
12
19
26
02
09
16
Jan
2015
Feb
date

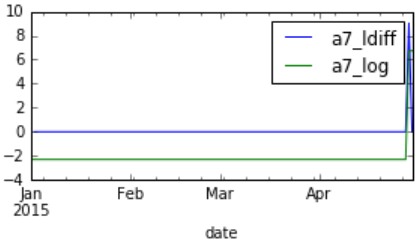

a7
05
12
19
26
02
09
16
Jan
2015
Feb
date

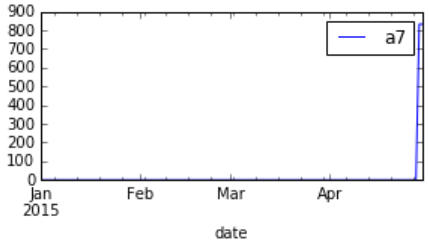

a7_ldiff
a7_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

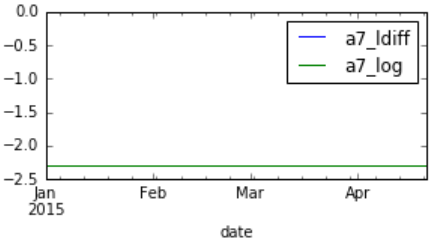

a7
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

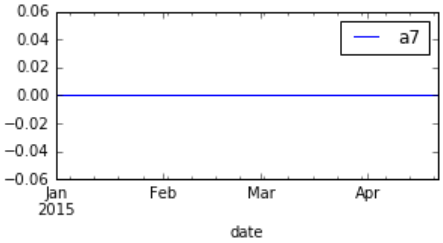

a7_ldiff
a7_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

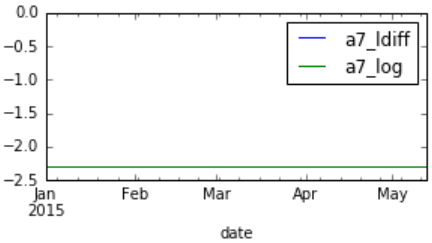

a7
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

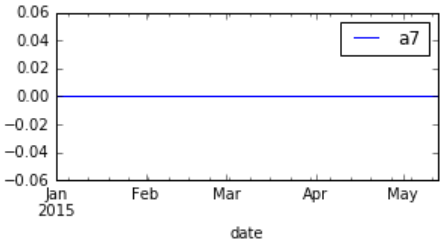

a7_ldiff
a7_log
Jan
2015
Feb
Mar
Apr
date

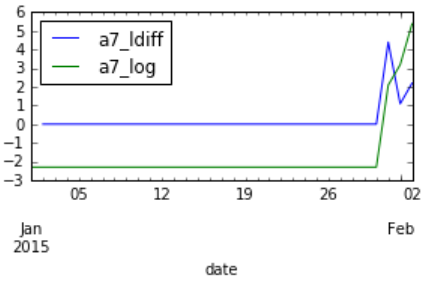

a7
Jan
2015
Feb
Mar
Apr
date

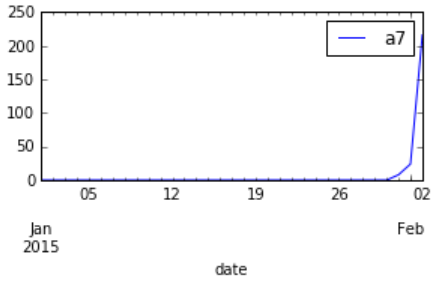

a7_ldiff
a7_log
Jan
2015
Feb
Mar
Apr
date

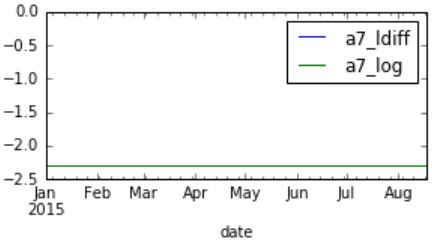

a7
Jan
2015
Feb
Mar
Apr
date

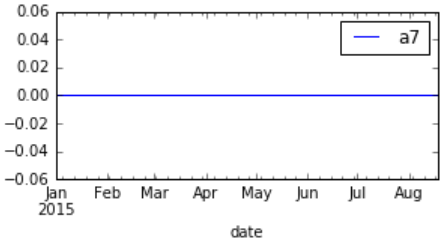

a7_ldiff
a7_log
Jan
2015
Feb
Mar
Apr
May
date

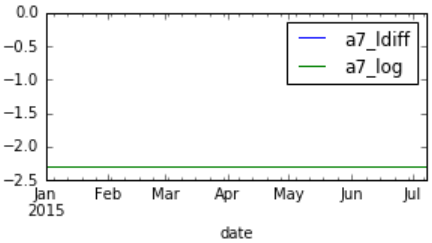

a7
Jan
2015
Feb
Mar
Apr
May
date

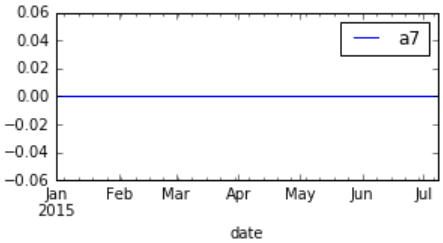

a7_ldiff
a7_log
6
5
4
3
2
1
0
−1
−2
−3
05
12
19
26
02
Jan
2015
Feb
date


a7
250
200
150
100
50
0
05
12
19
26
02
Jan
2015
Feb
date

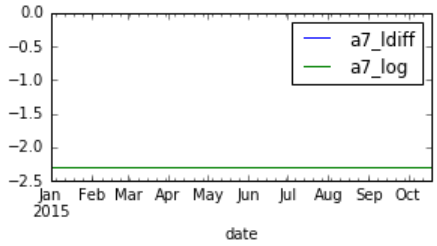

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

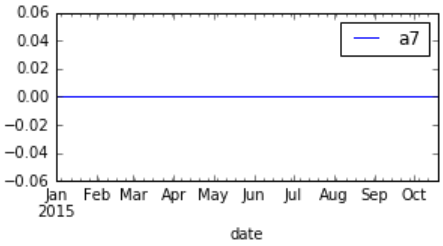

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

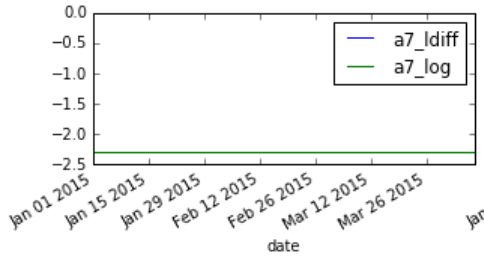

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

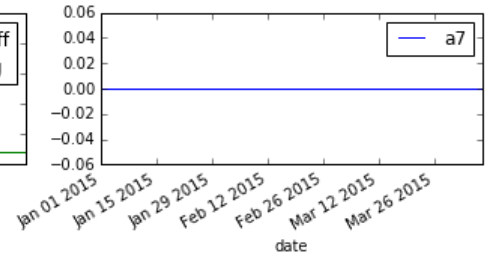

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

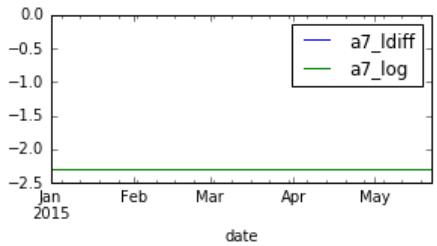
In [4]:
```
dd.plot_sample_history(dd.good_devs["device"],10)
```

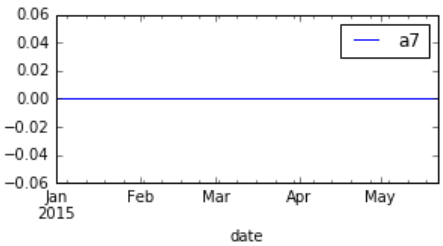

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
date

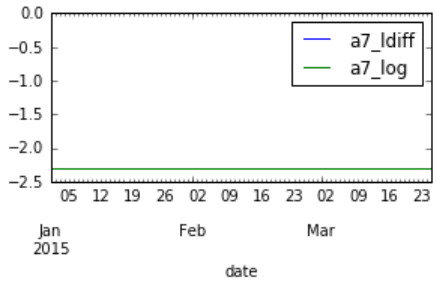

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
date

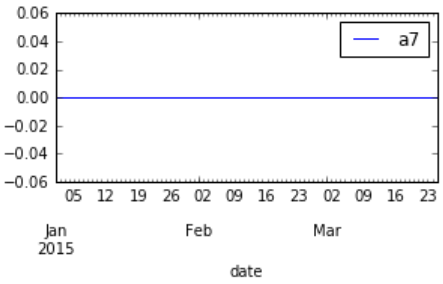

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 01 2015
Jan 15 2015
Jan 29 2015
Feb 12 2015
Feb 26 2015
Mar 12 2015
Mar 26 2015
date

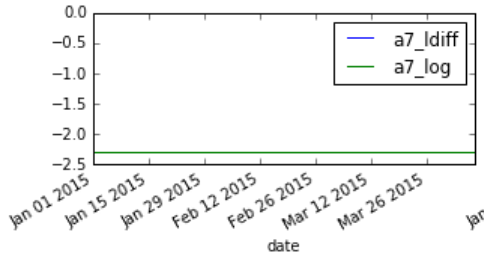

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 01 2015
Jan 15 2015
Jan 29 2015
Feb 12 2015
Feb 26 2015
Mar 12 2015
Mar 26 2015
date

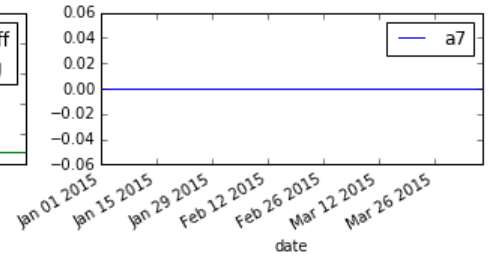

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
May
date

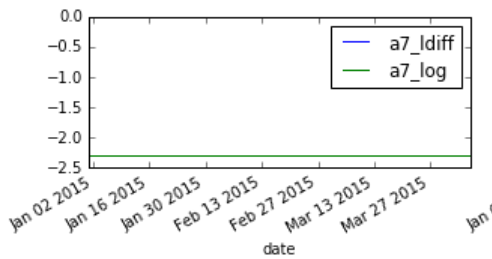

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
May
date

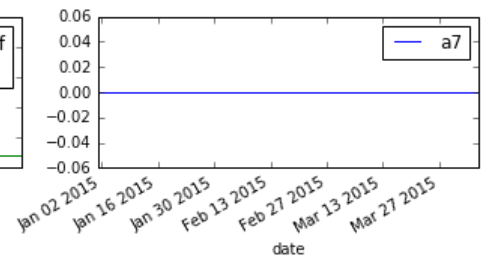

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05
12
19
26
02
09
16
23
02
09
16
23
Jan
2015
Feb
Mar
date

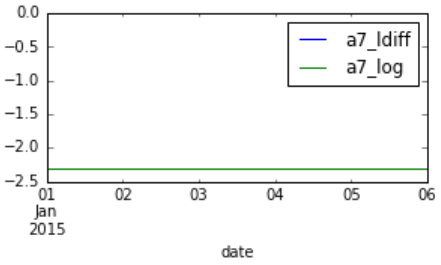

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05
12
19
26
02
09
16
23
02
09
16
23
Jan
2015
Feb
Mar
date

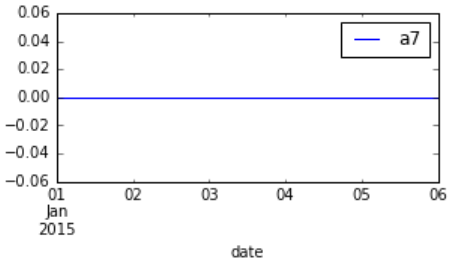

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 01 2015
Jan 15 2015
Jan 29 2015
Feb 12 2015
Feb 26 2015
Mar 12 2015
Mar 26 2015
date

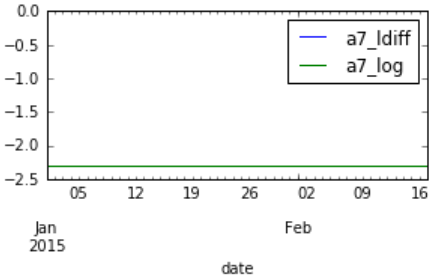

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 01 2015
Jan 15 2015
Jan 29 2015
Feb 12 2015
Feb 26 2015
Mar 12 2015
Mar 26 2015
date

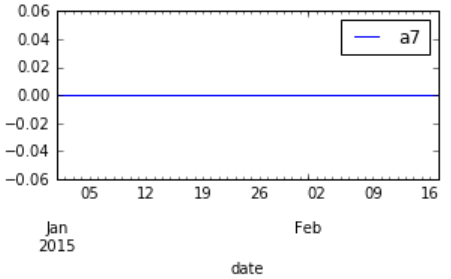

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 02 2015
Jan 16 2015
Jan 30 2015
Feb 13 2015
Feb 27 2015
Mar 13 2015
Mar 27 2015
date

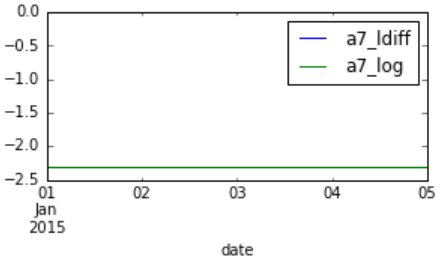

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 02 2015
Jan 16 2015
Jan 30 2015
Feb 13 2015
Feb 27 2015
Mar 13 2015
Mar 27 2015
date

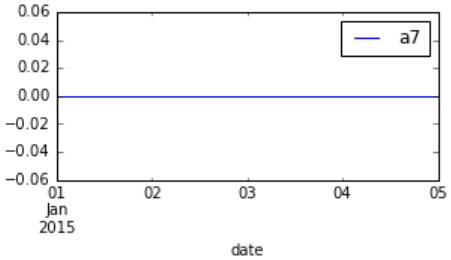

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
02
03
04
05
06
Jan
2015
date

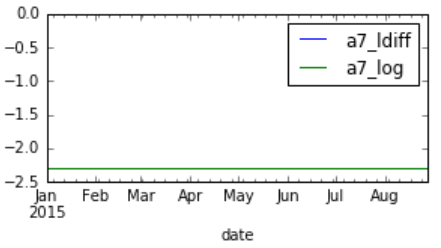

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
02
03
04
05
06
Jan
2015
date

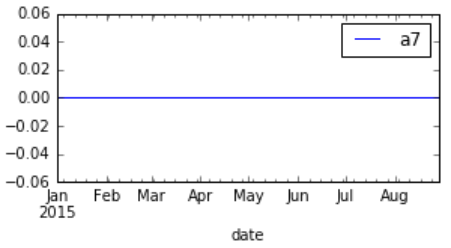

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05
12
19
26
02
09
16
Jan
2015
Feb
date


a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05
12
19
26
02
09
16
Jan
2015
Feb
date


a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
02
03
04
05
Jan
2015
date

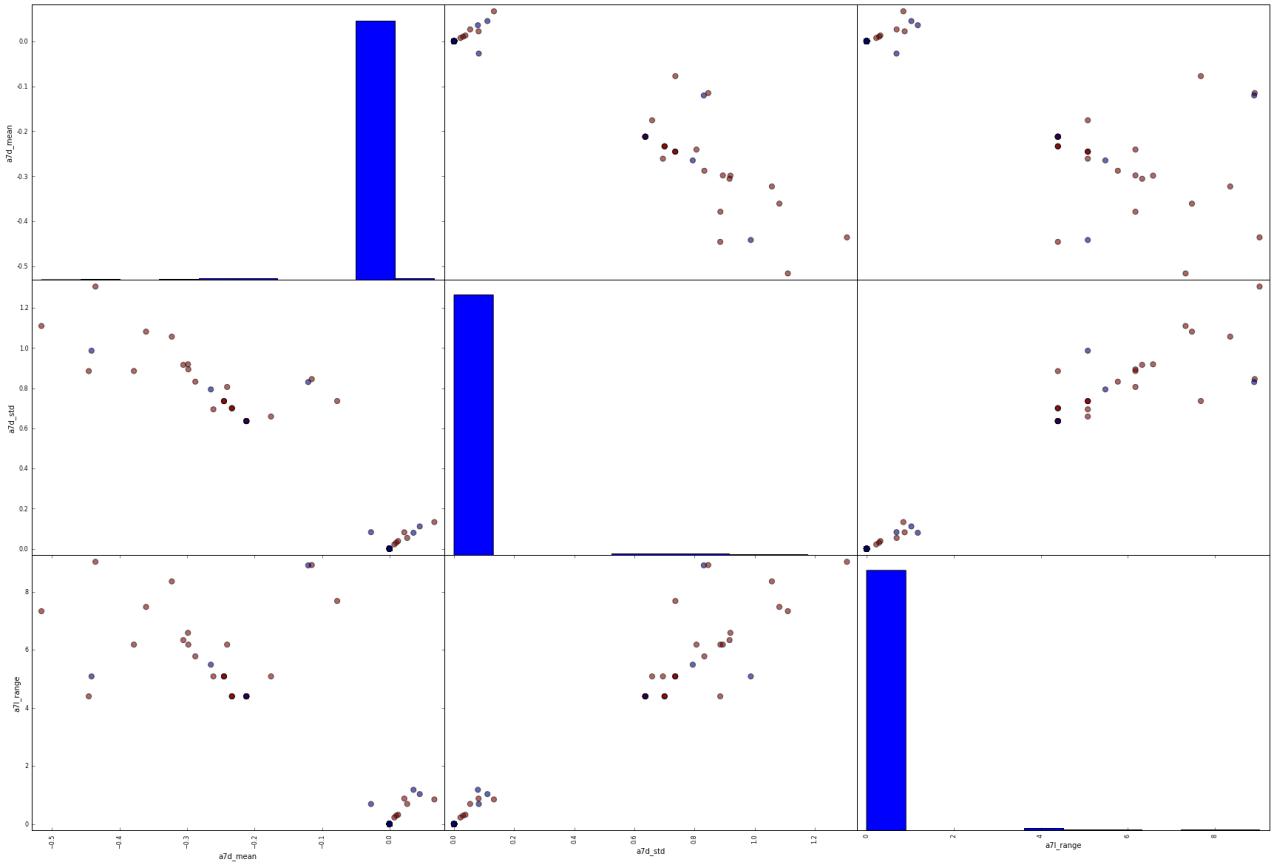

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
02
03
04
05
Jan
2015
date


a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date


a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

In [5]:
```
sfeature = dd.sfeature
fcols_tmp = [sfeature +"d_mean", sfeature +"d_std", sfeature +"l_range"]
df_sfeature = dd.df_sfeature
pd.scatter_matrix(df_sfeature[fcols_tmp], figsize=(30,20), s=200, c=df_sfeature["failure"], alpha=0.6)
```

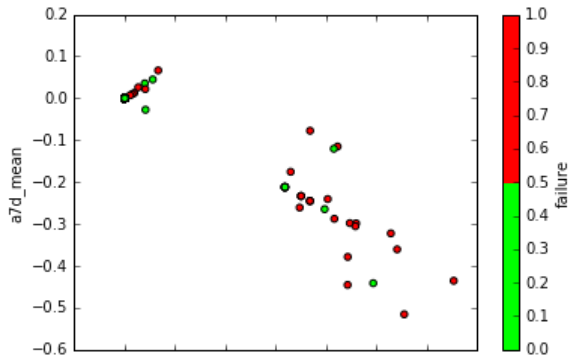
Out[5]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f2499076f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f2498fc9c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f2498f4bc90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f2498def890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f2498d2fd50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f2498d08590>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f24989966d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f249871a590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f2495746650>]], dtype=object)
```

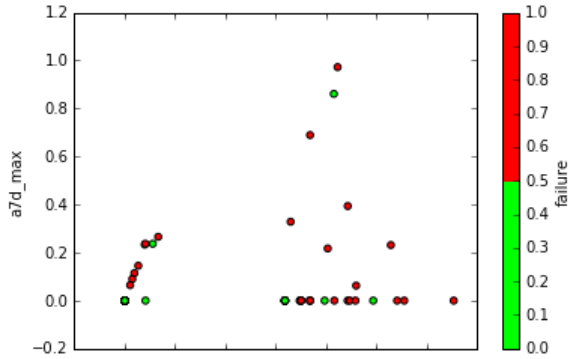


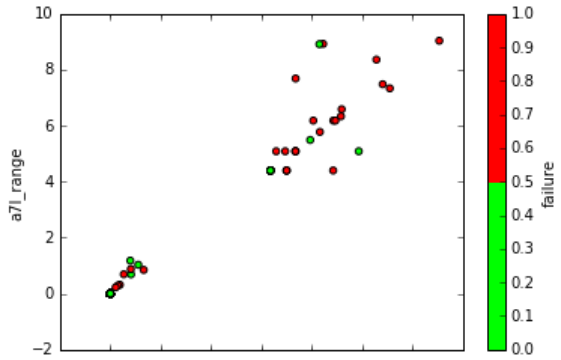
Decent. See seperation between good and bad devices.

In [13]:
```
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_mean", c="failure", colormap=cmap_bold)
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_max", c="failure", colormap=cmap_bold)
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"l_range", c="failure", colormap=cmap_bold)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2498797e50>







Do see seperation in good Vs bad

Unsure which algorithm will work best at this point

In [14]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [15]:
```
df_sfeature = dd.df_sfeature
sfeature = dd.sfeature
df_results = au.run_algo_analysis(df_sfeature, sfeature, fcols, algos_str, algos_dd)
```

```
----------
LogisticRegression:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.24
Cross-val-score(precision)= 0.81
Cross-val-score(f1)       = 0.81
----------
LogisticRegressionB:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.31
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
KNeighborsClassifier:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.29
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
LinearDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.79
Cross-val-score(f1)       = 0.79
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.32
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
LogisticRegression:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.81
Cross-val-score(f1)       = 0.81
----------
LogisticRegressionB:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.32
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
KNeighborsClassifier:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.29
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
LinearDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.79
Cross-val-score(f1)       = 0.79
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.32
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
LogisticRegression:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
```

```
Cross-val-score(f1)       = 0.82
----------
LogisticRegressionB:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.31
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
KNeighborsClassifier:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.29
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
LinearDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
QuadraticDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.79
Cross-val-score(f1)       = 0.79
----------
LogisticRegression:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
LogisticRegressionB:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.31
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
KNeighborsClassifier:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.29
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
LinearDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
QuadraticDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
LogisticRegression:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
LogisticRegressionB:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.29
```

```
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
KNeighborsClassifier:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.30
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
LinearDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
QuadraticDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.66
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.26
Cross-val-score(precision)= 0.82
Cross-val-score(f1)       = 0.82
----------
LogisticRegression:d_max
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.05
Cross-val-score(precision)= 0.28
Cross-val-score(f1)       = 0.28
----------
LogisticRegressionB:d_max
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.12
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
----------
KNeighborsClassifier:d_max
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.10
Cross-val-score(precision)= 0.70
Cross-val-score(f1)       = 0.70
----------
LinearDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.07
Cross-val-score(precision)= 0.58
Cross-val-score(f1)       = 0.58
----------
QuadraticDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.10
Cross-val-score(precision)= 0.70
Cross-val-score(f1)       = 0.70
```

In [16]:
```
df_results
```

Out[16]:

	fcols	algo	recall	precision	f1	roc_auc	accuracy
4	d_mean:d_std:d_max:l_range	QuadraticDiscriminantAnalysis	0.32	0.82	0.44	0.66	0.93
6	d_mean:d_std:l_range	LogisticRegressionB	0.32	0.82	0.44	0.66	0.93
9	d_mean:d_std:l_range	QuadraticDiscriminantAnalysis	0.32	0.82	0.44	0.66	0.93
1	d_mean:d_std:d_max:l_range	LogisticRegressionB	0.31	0.82	0.43	0.66	0.93
11	d_std:l_range	LogisticRegressionB	0.31	0.80	0.43	0.66	0.93
16	l_range	LogisticRegressionB	0.31	0.80	0.43	0.66	0.93
22	d_std	KNeighborsClassifier	0.30	0.80	0.42	0.66	0.93
2	d_mean:d_std:d_max:l_range	KNeighborsClassifier	0.29	0.80	0.41	0.66	0.93
7	d_mean:d_std:l_range	KNeighborsClassifier	0.29	0.80	0.41	0.66	0.93
12	d_std:l_range	KNeighborsClassifier	0.29	0.80	0.41	0.66	0.93
17	l_range	KNeighborsClassifier	0.29	0.80	0.41	0.66	0.93
21	d_std	LogisticRegressionB	0.29	0.80	0.41	0.66	0.93
3	d_mean:d_std:d_max:l_range	LinearDiscriminantAnalysis	0.26	0.79	0.38	0.66	0.93
5	d_mean:d_std:l_range	LogisticRegression	0.26	0.81	0.38	0.66	0.93
8	d_mean:d_std:l_range	LinearDiscriminantAnalysis	0.26	0.79	0.38	0.66	0.93
10	d_std:l_range	LogisticRegression	0.26	0.82	0.38	0.66	0.93
13	d_std:l_range	LinearDiscriminantAnalysis	0.26	0.82	0.38	0.66	0.93
15	l_range	LogisticRegression	0.26	0.82	0.38	0.66	0.93
18	l_range	LinearDiscriminantAnalysis	0.26	0.82	0.38	0.66	0.93
19	l_range	QuadraticDiscriminantAnalysis	0.26	0.82	0.38	0.66	0.93
20	d_std	LogisticRegression	0.26	0.82	0.38	0.66	0.93
23	d_std	LinearDiscriminantAnalysis	0.26	0.82	0.38	0.66	0.93
24	d_std	QuadraticDiscriminantAnalysis	0.26	0.82	0.38	0.66	0.93
14	d_std:l_range	QuadraticDiscriminantAnalysis	0.26	0.79	0.37	0.66	0.93
0	d_mean:d_std:d_max:l_range	LogisticRegression	0.24	0.81	0.36	0.65	0.93
26	d_max	LogisticRegressionB	0.12	0.80	0.21	0.56	0.92
27	d_max	KNeighborsClassifier	0.10	0.70	0.17	0.56	0.92
29	d_max	QuadraticDiscriminantAnalysis	0.10	0.70	0.17	0.56	0.92
28	d_max	LinearDiscriminantAnalysis	0.07	0.58	0.13	0.56	0.91
25	d_max	LogisticRegression	0.05	0.28	0.08	0.56	0.91

Results look bit better than expected!

QuadraticDiscriminantAnalysis(QDA) again produces fairly precise results.

Lets see why Recall(Sensitivity) is so low

In [19]:
```
sfeature = dd.sfeature
algo_str = "QuadraticDiscriminantAnalysis"
fcols = [sfeature + x for x in "d_mean:d_std:d_max:l_range".split(":")]
```

In [20]:
```
analysisdf = au.do_clf_validate(df_sfeature, algo_str,algos_dd[algo_str], fcols, "failure")
```

```
Accuracy = 0.96
Confusion Matrix
[[270   1]
 [ 12   9]]
recall_sensitivity = 0.43
precision          = 0.90
f1                 = 0.58
```

In [23]:
```
mispredictdf.head()
```

Out[23]:

	index	a7d_mean	a7d_std	a7d_max	a7l_range	failure	y_pred
0	Z1F1VQFY	0	0	0	0	1	0
1	W1F0Z3KR	0	0	0	0	1	0
2	S1F0S4T6	0	0	0	0	1	0
3	Z1F0MRPJ	0	0	0	0	1	0
4	Z1F130LH	0	0	0	0	1	0

In [41]:
```
mispredictdf = analysisdf[analysisdf["failure"] != analysisdf["y_pred"]]
mispredictdf.reset_index(inplace=True)
mispredictdf.columns = ["device"] + list(mispredictdf.columns[1:])
```

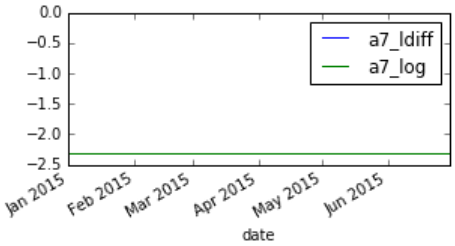
In [42]:
```
mispredictdf
```

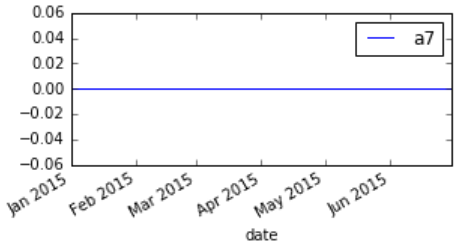
Out[42]:

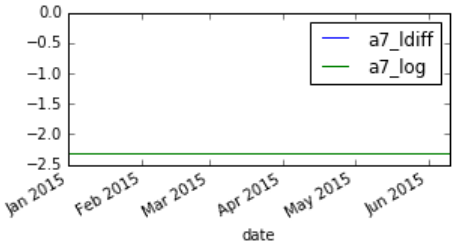
	device	a7d_mean	a7d_std	a7d_max	a7l_range	failure	y_pred
0	Z1F1VQFY	0.000000	0.000000	0.000000	0.000000	1	0
1	W1F0Z3KR	0.000000	0.000000	0.000000	0.000000	1	0
2	S1F0S4T6	0.000000	0.000000	0.000000	0.000000	1	0
3	Z1F0MRPJ	0.000000	0.000000	0.000000	0.000000	1	0
4	Z1F130LH	0.000000	0.000000	0.000000	0.000000	1	0
5	S1F0PJJW	0.000000	0.000000	0.000000	0.000000	1	0
6	Z1F0P5D9	0.000000	0.000000	0.000000	0.000000	1	0
7	W1F1B0KF	-0.120476	0.830599	0.860881	8.903951	0	1
8	S1F0JD7P	0.000000	0.000000	0.000000	0.000000	1	0
9	S1F10E6M	0.000000	0.000000	0.000000	0.000000	1	0
10	S1F0CTDN	0.000000	0.000000	0.000000	0.000000	1	0
11	S1F0T2LA	0.000000	0.000000	0.000000	0.000000	1	0
12	W1F0P114	0.000000	0.000000	0.000000	0.000000	1	0

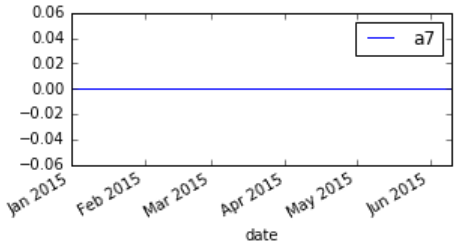
In [43]:
```
mispredict_devs = pd.DataFrame(mispredictdf.device.unique())
mispredict_devs.columns = ["device"]
```

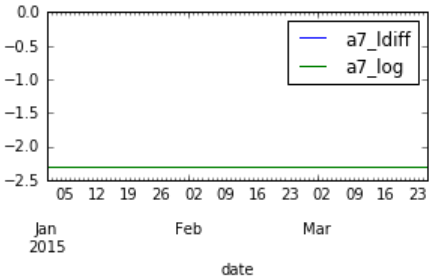
In [44]:
```
dd.plot_sample_history(mispredict_devs["device"],0)
```

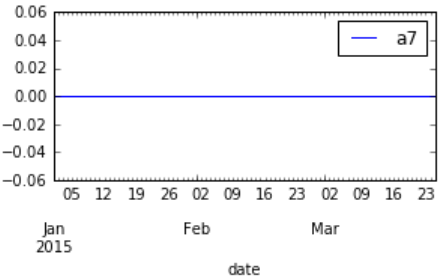

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

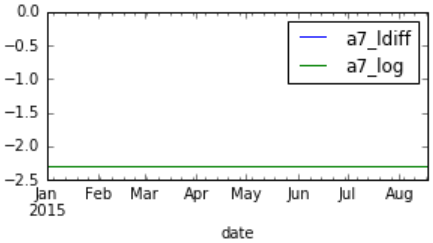

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

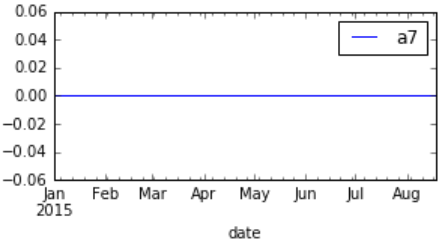

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

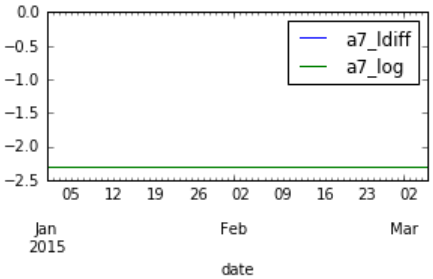

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

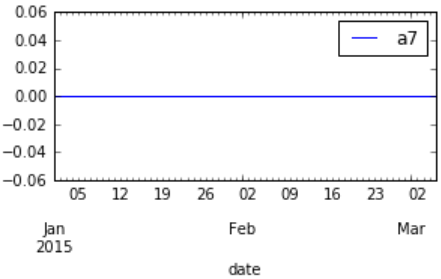

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

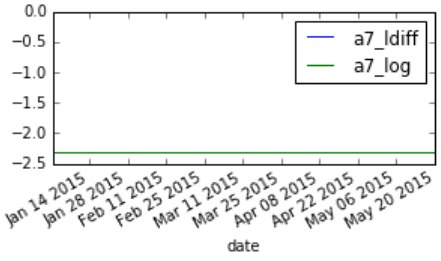

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

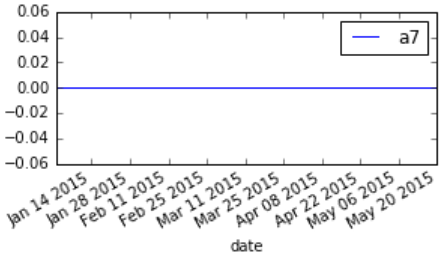

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

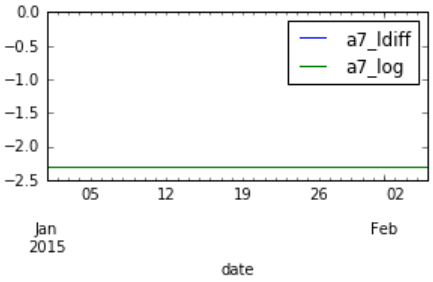

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

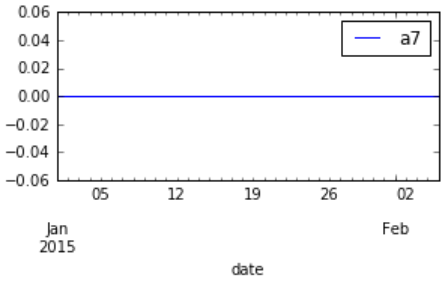

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05 12 19 26 02 09 16 23 02
Jan 2015
Feb
Mar
date

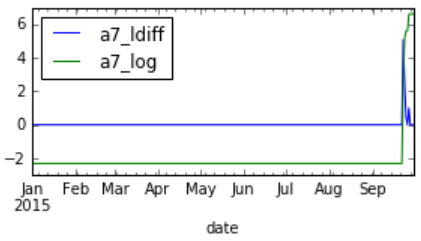

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05 12 19 26 02 09 16 23 02
Jan 2015
Feb
Mar
date

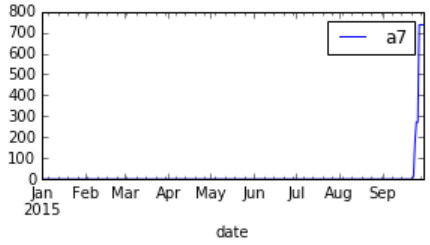

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 14 2015
Jan 28 2015
Feb 11 2015
Feb 25 2015
Mar 11 2015
Mar 25 2015
Apr 08 2015
Apr 22 2015
May 06 2015
May 20 2015
date

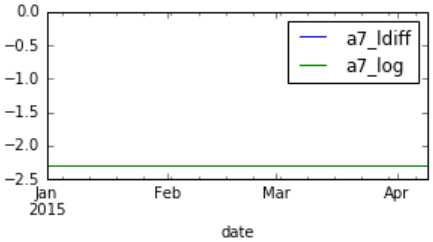

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 14 2015
Jan 28 2015
Feb 11 2015
Feb 25 2015
Mar 11 2015
Mar 25 2015
Apr 08 2015
Apr 22 2015
May 06 2015
May 20 2015
date

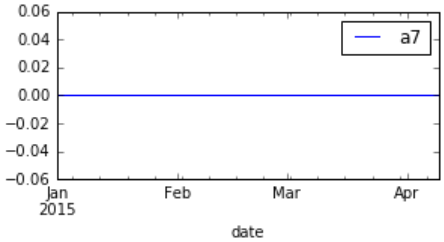

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
05
12
19
26
02
Jan
2015
Feb
date

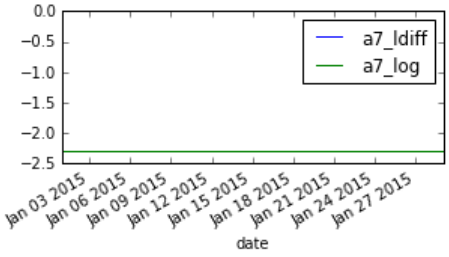

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
05
12
19
26
02
Jan
2015
Feb
date

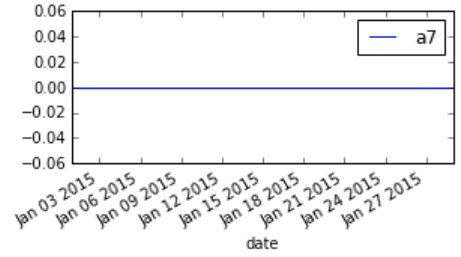

a7_ldiff
a7_log
6
4
2
0
−2
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
date

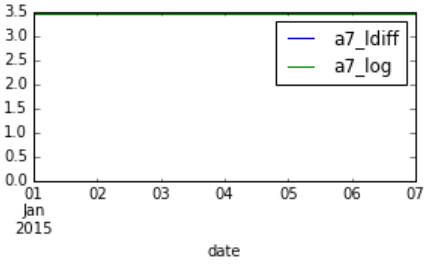

a7
800
700
600
500
400
300
200
100
0
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
date

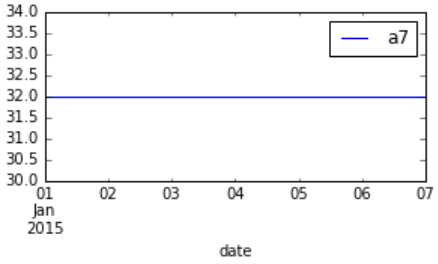

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
date

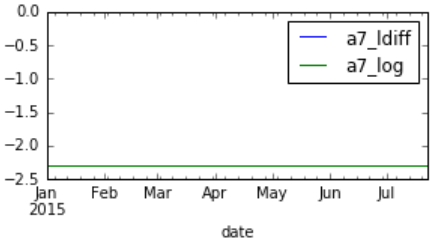

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
date

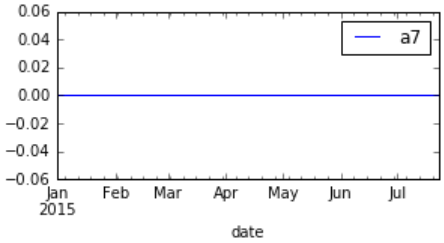

a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan 03 2015
Jan 06 2015
Jan 09 2015
Jan 12 2015
Jan 15 2015
Jan 18 2015
Jan 21 2015
Jan 24 2015
Jan 27 2015
date

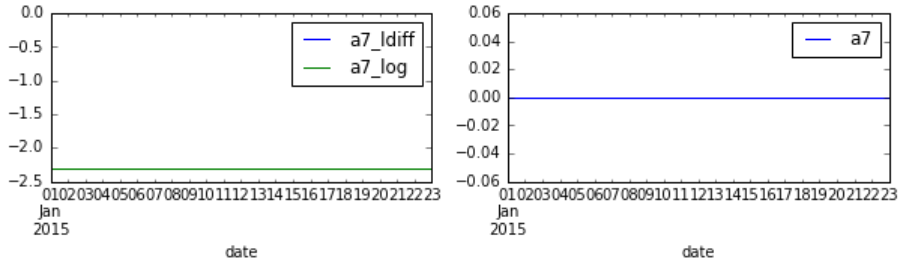

a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan 03 2015
Jan 06 2015
Jan 09 2015
Jan 12 2015
Jan 15 2015
Jan 18 2015
Jan 21 2015
Jan 24 2015
Jan 27 2015
date


a7_ldiff
a7_log
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
01
02
03
04
05
06
07
Jan
2015
date


a7
34.0
33.5
33.0
32.5
32.0
31.5
31.0
30.5
30.0
01
02
03
04
05
06
07
Jan
2015
date


a7_ldiff
a7_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date


a7
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date



Summary

Results for a7 look even better than a2! Better precision.

Take even False Positives seriously!

Reason recall rate is very low is because there is just no signal in a7 for these failed cases. Need to look for other features for failures

In [ ]:

Step 5: a4 Feature Analysis

Use modules built for a2 and a7 (evolve if necessary)

In [1]:
```
import pandas as pd
import sys
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
sys.path.append('../utils')
import DataAggregation as da
import AlgoUtils as au
cmap_bold = ListedColormap(['#00FF00','#FF0000'])
```

In [2]:
```
dd = da.GetFrames("../data/device_failure.csv", "a4", ldays=-30, lday_strict=False)
```

Features ldays, lday_strict were added after I went through analysis few times.

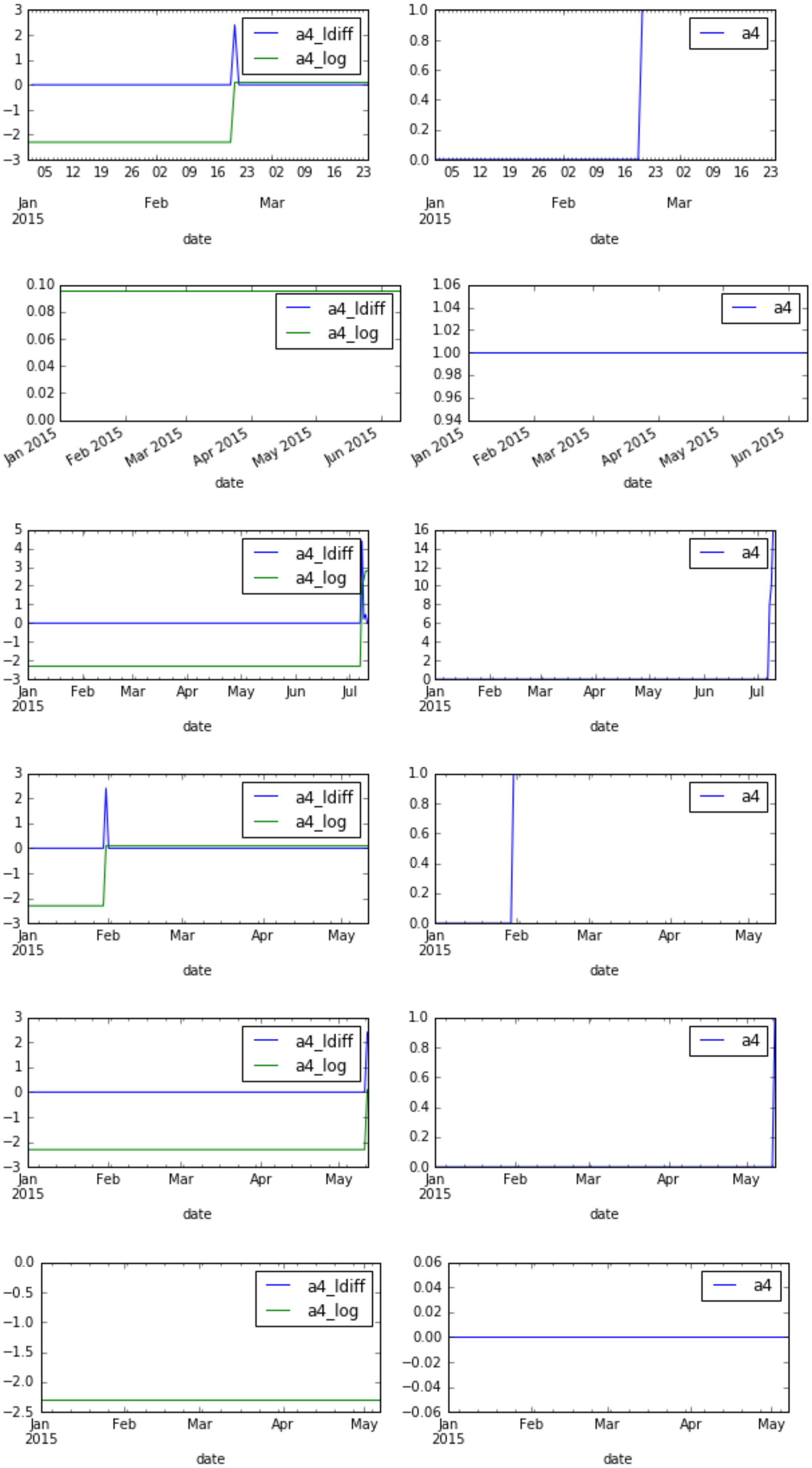
Unlike a2 and a7, Failures with a4 are not that dramatic.

For a2, a7: Getting log & log differential metrics for last 10 days was sufficient.

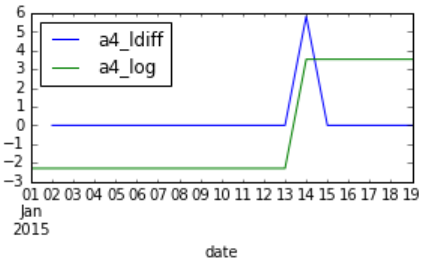
ldays = -30: For a4, increasing N (last N days) to 30 is giving better results

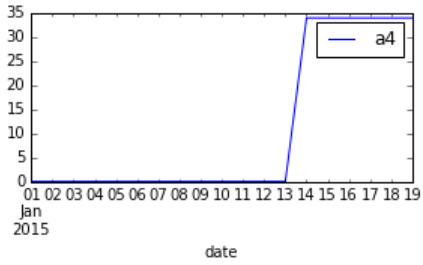
lday_strict=False: Look at log range for entire year

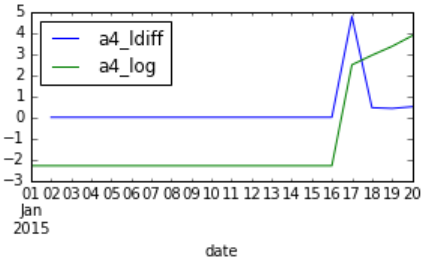
In [5]:
```
dd.plot_sample_history(dd.failed_devs["device"],10)
```

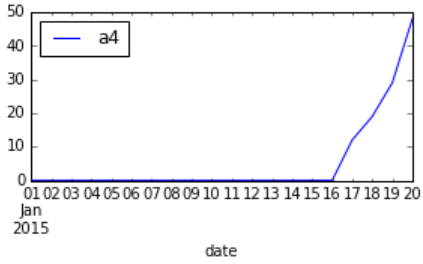


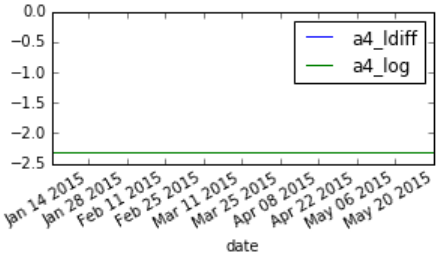
a4_ldiff
a4_log
a4
date
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015

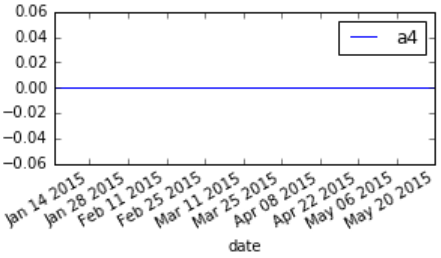

a4_ldiff
a4_log
date

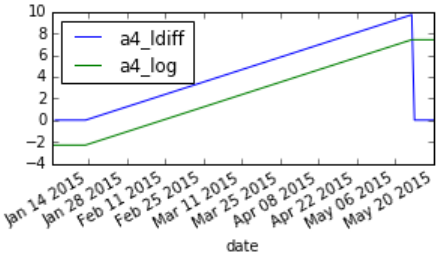

a4
date

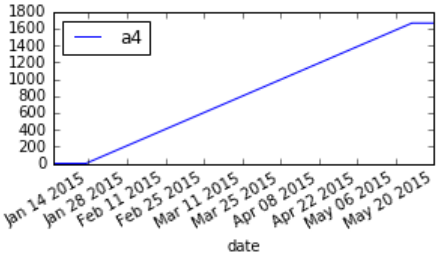

a4_ldiff
a4_log
date


a4
date

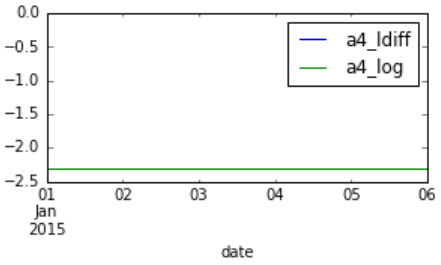

a4_ldiff
a4_log
date

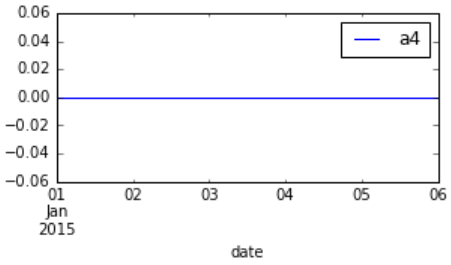

a4
date

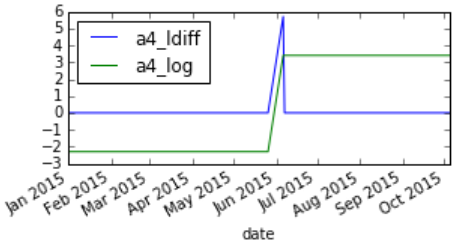

a4_ldiff
a4_log
date

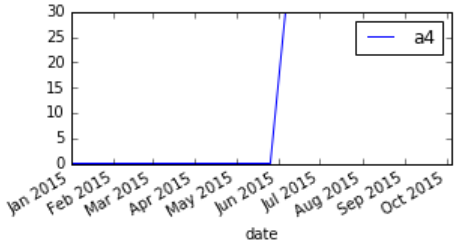

a4
date

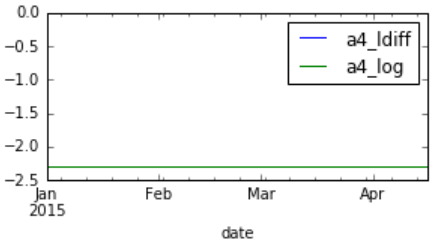
In [4]:
```
dd.plot_sample_history(dd.good_devs["device"],10)
```

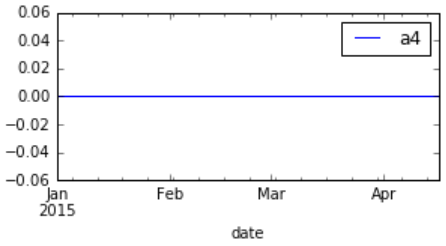

a4_ldiff
a4_log
01
Jan
2015
02
03
04
05
06
date

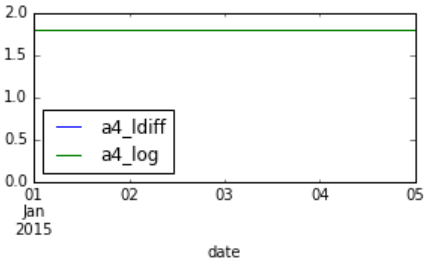

a4
01
Jan
2015
02
03
04
05
06
date

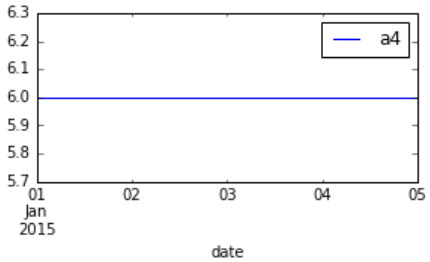

a4_ldiff
a4_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
date

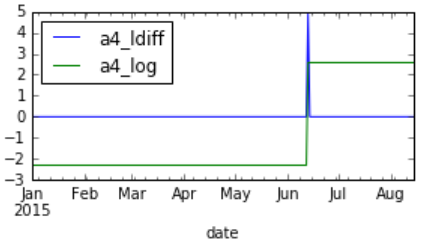

a4
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
date

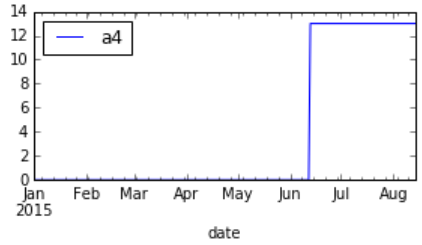

a4_ldiff
a4_log
Jan
2015
Feb
Mar
Apr
date

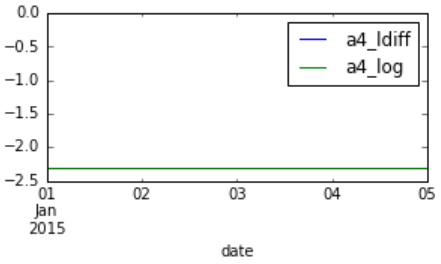

a4
Jan
2015
Feb
Mar
Apr
date

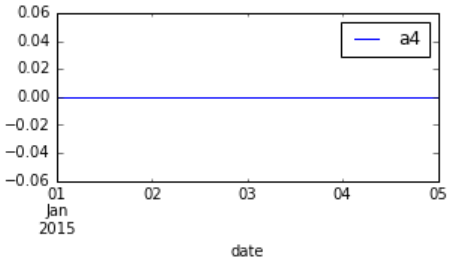

a4_ldiff
a4_log
01
Jan
2015
02
03
04
05
date

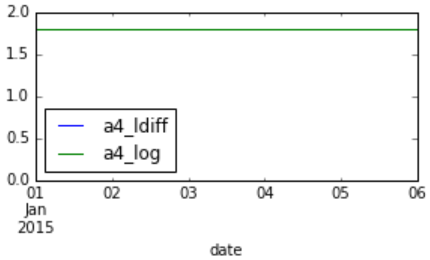

a4
01
Jan
2015
02
03
04
05
date

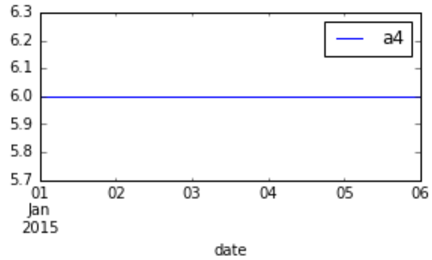

a4_ldiff
a4_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

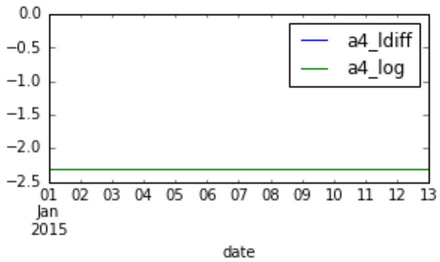

a4
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

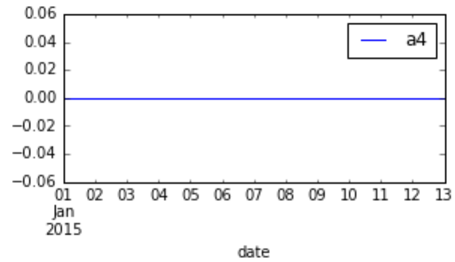

a4_ldiff
a4_log
01
Jan
2015
02
03
04
05
date

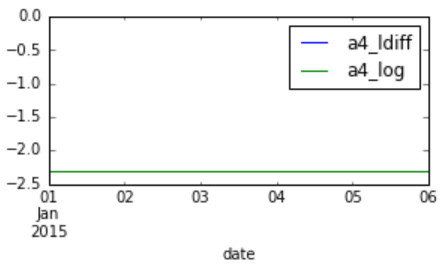

a4
01
Jan
2015
02
03
04
05
date

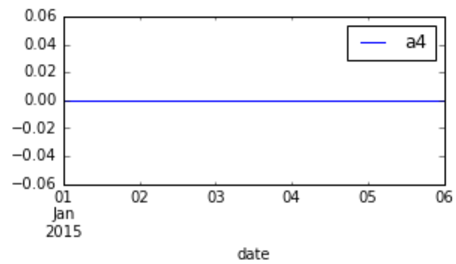

a4_ldiff
a4_log
2.0
1.5
1.0
0.5
0.0
01
Jan
2015
02
03
04
05
06
date

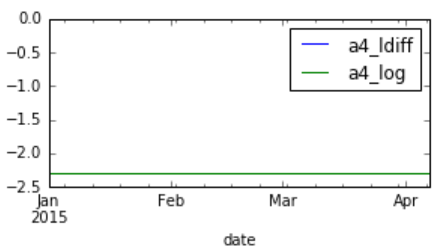

a4
6.3
6.2
6.1
6.0
5.9
5.8
5.7
01
Jan
2015
02
03
04
05
06
date

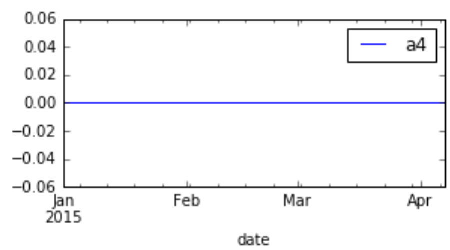

a4_ldiff
a4_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
Jan
2015
02
03
04
05
06
07
08
09
10
11
12
13
date


a4
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
Jan
2015
02
03
04
05
06
07
08
09
10
11
12
13
date


a4_ldiff
a4_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
01
Jan
2015
02
03
04
05
06
date

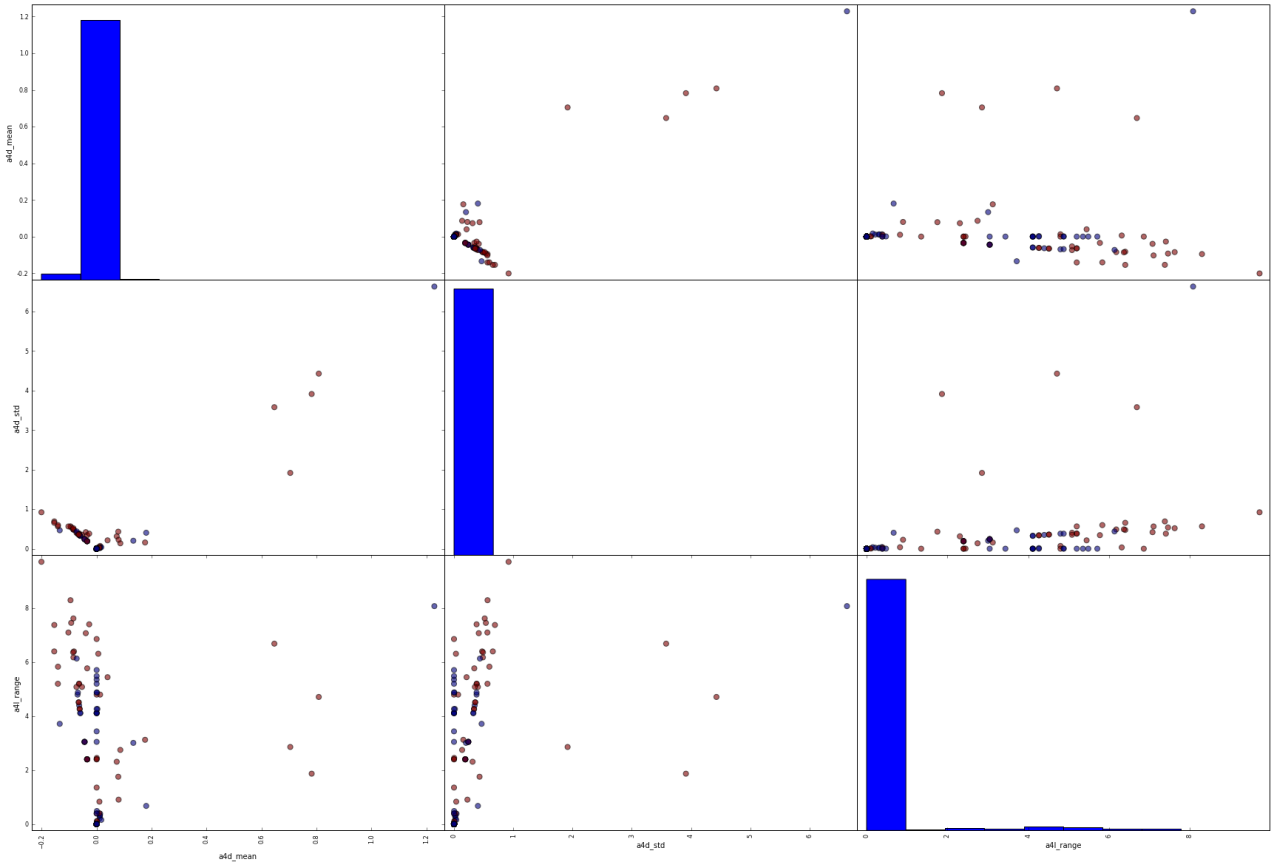

a4
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
01
Jan
2015
02
03
04
05
06
date


a4_ldiff
a4_log
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Jan
2015
Feb
Mar
Apr
date


a4
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Jan
2015
Feb
Mar
Apr
date

In [6]:
```
sfeature = dd.sfeature
fcols_tmp = [sfeature +"d_mean", sfeature +"d_std", sfeature +"l_range"]
df_sfeature = dd.df_sfeature
pd.scatter_matrix(df_sfeature[fcols_tmp], figsize=(30,20), s=200, c=df_sfeature["failure"], alpha=0.6)
```

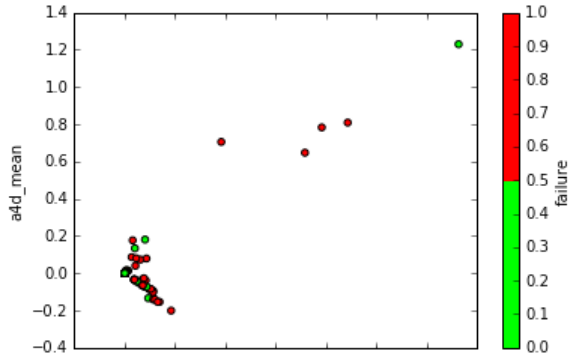
Out[6]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc81ee8e0d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc81e605fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc81e358090>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc81b52cc50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc81b13a150>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc81b086950>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc81b014a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc81af96950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc81af04a10>]], dtype=object)
```

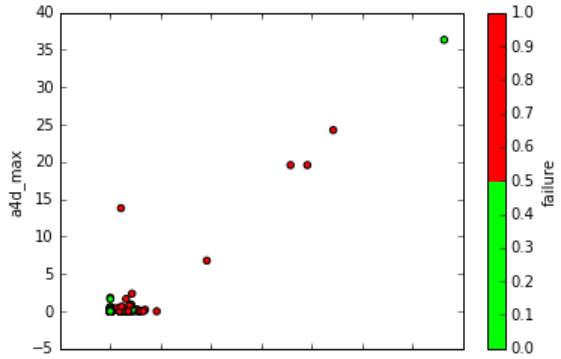


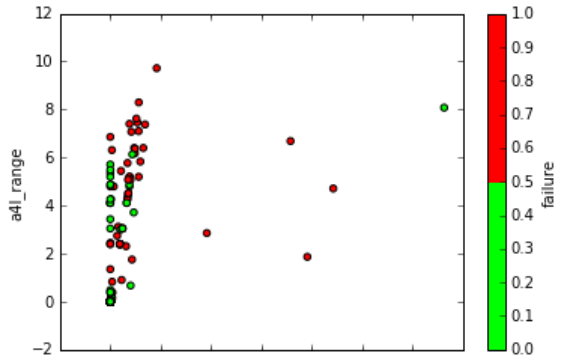
Decent! See seperation between good and bad devices.

In [7]:
```
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_mean", c="failure", colormap=cmap_bold)
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_max", c="failure", colormap=cmap_bold)
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"l_range", c="failure", colormap=cmap_bold)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc81f181ed0>







Last graph above a4d_std Vs a4l_lrange is interesting

Note the lack of good devices that have a4l_range between 1 and 3

Almost feel like there are 2 classes of devices: TBD

In [9]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [10]:
```
df_sfeature = dd.df_sfeature
sfeature = dd.sfeature
df_results = au.run_algo_analysis(df_sfeature, sfeature, fcols, algos_str, algos_dd)
```

```
/home/vagrant/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Undefine
dMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/vagrant/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Undefine
dMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
----------
LogisticRegression:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.25
Cross-val-score(precision)= 0.57
Cross-val-score(f1)       = 0.57
----------
LogisticRegressionB:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.44
Cross-val-score(precision)= 0.61
Cross-val-score(f1)       = 0.61
----------
KNeighborsClassifier:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.34
Cross-val-score(precision)= 0.70
Cross-val-score(f1)       = 0.70
----------
LinearDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.33
Cross-val-score(precision)= 0.53
Cross-val-score(f1)       = 0.53
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.71
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.40
Cross-val-score(precision)= 0.59
Cross-val-score(f1)       = 0.59
----------
LogisticRegression:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.27
Cross-val-score(precision)= 0.62
Cross-val-score(f1)       = 0.62
----------
LogisticRegressionB:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.46
Cross-val-score(precision)= 0.61
Cross-val-score(f1)       = 0.61
----------
KNeighborsClassifier:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.74
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.37
Cross-val-score(precision)= 0.73
Cross-val-score(f1)       = 0.73
----------
LinearDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.34
Cross-val-score(precision)= 0.53
Cross-val-score(f1)       = 0.53
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.43
Cross-val-score(precision)= 0.58
Cross-val-score(f1)       = 0.58
----------
LogisticRegression:d_std:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.25
Cross-val-score(precision)= 0.59
```

```
Cross-val-score(f1)       = 0.59
----------
LogisticRegressionB:d_std:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.46
Cross-val-score(precision)= 0.61
Cross-val-score(f1)       = 0.61
----------
KNeighborsClassifier:d_std:l_range
Cross-val-score(roc_auc) = 0.74
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.36
Cross-val-score(precision)= 0.70
Cross-val-score(f1)       = 0.70
----------
LinearDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.34
Cross-val-score(precision)= 0.53
Cross-val-score(f1)       = 0.53
----------
QuadraticDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.42
Cross-val-score(precision)= 0.59
Cross-val-score(f1)       = 0.59
----------
LogisticRegression:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.28
Cross-val-score(precision)= 0.57
Cross-val-score(f1)       = 0.57
----------
LogisticRegressionB:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.46
Cross-val-score(precision)= 0.61
Cross-val-score(f1)       = 0.61
----------
KNeighborsClassifier:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.36
Cross-val-score(precision)= 0.67
Cross-val-score(f1)       = 0.67
----------
LinearDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.35
Cross-val-score(precision)= 0.58
Cross-val-score(f1)       = 0.58
----------
QuadraticDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.73
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.41
Cross-val-score(precision)= 0.59
Cross-val-score(f1)       = 0.59
----------
LogisticRegression:d_std
Cross-val-score(roc_auc) = 0.71
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.08
Cross-val-score(precision)= 0.50
Cross-val-score(f1)       = 0.50
----------
LogisticRegressionB:d_std
Cross-val-score(roc_auc) = 0.71
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.38
```

```
Cross-val-score(precision)= 0.70
Cross-val-score(f1)       = 0.70
----------
KNeighborsClassifier:d_std
Cross-val-score(roc_auc) = 0.70
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.37
Cross-val-score(precision)= 0.65
Cross-val-score(f1)       = 0.65
----------
LinearDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.71
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.04
Cross-val-score(precision)= 0.20
Cross-val-score(f1)       = 0.20
----------
QuadraticDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.71
Cross-val-score(accuracy) = 0.92
Cross-val-score(recall)   = 0.08
Cross-val-score(precision)= 0.55
Cross-val-score(f1)       = 0.55
----------
LogisticRegression:d_max
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:d_max
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.27
Cross-val-score(precision)= 0.73
Cross-val-score(f1)       = 0.73
----------
KNeighborsClassifier:d_max
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.28
Cross-val-score(precision)= 0.73
Cross-val-score(f1)       = 0.73
----------
LinearDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.03
Cross-val-score(precision)= 0.10
Cross-val-score(f1)       = 0.10
----------
QuadraticDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.61
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.03
Cross-val-score(precision)= 0.15
Cross-val-score(f1)       = 0.15
```

In [11]:
```
df_results
```

Out[11]:

	fcols	algo	recall	precision	f1	roc_auc	accuracy
6	d_mean:d_std:l_range	LogisticRegressionB	0.46	0.61	0.50	0.73	0.93
11	d_std:l_range	LogisticRegressionB	0.46	0.61	0.50	0.73	0.93
16	l_range	LogisticRegressionB	0.46	0.61	0.50	0.73	0.93
1	d_mean:d_std:d_max:l_range	LogisticRegressionB	0.44	0.61	0.49	0.72	0.93
9	d_mean:d_std:l_range	QuadraticDiscriminantAnalysis	0.43	0.58	0.47	0.73	0.93
14	d_std:l_range	QuadraticDiscriminantAnalysis	0.42	0.59	0.47	0.73	0.93
19	l_range	QuadraticDiscriminantAnalysis	0.41	0.59	0.46	0.73	0.92
4	d_mean:d_std:d_max:l_range	QuadraticDiscriminantAnalysis	0.40	0.59	0.46	0.71	0.92
21	d_std	LogisticRegressionB	0.38	0.70	0.47	0.71	0.93
7	d_mean:d_std:l_range	KNeighborsClassifier	0.37	0.73	0.48	0.74	0.93
22	d_std	KNeighborsClassifier	0.37	0.65	0.44	0.70	0.93
12	d_std:l_range	KNeighborsClassifier	0.36	0.70	0.46	0.74	0.93
17	l_range	KNeighborsClassifier	0.36	0.67	0.45	0.73	0.93
18	l_range	LinearDiscriminantAnalysis	0.35	0.58	0.42	0.73	0.92
2	d_mean:d_std:d_max:l_range	KNeighborsClassifier	0.34	0.70	0.44	0.72	0.93
8	d_mean:d_std:l_range	LinearDiscriminantAnalysis	0.34	0.53	0.40	0.72	0.92
13	d_std:l_range	LinearDiscriminantAnalysis	0.34	0.53	0.40	0.72	0.92
3	d_mean:d_std:d_max:l_range	LinearDiscriminantAnalysis	0.33	0.53	0.40	0.72	0.92
27	d_max	KNeighborsClassifier	0.28	0.73	0.39	0.65	0.93
15	l_range	LogisticRegression	0.28	0.57	0.36	0.73	0.92
26	d_max	LogisticRegressionB	0.27	0.73	0.38	0.65	0.93
5	d_mean:d_std:l_range	LogisticRegression	0.27	0.62	0.36	0.72	0.92
0	d_mean:d_std:d_max:l_range	LogisticRegression	0.25	0.57	0.34	0.72	0.92
10	d_std:l_range	LogisticRegression	0.25	0.59	0.34	0.72	0.92
20	d_std	LogisticRegression	0.08	0.50	0.14	0.71	0.92
24	d_std	QuadraticDiscriminantAnalysis	0.08	0.55	0.14	0.71	0.92
23	d_std	LinearDiscriminantAnalysis	0.04	0.20	0.06	0.71	0.91
28	d_max	LinearDiscriminantAnalysis	0.03	0.10	0.04	0.65	0.91
29	d_max	QuadraticDiscriminantAnalysis	0.03	0.15	0.04	0.61	0.91
25	d_max	LogisticRegression	0.00	0.00	0.00	0.65	0.91

Good thing is a4 is significantly increasing recall (sensitivity)

Precision did go down a bit, but it's not that bad

Probably indicate that time to failure is a bit large once we detect failure

This could turn out real good if we keep getting data from field to validate

Understanding Misclassification

In [14]:
```
sfeature = dd.sfeature
algo_str = "LogisticRegressionB"
fcols = [sfeature + x for x in "d_mean:d_std:l_range".split(":")]
```

In [16]:
```
analysisdf = au.do_clf_validate(df_sfeature, algo_str,algos_dd[algo_str], fcols, "failure")
```

```
Accuracy = 0.93
Confusion Matrix
[[265   6]
 [ 14   7]]
recall_sensitivity = 0.33
precision          = 0.54
f1                 = 0.41
```

In [18]:
```
mispredictdf = analysisdf[analysisdf["failure"] != analysisdf["y_pred"]]
mispredictdf.reset_index(inplace=True)
mispredictdf.columns = ["device"] + list(mispredictdf.columns[1:])
```

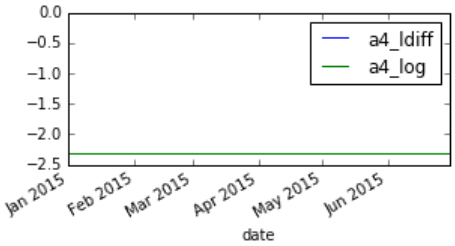
In [19]:
```
mispredictdf
```

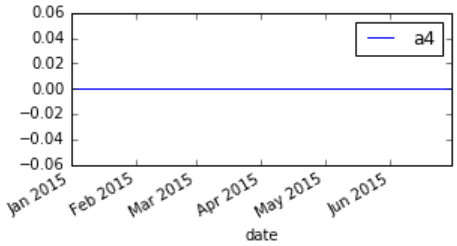
Out[19]:

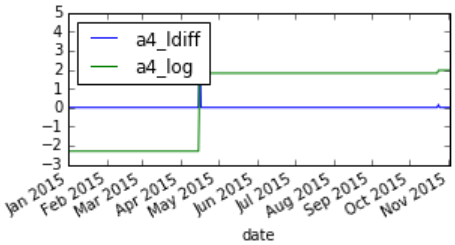
	device	a4d_mean	a4d_std	a4l_range	failure	y_pred
0	Z1F1VQFY	0.000000	0.000000	0.000000	1	0
1	S1F0GCED	0.002798	0.015327	4.262680	0	1
2	W1F0Z3KR	0.000000	0.000000	0.000000	1	0
3	S1F0S4T6	0.000000	0.000000	0.000000	1	0
4	S1F0S7B7	0.000000	0.000000	5.198497	0	1
5	Z1F0MRPJ	0.000000	0.000000	0.000000	1	0
6	Z1F130LH	0.000000	0.000000	0.000000	1	0
7	S1F0PJJW	0.000000	0.000000	0.000000	1	0
8	Z1F0P5D9	0.000000	0.000000	0.000000	1	0
9	W1F1B0KF	-0.072681	0.437537	6.133398	0	1
10	Z1F0LM6T	0.000000	0.000000	4.262680	0	1
11	S1F0BVK1	-0.059511	0.325955	4.110874	0	1
12	S1F10E6M	0.000000	0.000000	0.000000	1	0
13	S1F0CTDN	0.000000	0.000000	0.000000	1	0
14	S1F0T2LA	0.000000	0.000000	0.000000	1	0
15	W1F0X4FC	0.000000	0.000000	0.000000	1	0
16	Z1F04GCH	0.000000	0.000000	0.000000	1	0
17	S1F0S5CE	0.000000	0.000000	4.110874	0	1
18	W1F13SRV	0.000000	0.000000	0.000000	1	0
19	W1F0P114	0.009401	0.024611	0.399986	1	0

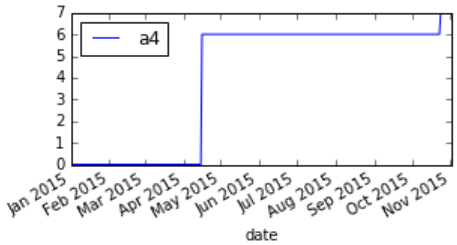
In [20]:
```
mispredict_devs = pd.DataFrame(mispredictdf.device.unique())
mispredict_devs.columns = ["device"]
```

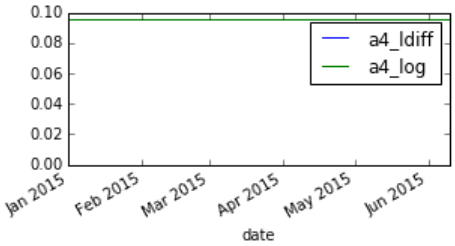
In [21]:
```
dd.plot_sample_history(mispredict_devs["device"],0)
```

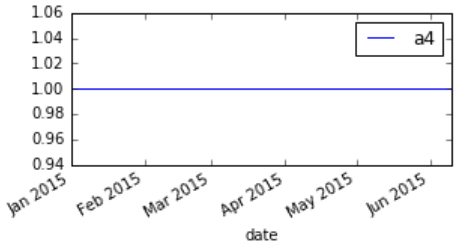

a4_ldiff
a4_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

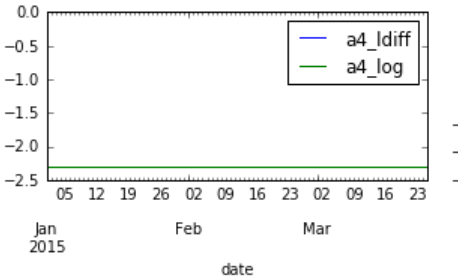

a4
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

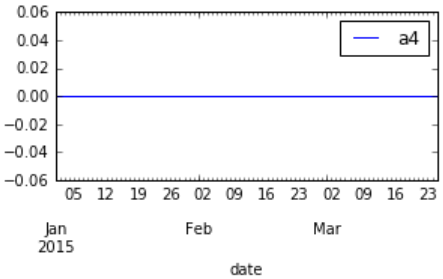

a4_ldiff
a4_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
date

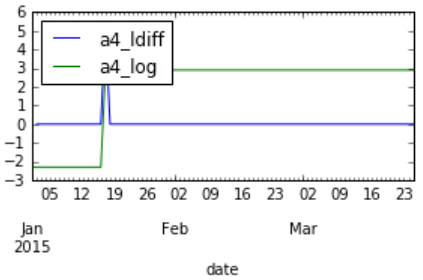

a4
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
date

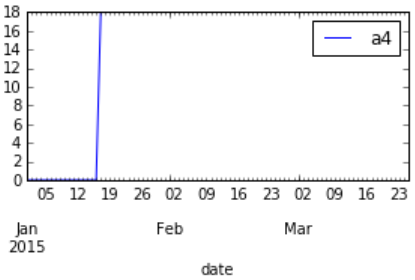

a4_ldiff
a4_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

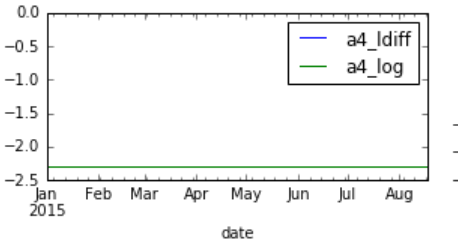

a4
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
date

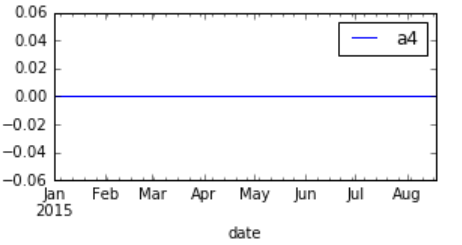

a4_ldiff
a4_log
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

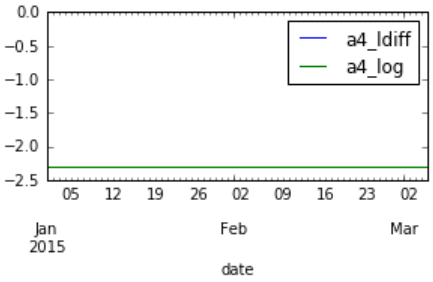

a4
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

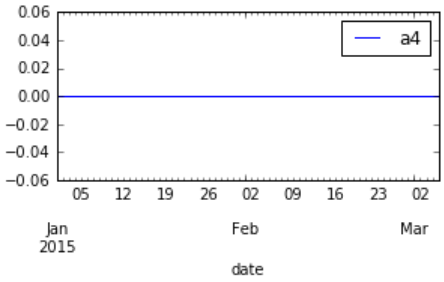

a4_ldiff
a4_log
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

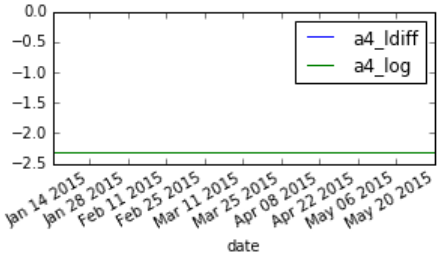

a4
05 12 19 26 02 09 16 23 02 09 16 23
Jan 2015
Feb
Mar
date

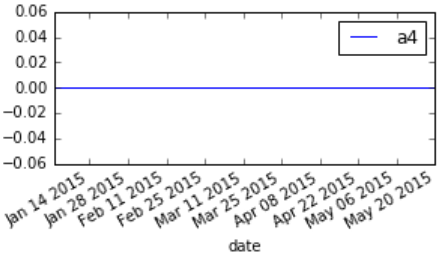

a4_ldiff
a4_log
Jan 2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

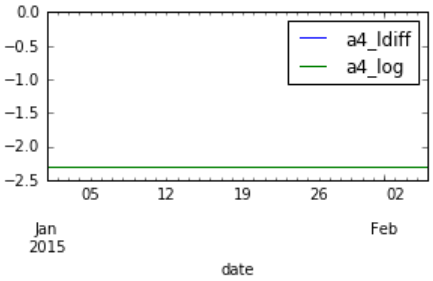

a4
Jan 2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

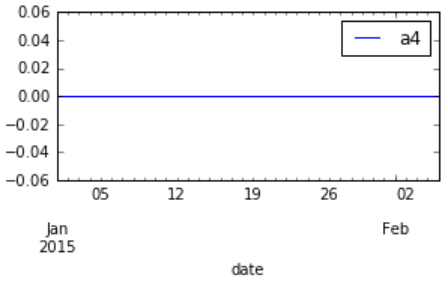

a4_ldiff
a4_log
Jan
2015
Feb
Mar
date

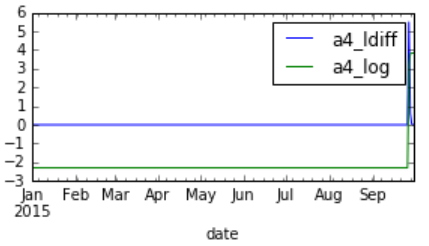

a4
Jan
2015
Feb
Mar
date

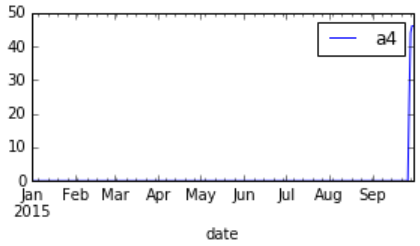

a4_ldiff
a4_log
Jan 14 2015
Jan 28 2015
Feb 11 2015
Feb 25 2015
Mar 11 2015
Mar 25 2015
Apr 08 2015
Apr 22 2015
May 06 2015
May 20 2015
date

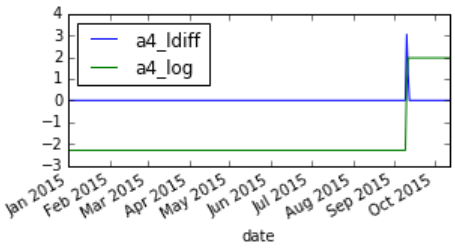

a4
Jan 14 2015
Jan 28 2015
Feb 11 2015
Feb 25 2015
Mar 11 2015
Mar 25 2015
Apr 08 2015
Apr 22 2015
May 06 2015
May 20 2015
date

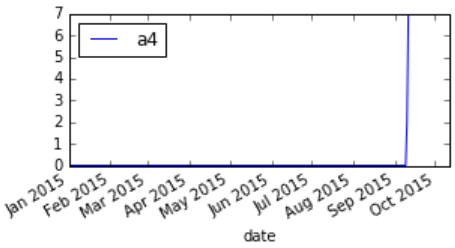

a4_ldiff
a4_log
Jan
2015
Feb
date

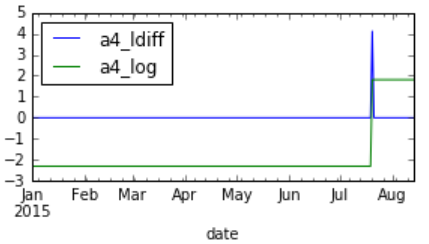

a4
Jan
2015
Feb
date

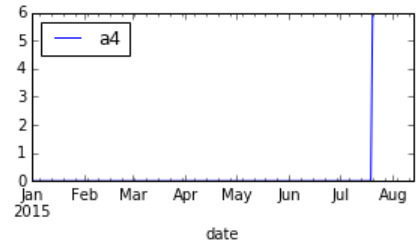

a4_ldiff
a4_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
date

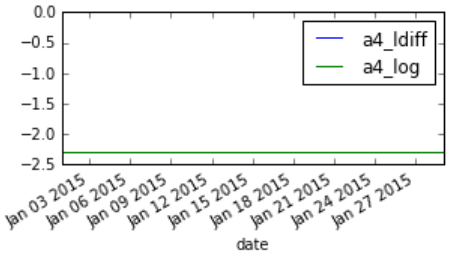

a4
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
date

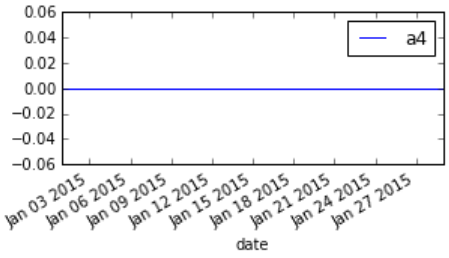

a4_ldiff
a4_log
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
date

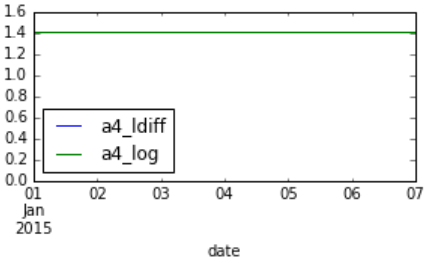

a4
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
date

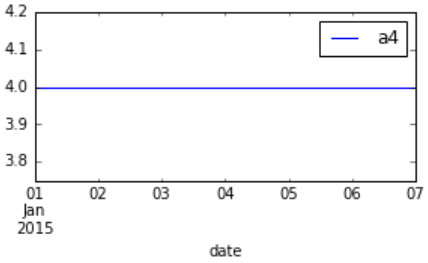

a4_ldiff
a4_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

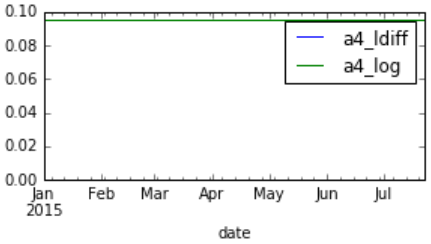

a4
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

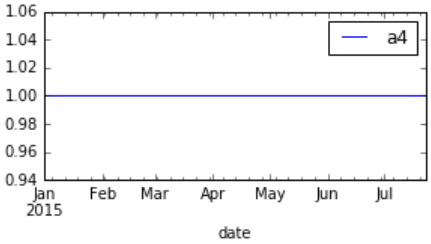

a4_ldiff
a4_log
Jan 03 2015
Jan 06 2015
Jan 09 2015
Jan 12 2015
Jan 15 2015
Jan 18 2015
Jan 21 2015
Jan 24 2015
Jan 27 2015
date

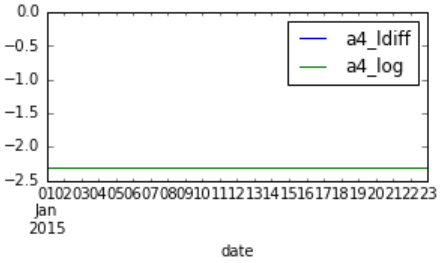

a4
Jan 03 2015
Jan 06 2015
Jan 09 2015
Jan 12 2015
Jan 15 2015
Jan 18 2015
Jan 21 2015
Jan 24 2015
Jan 27 2015
date

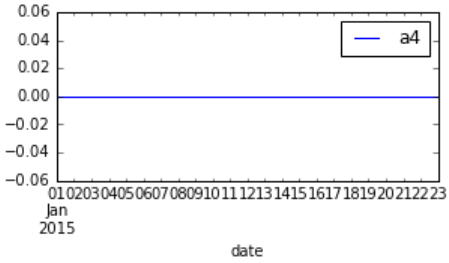

a4_ldiff
a4_log
Jan
2015
date

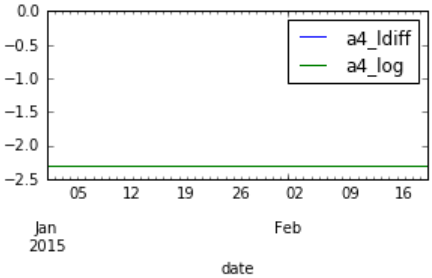

a4
Jan
2015
date

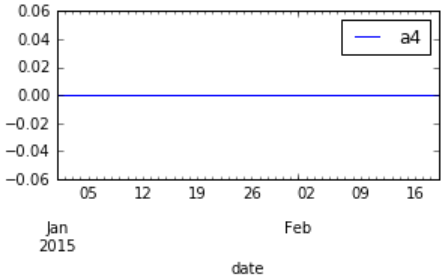

a4_ldiff
a4_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

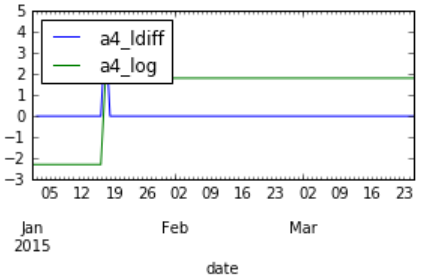

a4
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
date

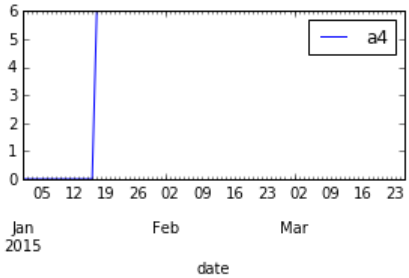

a4_ldiff
a4_log
Jan
2015
date

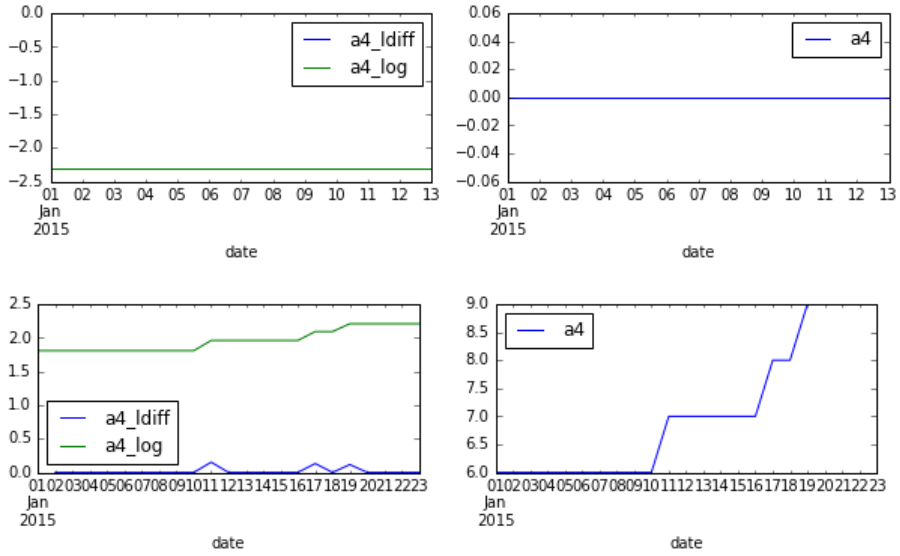

a4
Jan
2015
date


a4_ldiff
a4_log
Jan
2015
Feb
date


a4
Jan
2015
Feb
date


a4_ldiff
a4_log
Jan
2015
Feb
Mar
date


a4
Jan
2015
Feb
Mar
date



Summary

Good thing is a4 is significantly increasing recall (sensitivity)

Precision did go down a bit, but it's not that bad

Probably indicate that time to failure is a bit large once we detect failure

I do see signal in a4 for false positives.

This could turn out real good if we keep getting data from field to validate

Definitely inspect device and get more feedback from field

In [ ]:

Step 6: a1 Feature Analysis

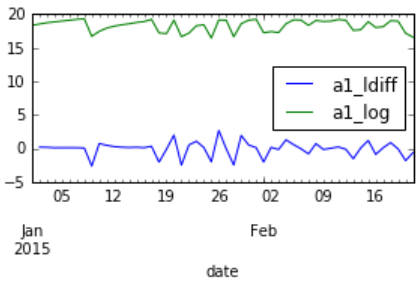
Use modules built for a2 and a7

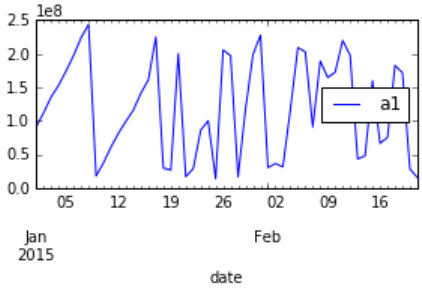
Really not expecting same strategy used for a2,a7,a4 to work for a1, but confirm

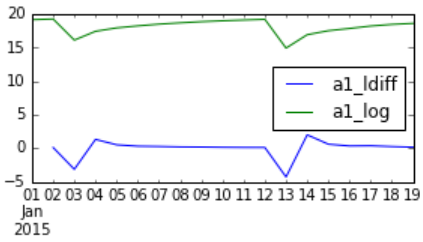
In [1]:
```
import pandas as pd
import sys
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np
from sklearn.linear_model import LinearRegression
%matplotlib inline
sys.path.append('../utils')
import DataAggregation as da
import AlgoUtils as au
cmap_bold = ListedColormap(['#00FF00','#FF0000'])
```

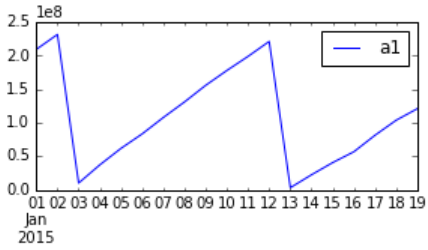
In [2]:
```
dd = da.GetFrames("../data/device_failure.csv", "a1", ldays=-300, lday_strict=False)
```

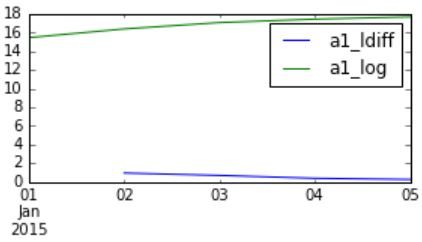
In [14]:
```
dd.plot_sample_history(dd.failed_devs["device"],10)
```

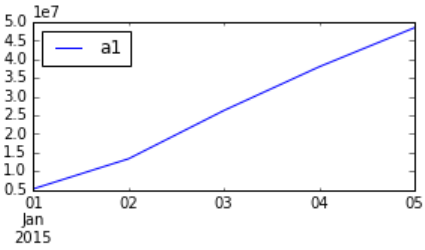

a1_ldiff
a1_log
Jan
2015
Feb
date


1e8
a1
Jan
2015
Feb
date

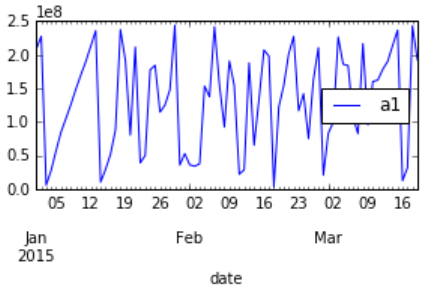

a1_ldiff
a1_log
Jan
2015
date


1e8
a1
Jan
2015
date

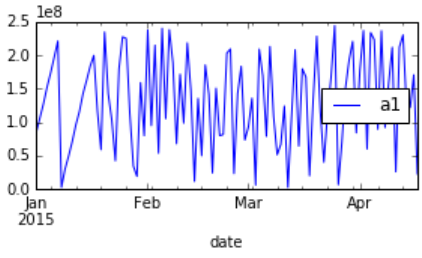

a1_ldiff
a1_log
Jan
2015
date

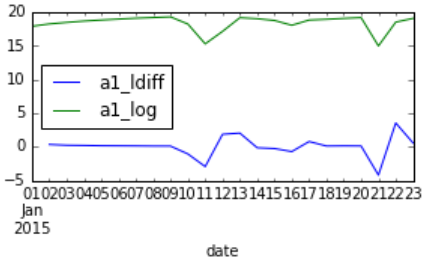

1e7
a1
Jan
2015
date

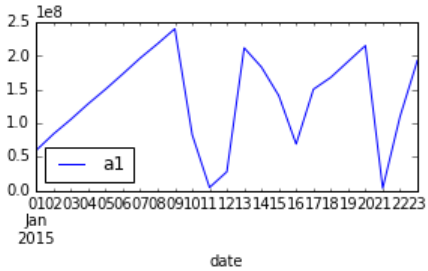

a1_ldiff
a1_log
Jan
2015
Feb
Mar
date

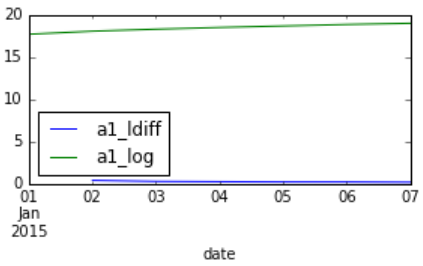

1e8
a1
Jan
2015
Feb
Mar
date

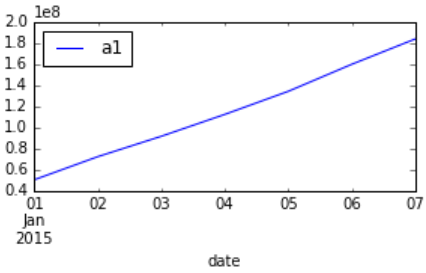

a1_ldiff
a1_log
Jan
2015
Feb
Mar
Apr
date

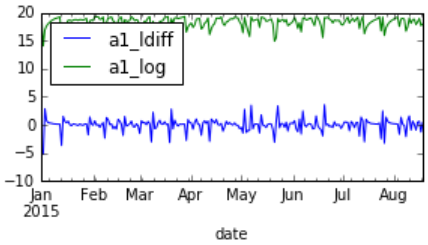

1e8
a1
Jan
2015
Feb
Mar
Apr
date

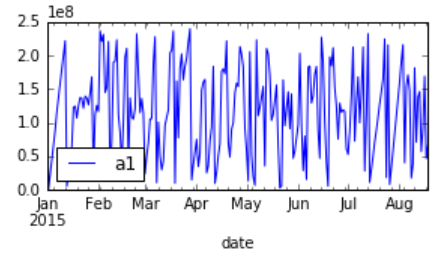

a1_ldiff
a1_log
Jan
2015
date

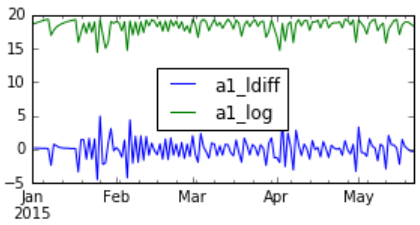

1e8
a1
Jan
2015
date

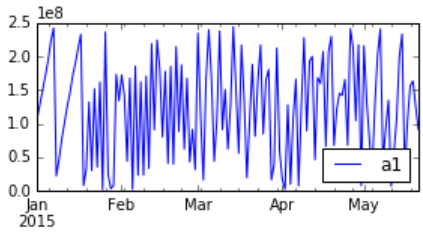

a1_ldiff
a1_log
01
Jan
2015
02
03
04
05
06
07
date

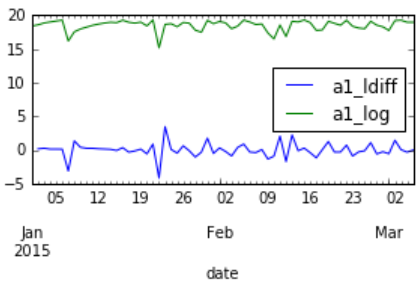

1e8
a1
01
Jan
2015
02
03
04
05
06
07
date

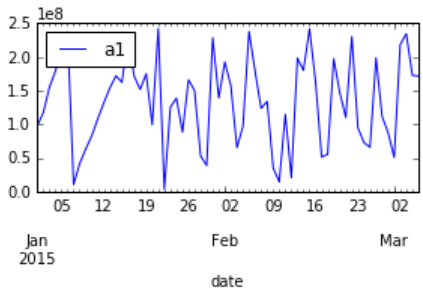

a1_ldiff
a1_log
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date


1e8
a1
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

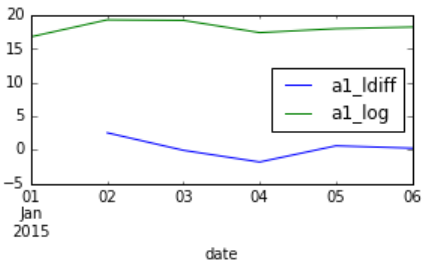

a1_ldiff
a1_log
Jan
2015
Feb
Mar
Apr
May
date

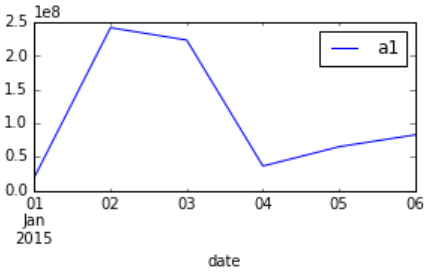

1e8
a1
Jan
2015
Feb
Mar
Apr
May
date

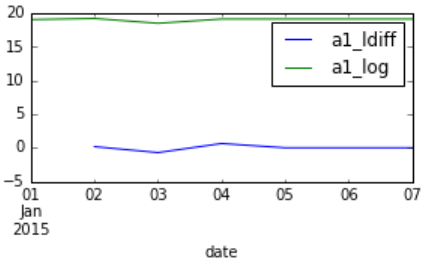

a1_ldiff
a1_log
05
12
19
26
02
09
16
23
02
Jan
2015
Feb
Mar
date

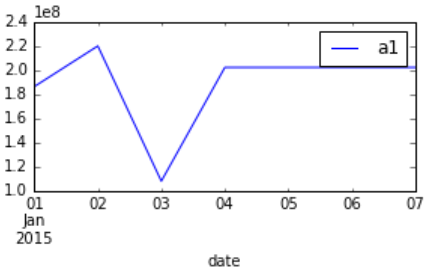

1e8
a1
05
12
19
26
02
09
16
23
02
Jan
2015
Feb
Mar
date

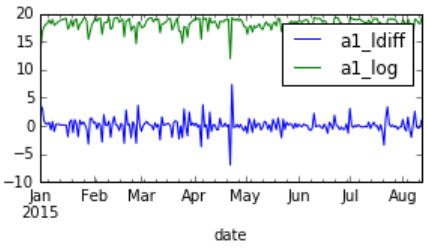
In [4]:
```
dd.plot_sample_history(dd.good_devs["device"],10)
```

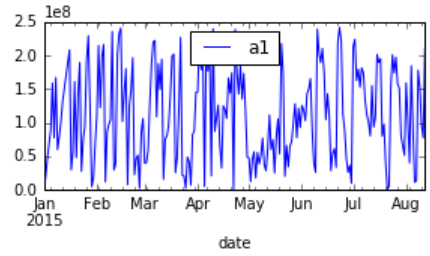

a1_ldiff
a1_log
date

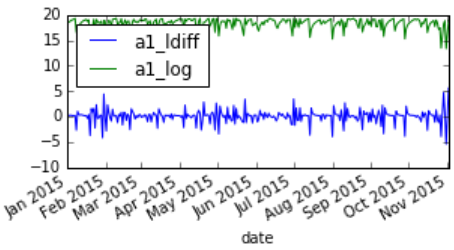

1e8
a1
date

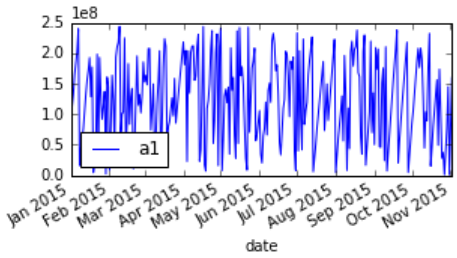

a1_ldiff
a1_log
date

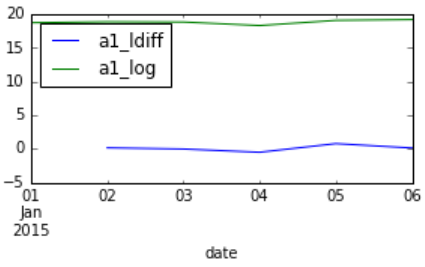

1e8
a1
date

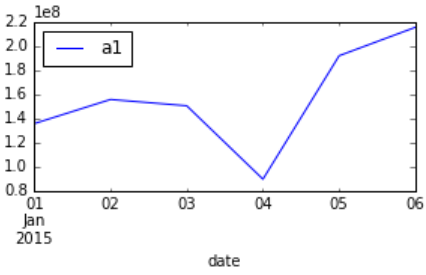

a1_ldiff
a1_log
Jan Feb Mar Apr May Jun Jul Aug
date

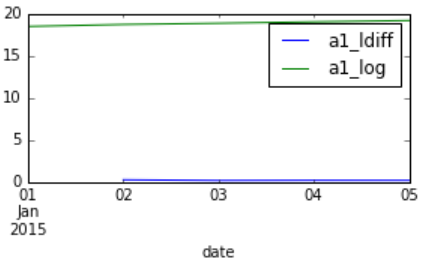

1e8
a1
Jan Feb Mar Apr May Jun Jul Aug
date

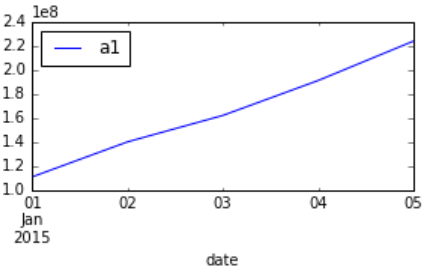

a1_ldiff
a1_log
Jan 2015 Feb 2015 Mar 2015 Apr 2015 May 2015 Jun 2015 Jul 2015 Aug 2015 Sep 2015 Oct 2015 Nov 2015
date

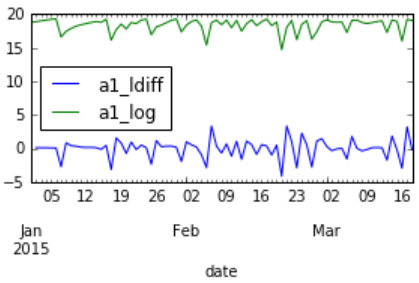

1e8
a1
Jan 2015 Feb 2015 Mar 2015 Apr 2015 May 2015 Jun 2015 Jul 2015 Aug 2015 Sep 2015 Oct 2015 Nov 2015
date

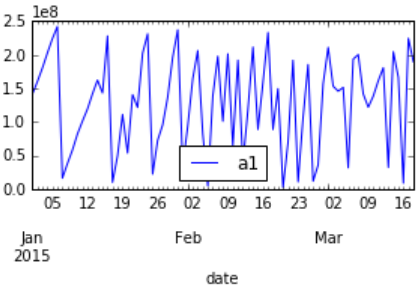

a1_ldiff
a1_log
date

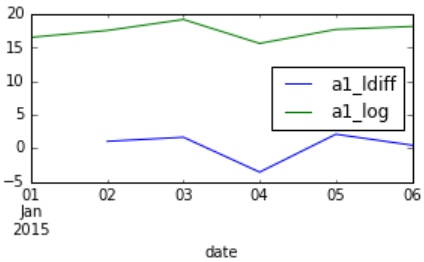

1e8
a1
date

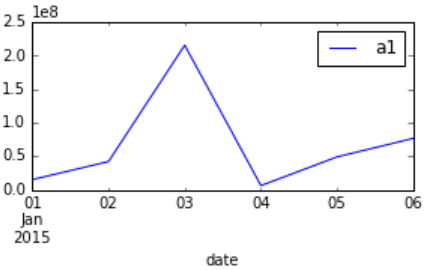

a1_ldiff
a1_log
date

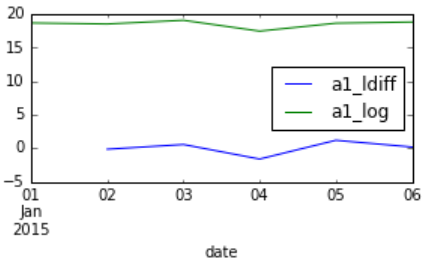

1e8
a1
date

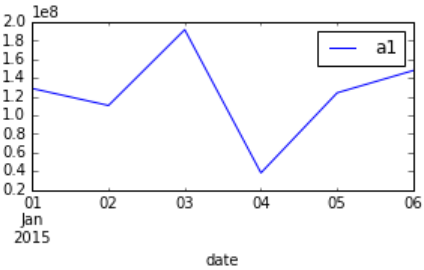

a1_ldiff
a1_log
Jan
2015
Feb
Mar
date

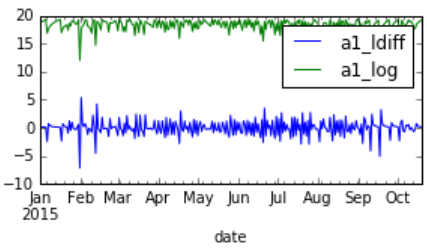

1e8
a1
Jan
2015
Feb
Mar
date

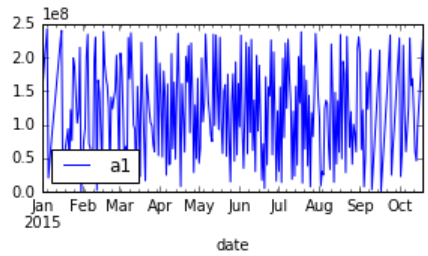

a1_ldiff
a1_log
Jan
2015
date


1e8
a1
Jan
2015
date


a1_ldiff
a1_log
Jan
2015
date

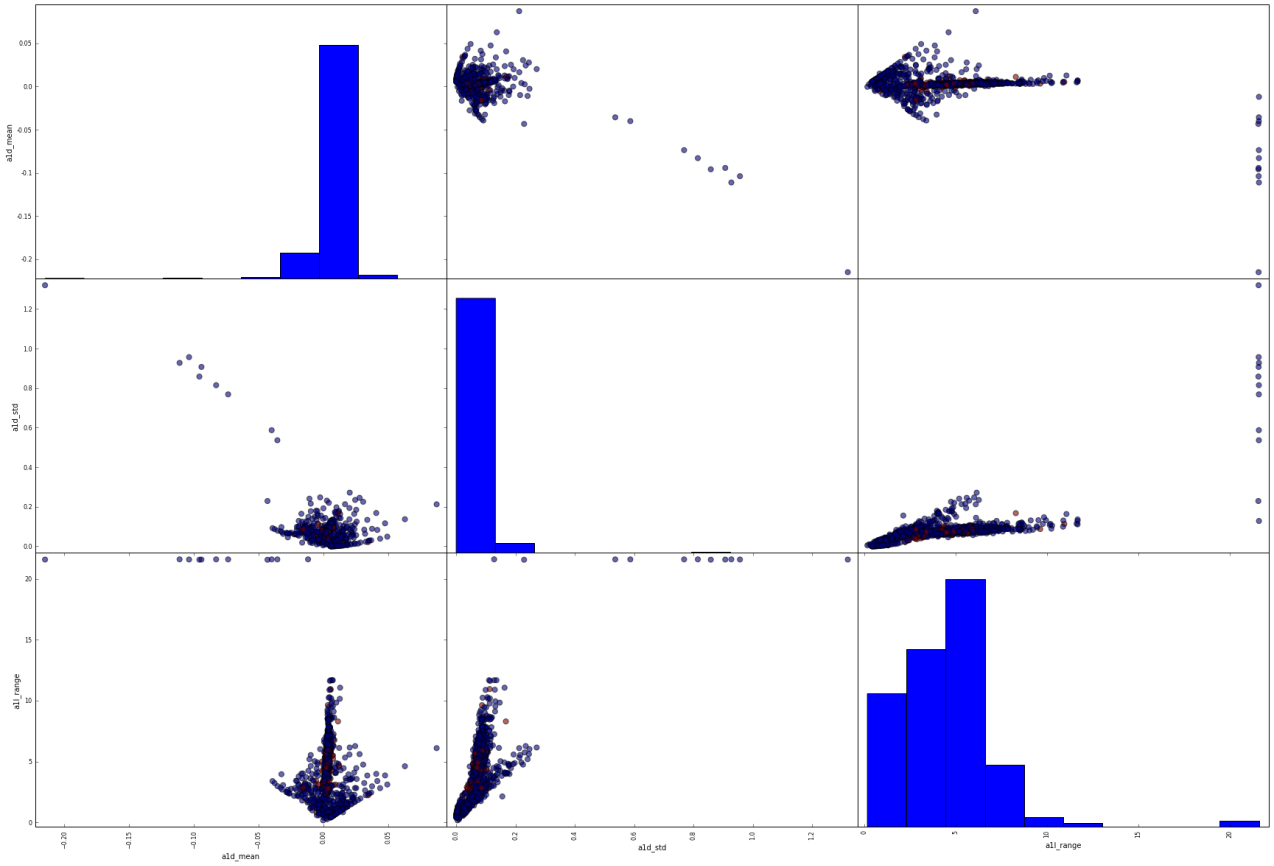

1e8
a1
Jan
2015
date


a1_ldiff
a1_log
Jan Feb Mar Apr May Jun Jul Aug Sep Oct
2015
date


1e8
a1
Jan Feb Mar Apr May Jun Jul Aug Sep Oct
2015
date

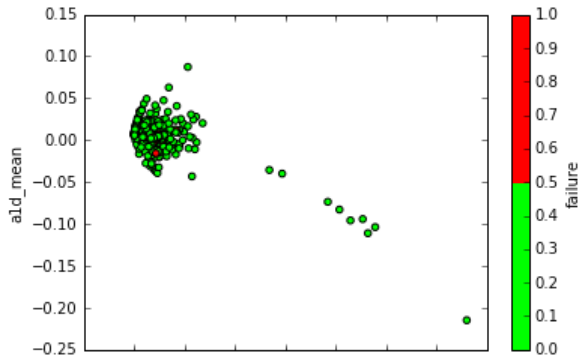
In [5]:
```
sfeature = dd.sfeature
fcols_tmp = [sfeature +"d_mean", sfeature +"d_std", sfeature +"l_range"]
df_sfeature = dd.df_sfeature
pd.scatter_matrix(df_sfeature[fcols_tmp], figsize=(30,20), s=200, c=df_sfeature["failure"], alpha=0.6)
```

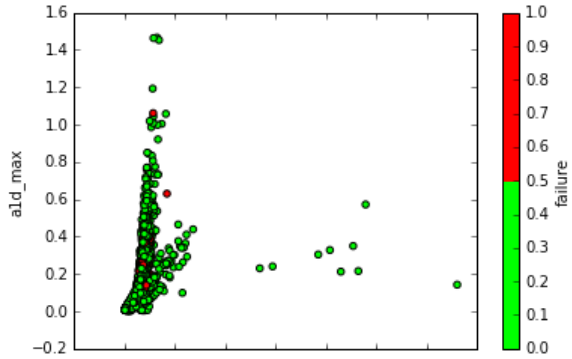
Out[5]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f00227c8e50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0022513d10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f00223a0390>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0022304310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f00220877d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0021feb0d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0021eac210>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0021bb10d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f001ea1a190>]], dtype=object)
```

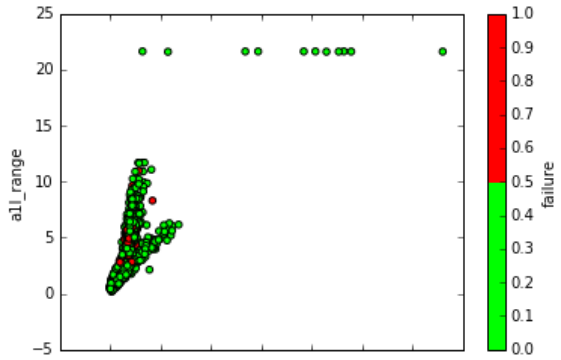


In [6]:
```
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_mean", c="failure", colormap=cmap_bold)
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"d_max", c="failure", colormap=cmap_bold)
df_sfeature.plot(kind='scatter', x=sfeature+'d_std', y=sfeature+"l_range", c="failure", colormap=cmap_bold)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f00227aadd0>







As expected, just don't see anything useful here

In [7]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [8]:
```
df_sfeature = dd.df_sfeature
sfeature = dd.sfeature
df_results = au.run_algo_analysis(df_sfeature, sfeature, fcols, algos_str, algos_dd)
```

```
/home/vagrant/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Undefine
dMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/vagrant/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1074: Undefine
dMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
----------
LogisticRegression:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.60
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.64
Cross-val-score(accuracy) = 0.60
Cross-val-score(recall)   = 0.57
Cross-val-score(precision)= 0.13
Cross-val-score(f1)       = 0.13
----------
KNeighborsClassifier:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.59
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.05
Cross-val-score(precision)= 0.45
Cross-val-score(f1)       = 0.45
----------
LinearDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.63
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:d_max:l_range
Cross-val-score(roc_auc) = 0.61
Cross-val-score(accuracy) = 0.52
Cross-val-score(recall)   = 0.65
Cross-val-score(precision)= 0.12
Cross-val-score(f1)       = 0.12
----------
LogisticRegression:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.57
Cross-val-score(accuracy) = 0.53
Cross-val-score(recall)   = 0.51
Cross-val-score(precision)= 0.10
Cross-val-score(f1)       = 0.10
----------
KNeighborsClassifier:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.60
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.01
Cross-val-score(precision)= 0.10
Cross-val-score(f1)       = 0.10
----------
LinearDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.57
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
QuadraticDiscriminantAnalysis:d_mean:d_std:l_range
Cross-val-score(roc_auc) = 0.62
Cross-val-score(accuracy) = 0.55
Cross-val-score(recall)   = 0.57
Cross-val-score(precision)= 0.12
Cross-val-score(f1)       = 0.12
----------
LogisticRegression:d_std:l_range
Cross-val-score(roc_auc) = 0.57
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
```

```
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:d_std:l_range
Cross-val-score(roc_auc) = 0.58
Cross-val-score(accuracy) = 0.53
Cross-val-score(recall)   = 0.53
Cross-val-score(precision)= 0.11
Cross-val-score(f1)       = 0.11
----------
KNeighborsClassifier:d_std:l_range
Cross-val-score(roc_auc) = 0.61
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.01
Cross-val-score(precision)= 0.10
Cross-val-score(f1)       = 0.10
----------
LinearDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.58
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
QuadraticDiscriminantAnalysis:d_std:l_range
Cross-val-score(roc_auc) = 0.63
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegression:l_range
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:l_range
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.53
Cross-val-score(recall)   = 0.57
Cross-val-score(precision)= 0.11
Cross-val-score(f1)       = 0.11
----------
KNeighborsClassifier:l_range
Cross-val-score(roc_auc) = 0.58
Cross-val-score(accuracy) = 0.90
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LinearDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.56
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
QuadraticDiscriminantAnalysis:l_range
Cross-val-score(roc_auc) = 0.65
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegression:d_std
Cross-val-score(roc_auc) = 0.49
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:d_std
Cross-val-score(roc_auc) = 0.49
Cross-val-score(accuracy) = 0.49
Cross-val-score(recall)   = 0.50
```

```
Cross-val-score(precision)= 0.08
Cross-val-score(f1)       = 0.08
----------
KNeighborsClassifier:d_std
Cross-val-score(roc_auc) = 0.57
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LinearDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.49
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
QuadraticDiscriminantAnalysis:d_std
Cross-val-score(roc_auc) = 0.58
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegression:d_max
Cross-val-score(roc_auc) = 0.48
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LogisticRegressionB:d_max
Cross-val-score(roc_auc) = 0.48
Cross-val-score(accuracy) = 0.49
Cross-val-score(recall)   = 0.46
Cross-val-score(precision)= 0.08
Cross-val-score(f1)       = 0.08
----------
KNeighborsClassifier:d_max
Cross-val-score(roc_auc) = 0.46
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
LinearDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.48
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
----------
QuadraticDiscriminantAnalysis:d_max
Cross-val-score(roc_auc) = 0.59
Cross-val-score(accuracy) = 0.91
Cross-val-score(recall)   = 0.00
Cross-val-score(precision)= 0.00
Cross-val-score(f1)       = 0.00
```

In [10]:
```
df_results
```

Out[10]:

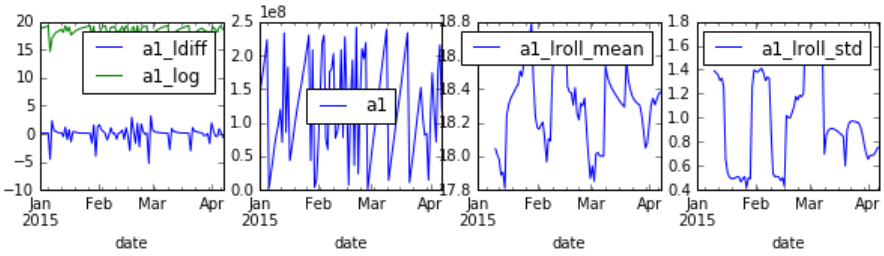
	fcols	algo	recall	precision	f1	roc_auc	accuracy
4	d_mean:d_std:d_max:l_range	QuadraticDiscriminantAnalysis	0.65	0.12	0.20	0.61	0.52
1	d_mean:d_std:d_max:l_range	LogisticRegressionB	0.57	0.13	0.21	0.64	0.60
9	d_mean:d_std:l_range	QuadraticDiscriminantAnalysis	0.57	0.12	0.19	0.62	0.55
16	l_range	LogisticRegressionB	0.57	0.11	0.18	0.56	0.53
11	d_std:l_range	LogisticRegressionB	0.53	0.11	0.18	0.58	0.53
6	d_mean:d_std:l_range	LogisticRegressionB	0.51	0.10	0.17	0.57	0.53
21	d_std	LogisticRegressionB	0.50	0.08	0.14	0.49	0.49
26	d_max	LogisticRegressionB	0.46	0.08	0.14	0.48	0.49
2	d_mean:d_std:d_max:l_range	KNeighborsClassifier	0.05	0.45	0.09	0.59	0.91
12	d_std:l_range	KNeighborsClassifier	0.01	0.10	0.02	0.61	0.91
7	d_mean:d_std:l_range	KNeighborsClassifier	0.01	0.10	0.02	0.60	0.91
19	l_range	QuadraticDiscriminantAnalysis	0.00	0.00	0.00	0.65	0.91
3	d_mean:d_std:d_max:l_range	LinearDiscriminantAnalysis	0.00	0.00	0.00	0.63	0.91
14	d_std:l_range	QuadraticDiscriminantAnalysis	0.00	0.00	0.00	0.63	0.91
0	d_mean:d_std:d_max:l_range	LogisticRegression	0.00	0.00	0.00	0.60	0.91
29	d_max	QuadraticDiscriminantAnalysis	0.00	0.00	0.00	0.59	0.91
13	d_std:l_range	LinearDiscriminantAnalysis	0.00	0.00	0.00	0.58	0.91
24	d_std	QuadraticDiscriminantAnalysis	0.00	0.00	0.00	0.58	0.91
17	l_range	KNeighborsClassifier	0.00	0.00	0.00	0.58	0.90
8	d_mean:d_std:l_range	LinearDiscriminantAnalysis	0.00	0.00	0.00	0.57	0.91
10	d_std:l_range	LogisticRegression	0.00	0.00	0.00	0.57	0.91
22	d_std	KNeighborsClassifier	0.00	0.00	0.00	0.57	0.91
5	d_mean:d_std:l_range	LogisticRegression	0.00	0.00	0.00	0.56	0.91
15	l_range	LogisticRegression	0.00	0.00	0.00	0.56	0.91
18	l_range	LinearDiscriminantAnalysis	0.00	0.00	0.00	0.56	0.91
20	d_std	LogisticRegression	0.00	0.00	0.00	0.49	0.91
23	d_std	LinearDiscriminantAnalysis	0.00	0.00	0.00	0.49	0.91
25	d_max	LogisticRegression	0.00	0.00	0.00	0.48	0.91
28	d_max	LinearDiscriminantAnalysis	0.00	0.00	0.00	0.48	0.91
27	d_max	KNeighborsClassifier	0.00	0.00	0.00	0.46	0.91

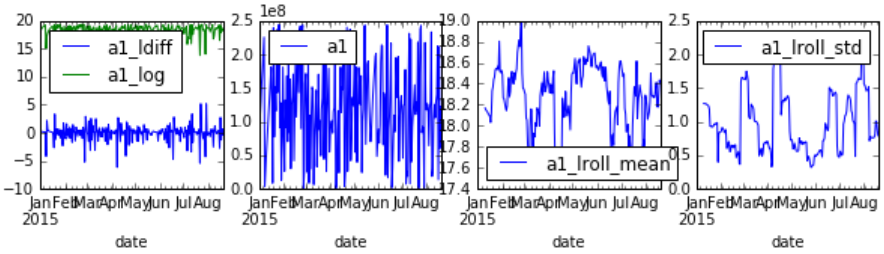
Even though recall is high, precision, accuracy and roc_auc are too low

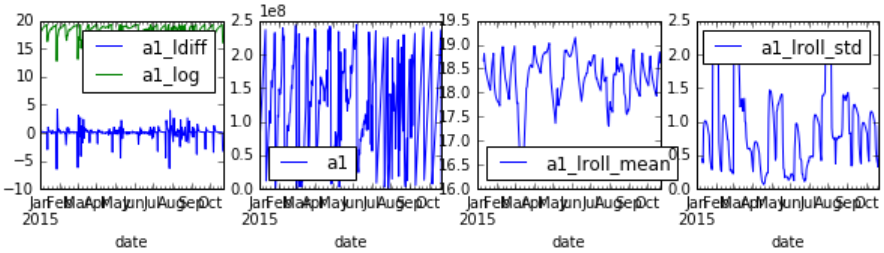
This is not right approach.

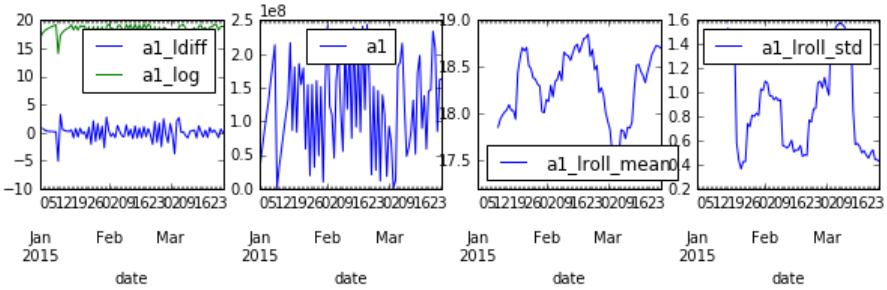
See if rolling averages and rolling standard deviations look different in good vs bad

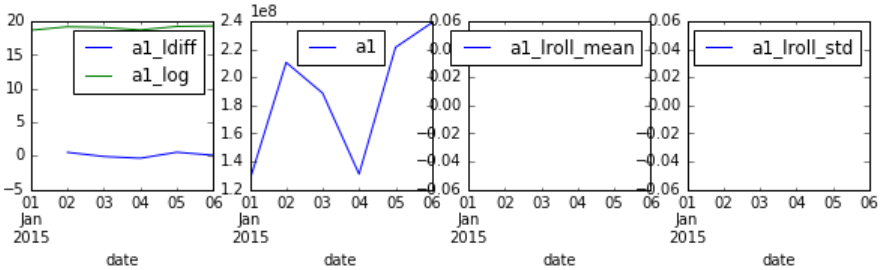
In [11]:
```
dd.plot_sample_history(dd.good_devs["device"], 20, plot_type="rolling")
```

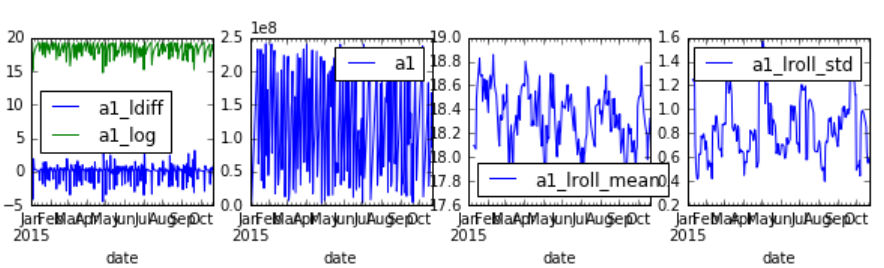

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
Apr
2015
date

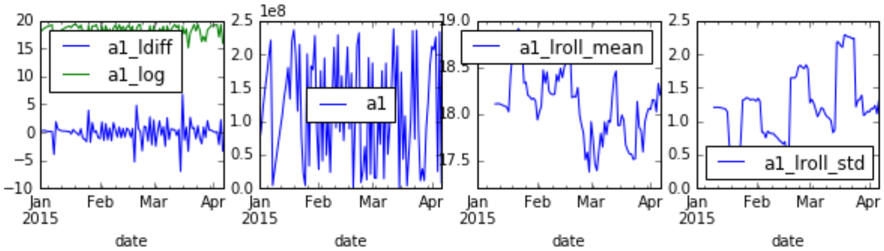

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
2015
date

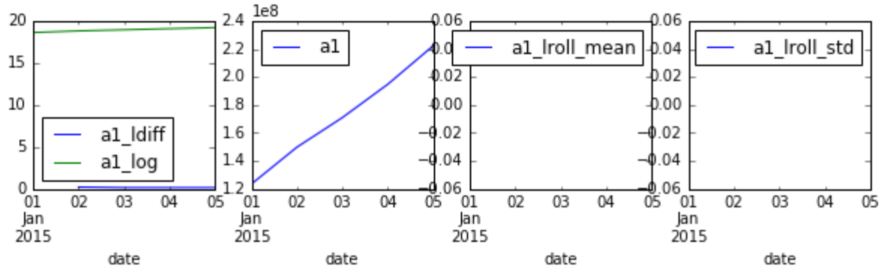

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
2015
date

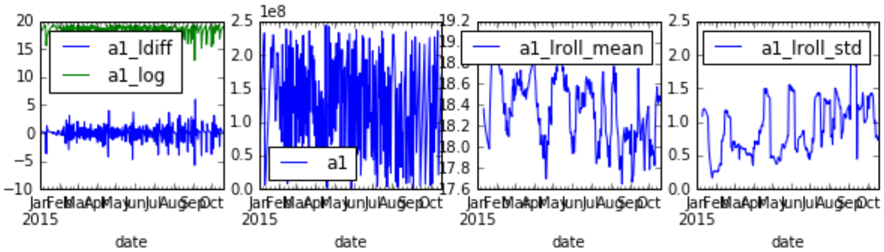

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
2015
date

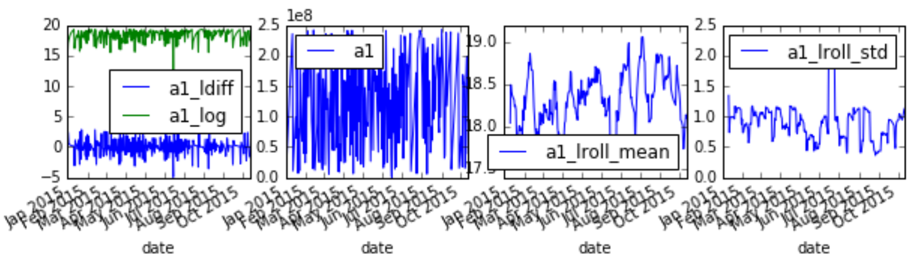

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
2015
date

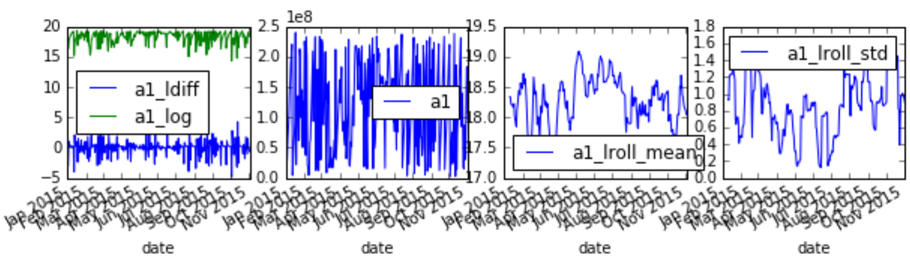

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
2015
date

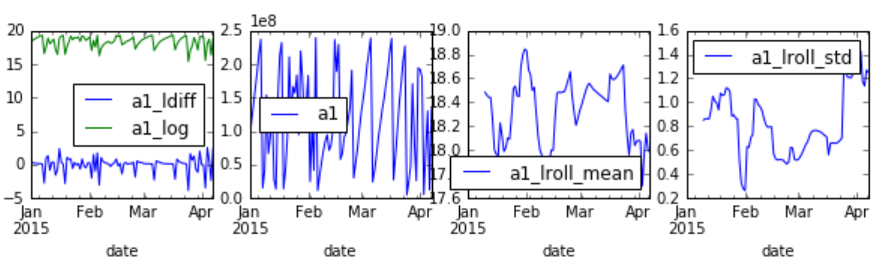

a1_ldiff
a1_log
a1
1e8
a1_lroll_mean
a1_lroll_std
Jan 2015
Feb
Mar
Apr
date

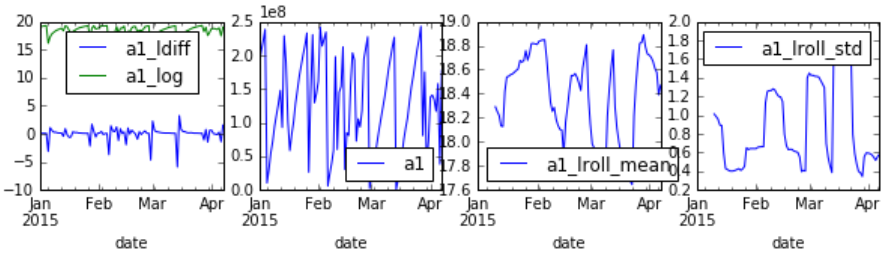

a1_ldiff
a1_log
a1
1e8
a1_lroll_mean
a1_lroll_std
01 Jan 2015
02
03
04
05
date

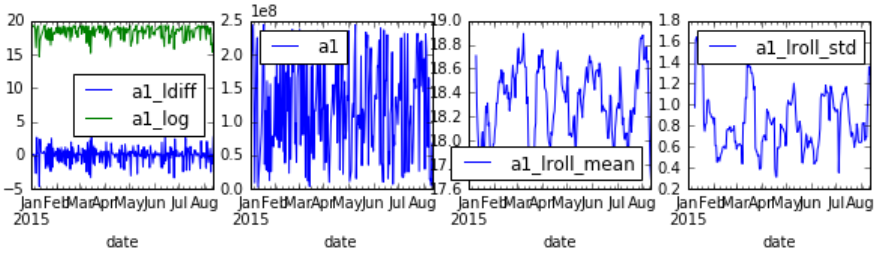

a1_ldiff
a1_log
a1
1e8
a1_lroll_mean
a1_lroll_std
date

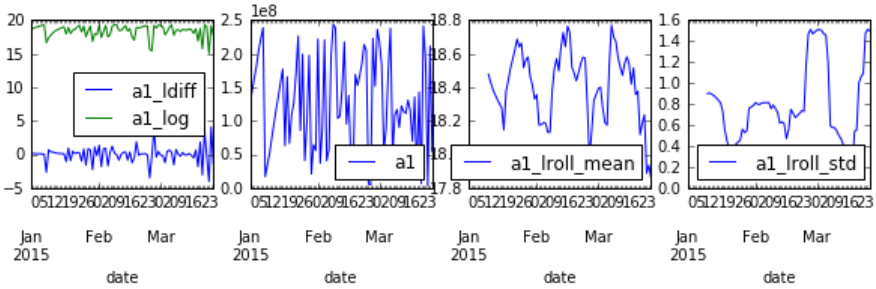

a1_ldiff
a1_log
a1
1e8
a1_lroll_mean
a1_lroll_std
date

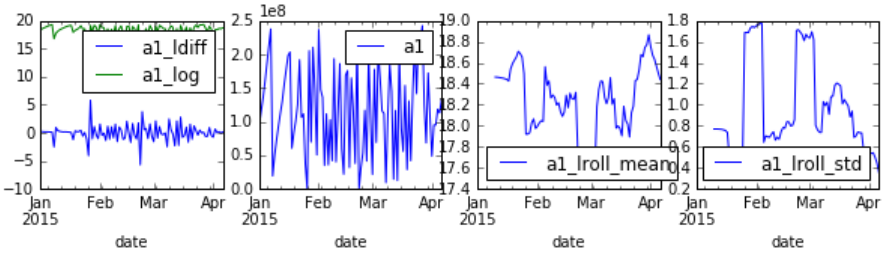

a1_ldiff
a1_log
a1
1e8
a1_lroll_mean
a1_lroll_std
date

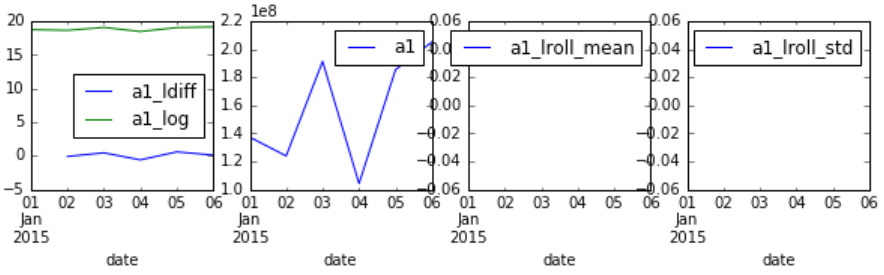

a1_ldiff
a1_log
a1
1e8
a1_lroll_mean
a1_lroll_std
Jan 2015
Feb
Mar
Apr
date

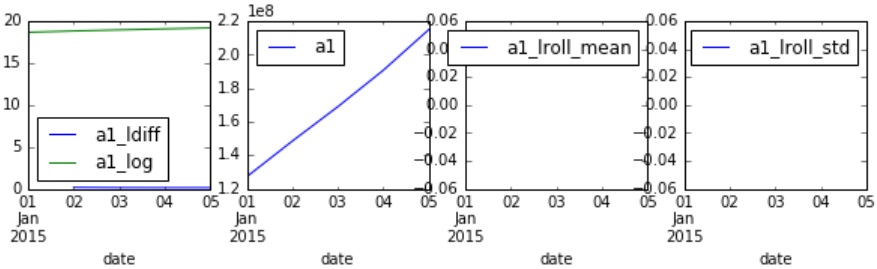

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
Mar
Apr
date

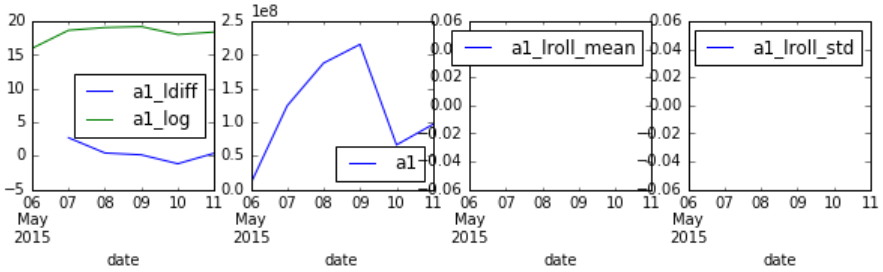

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
Mar
Apr
May
Jun
Jul
Aug
date

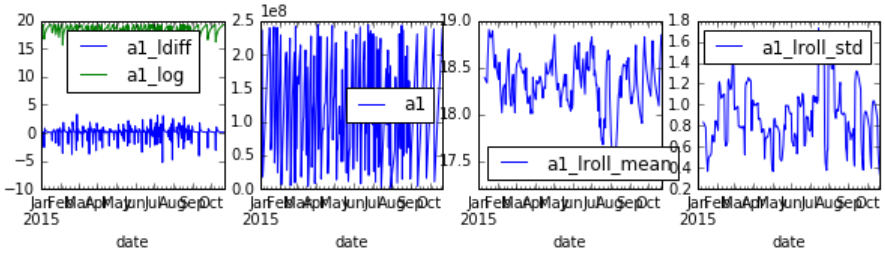

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
Mar
date


a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
Mar
Apr
date

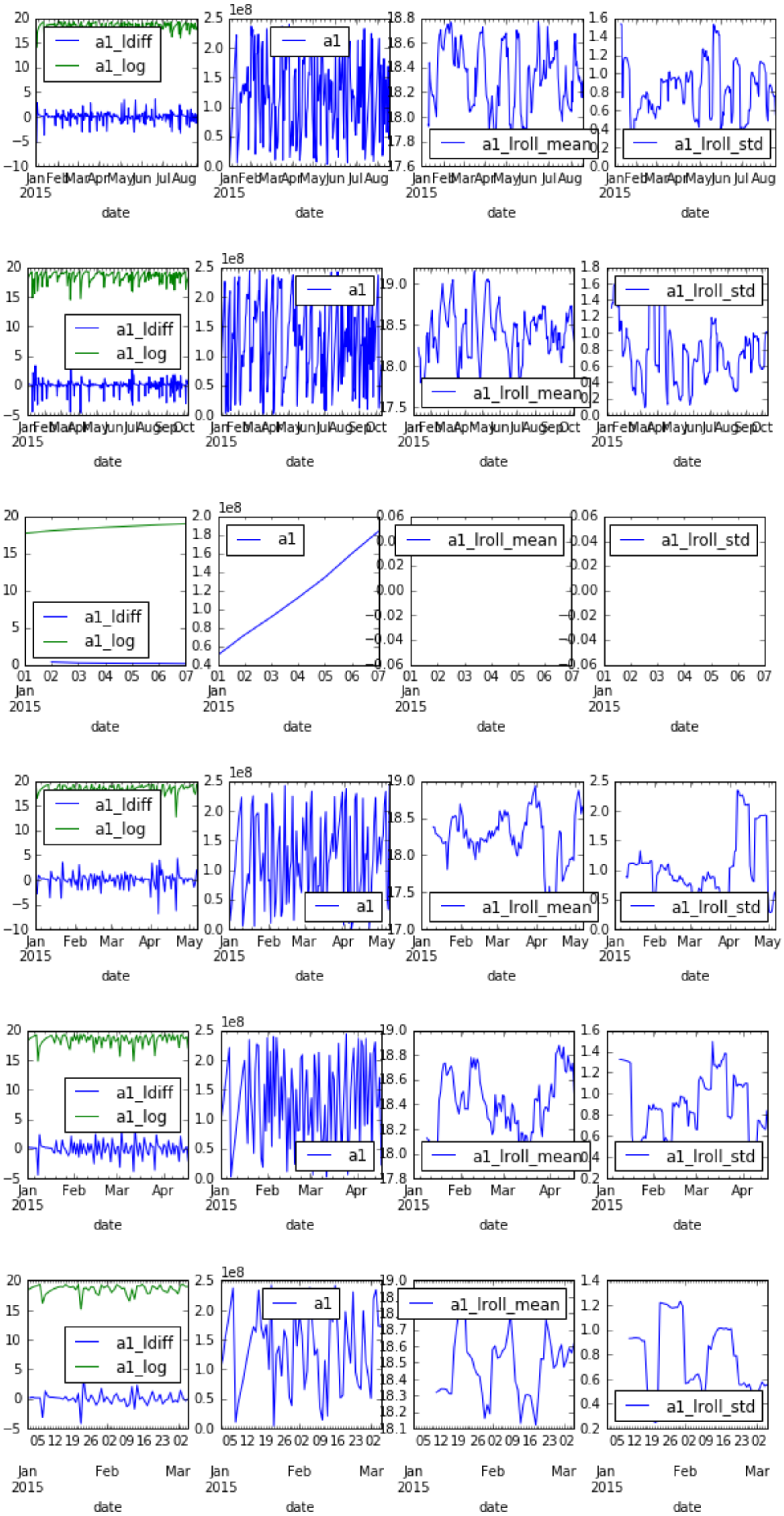

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
01
Jan
2015
02
03
04
05
06
date

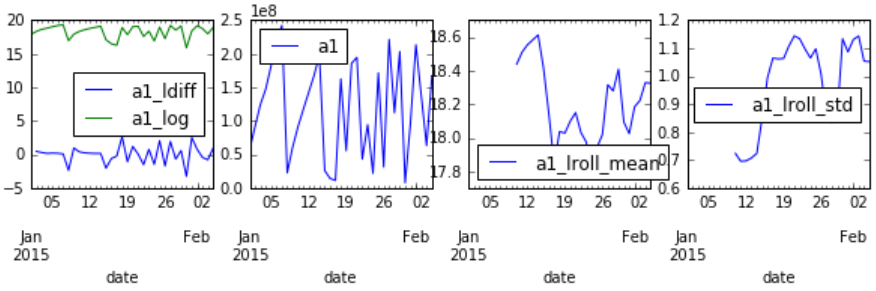

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
01
Jan
2015
02
03
04
05
date

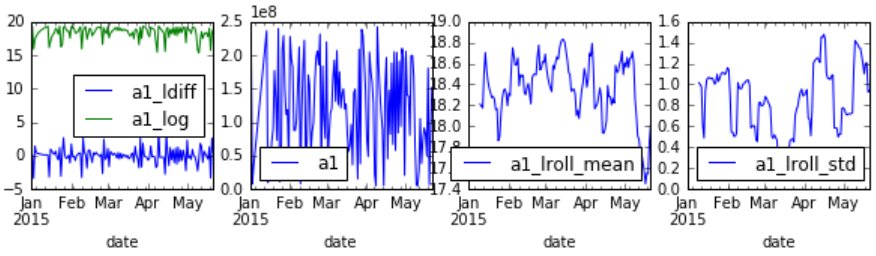

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
06
07
08
09
10
11
May
2015
date

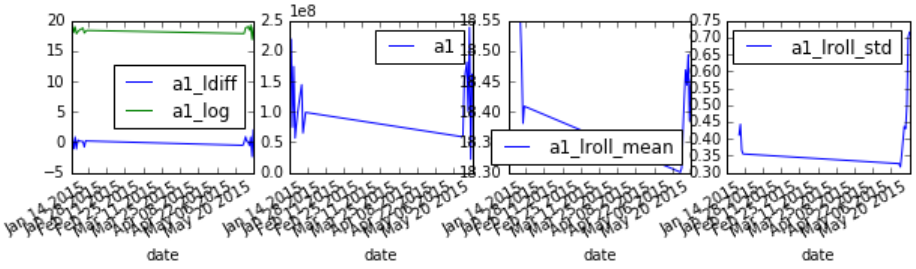

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
2015
date

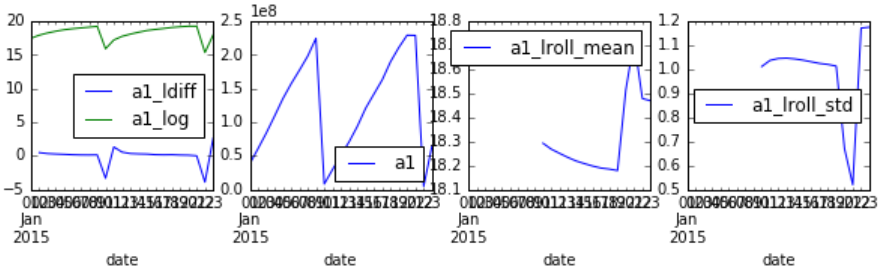
In [13]:
```
dd.plot_sample_history(dd.failed_devs["device"], 20, plot_type="roll")
```

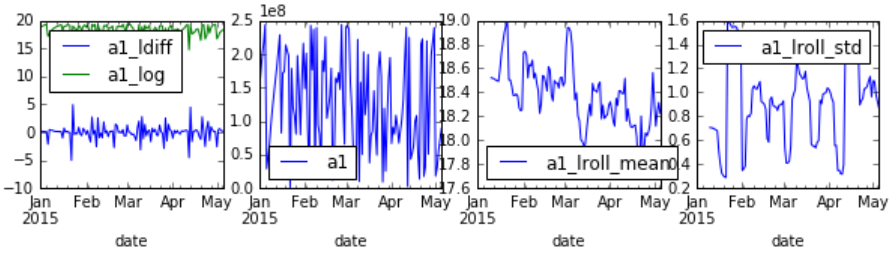

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
date
2015

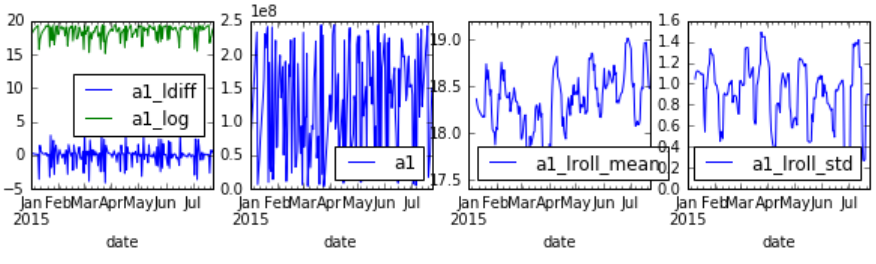

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
date

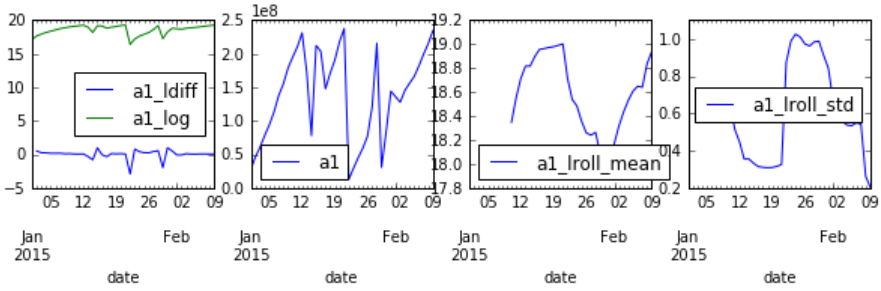

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
Mar
Apr
May
date

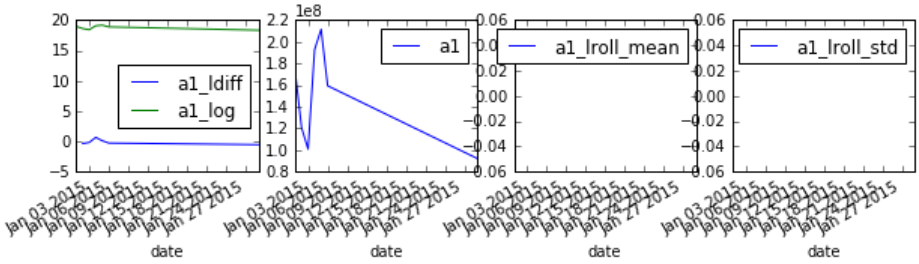

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
date

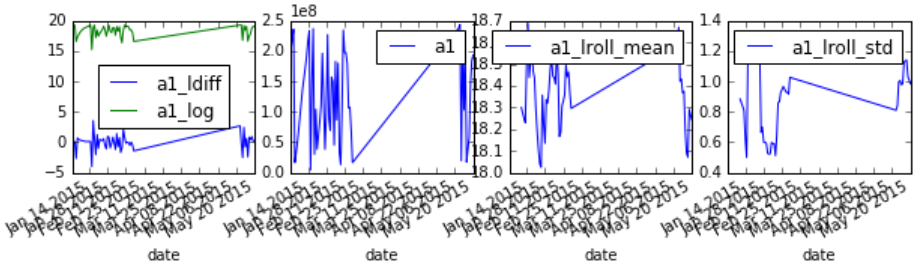

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
date

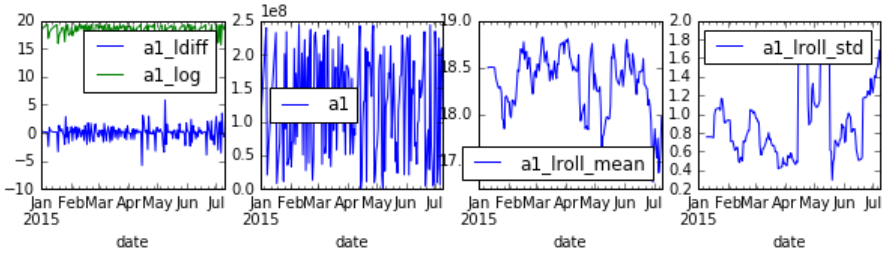

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
Mar
Apr
May
date

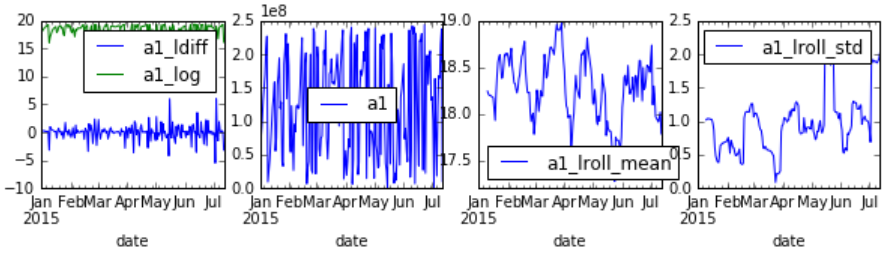

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
Feb
Mar
Apr
May
Jun
Jul
2015
date

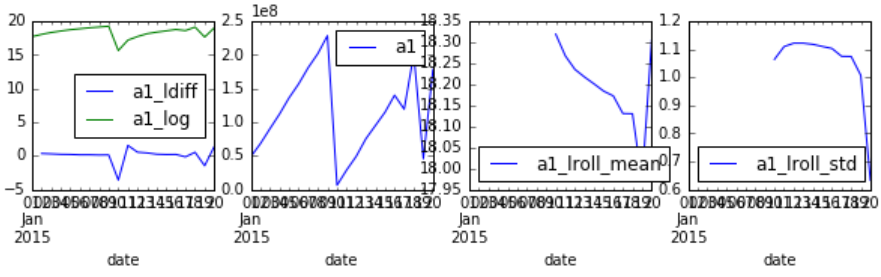

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
Feb
date

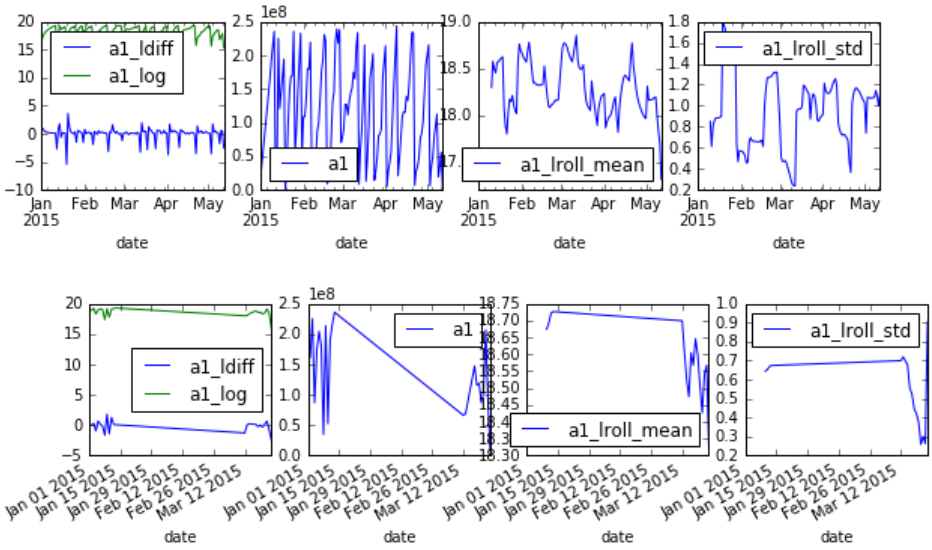

a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
date


a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
date


a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan Feb Mar Apr May Jun Jul
2015
date


a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan Feb Mar Apr May Jun Jul
2015
date


a1_ldiff
a1_log
a1
a1_lroll_mean
a1_lroll_std
1e8
Jan
2015
date



These features don't look promising either

Try using bias and coeifficient of regression line of values as feature

Purpose is to see if overall trend is going up or down

Don't think it will work, but should give it a try

In [3]:
```
def get_lnr_coef_intercept(tdf, dev):
    ttdf = tdf[tdf["device"] == dev][["date","a1"]]
    ttdf["day"] = ttdf.date.map(lambda x: x.month*30 + x.day)
    y = np.log(ttdf["a1"] + 0.01)
    X = pd.DataFrame(ttdf["day"])
    reg = LinearRegression()
    reg.fit(X,y)
    return (reg.coef_[0], reg.intercept_)
```

In [4]:
```
def getlnrcoef_df(tdf, alldevs, feature):
    dd = dict()
    for dev in alldevs.index.values:
        dd[dev] = dict()
        coef, bias = get_lnr_coef_intercept(tdf, dev)
        dd[dev]["coef"] = coef
        dd[dev]["bias"] = bias
    return pd.DataFrame(dd).transpose().join(alldevs)
```

In [5]:
```
get_lnr_coef_intercept(dd.df, "S1F013BB")
```

Out[5]:
```
(0.35233196919027882, -37.601320719261153)
```

In [6]:
```
ldf = getlnrcoef_df(dd.df, dd.all_devs, "a1")
```

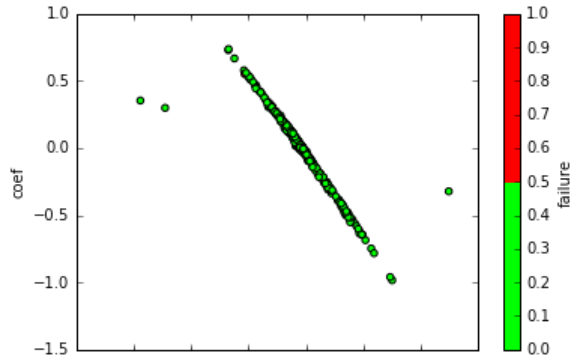
In [7]:
```
ldf.head()
```

Out[7]:

	bias	coef	failure
S1F01085	8.705682	0.276836	0
S1F013BB	-37.601321	0.352332	0
S1F0166B	27.670013	-0.286422	0
S1F01E6Y	18.578027	-0.001026	0
S1F01JE0	13.761807	0.152268	0

In [8]:
```
ldf.plot(kind='scatter', x='bias', y='coef', c="failure", colormap=cmap_bold)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0aad84f1d0>
```



This looks just as useless

In [9]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {},
    "SVC": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [11]:
```
algo_str = "QuadraticDiscriminantAnalysis"
scols = ["bias", "coef"]
analysisdf = au.do_clf_validate_new(ldf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.60
Cross-val-score(accuracy) = 0.44
Cross-val-score(recall)   = 0.78
Cross-val-score(precision)= 0.11
Cross-val-score(f1)       = 0.11
```

a1 Summary:

Precision and Accuracy are too low to take these derived features seriously

Need to do further investigation and take some advise from any Senior Signal Processing Scientist if necessary

Considering data quality issues, not sure if it is good idea to even try frequency component analysis. For frequency analysis to make sense, we at least need all devices data every day for last 64/128/256 days very consistently.

In [ ]:
```
```

Step 7: Feature Selection

Based on Steps 1-6. Only doing feature selection among a2, a7 and a4 derived features

In [2]:
```
import pandas as pd
import sys
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression

from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import RFE

from sklearn.linear_model import RandomizedLasso

%matplotlib inline
cmap_bold = ListedColormap(['#00FF00','#FF0000'])
```

In [4]:
```
sys.path.append('../utils')
```

In [5]:
```
import DataAggregation as da
import AlgoUtils as au
```

In [6]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {},
    "SVC": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [7]:
```
a2 = da.GetFrames("../data/device_failure.csv", "a2")
a7 = da.GetFrames("../data/device_failure.csv", "a7")
a4 = da.GetFrames("../data/device_failure.csv", "a4", ldays=-30, lday_strict=False)
tdf = a2.df_sfeature.drop("failure", axis=1).join(a7.df_sfeature.drop("failure", axis=1)).join(a4.df_s
feature)
```

In [8]:
```
tdf.head()
```

Out[8]:

	a2d_max	a2d_mean	a2d_std	a2l_range	a7d_max	a7d_mean	a7d_std	a7l_range	a4d_max	a4d_mean	a4d_std
S1F01085	0	0	0	0	0	0	0	0	0	0	0
S1F013BB	0	0	0	0	0	0	0	0	0	0	0
S1F0166B	0	0	0	0	0	0	0	0	0	0	0
S1F01E6Y	0	0	0	0	0	0	0	0	0	0	0
S1F01JE0	0	0	0	0	0	0	0	0	0	0	0

Baseline of what I think will work the best

In [10]:
```
algo_str = "QuadraticDiscriminantAnalysis"
scols = ["a2d_std", "a7d_std","a7l_range","a7d_mean", "a7d_max"]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.72
Cross-val-score(accuracy) = 0.94
Cross-val-score(recall)   = 0.42
Cross-val-score(precision)= 0.80
Cross-val-score(f1)       = 0.80
```

Baseline with all features

In [11]:
```
algo_str = "QuadraticDiscriminantAnalysis"
scols = tdf.columns[:-1]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.78
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.56
Cross-val-score(precision)= 0.65
Cross-val-score(f1)       = 0.65
```

Recall is definitely much better with a4 derived Features!

Loss in precision might be worth it if we can do hypothesis testing from field data

Let's see if we can get similar or better performance with lesser features

In [ ]:
```

```

Variance Threshold based Feature selection

In [16]:
```
X = tdf[tdf.columns[:-1]]
y = tdf["failure"]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
X_new = sel.fit_transform(X)
sel.get_support()
cnt = 0
allcols = tdf.columns[:-1]
for decision in sel.get_support():
    if decision == True:
        print allcols[cnt]
    cnt = cnt + 1
```

```
a2l_range
a7l_range
a4d_max
a4l_range
```

In [17]:
```
#Adding a2l_range, a4d_max and a4l_range based on above analysis
scols = ["a2l_range", "a2d_std", "a7d_std","a7l_range","a4d_max", "a4l_range"]
algo_str = "QuadraticDiscriminantAnalysis"
scols = ["a2l_range", "a2d_std", "a7d_std","a7l_range","a4d_max", "a4l_range"]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.78
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.51
Cross-val-score(precision)= 0.65
Cross-val-score(f1)       = 0.65
```

Variance Threshold Feature Selection Summary

a2l_range, a4d_max and a4l_range: Definitely adding more sensitivity (expense of precision)

But for same precision, All features have recall = 0.56 (better than 0.51)

In [ ]:

Randomized Lasso

(Surprisingly, of no use: TBD: Need to understand why)

In [23]:
```
fcols = tdf.columns[:-1]
X = tdf[fcols]
Y = tdf["failure"]
rlasso = RandomizedLasso(alpha=0.025)
rlasso.fit(X, Y)

print "Features sorted by their score:"
print sorted(zip(map(lambda x: round(x, 4), rlasso.scores_),
                 fcols), reverse=True)
```

```
Features sorted by their score:
[(0.0, 'a7l_range'), (0.0, 'a7d_std'), (0.0, 'a7d_mean'), (0.0, 'a7d_max'), (0.0, 'a4l_range'), (0.
0, 'a4d_std'), (0.0, 'a4d_mean'), (0.0, 'a4d_max'), (0.0, 'a2l_range'), (0.0, 'a2d_std'), (0.0, 'a2d
_mean'), (0.0, 'a2d_max')]
```

In [ ]:

Recursive Feature Elimination (QDA doesn't support RFE). Use LogisticRegression instead

In [24]:
```
fcols = [x for x in tdf.columns[:-1]]
X = tdf[fcols]
Y = tdf["failure"]
clf = LogisticRegression()
rfe = RFE(clf, n_features_to_select=2)
rfe.fit(X,Y)

print "Features sorted by their rank:"
print sorted(zip(rfe.ranking_, fcols))
```

```
Features sorted by their rank:
[(1, 'a2l_range'), (1, 'a7l_range'), (2, 'a7d_max'), (3, 'a4d_mean'), (4, 'a4l_range'), (5, 'a7d_st
d'), (6, 'a7d_mean'), (7, 'a4d_std'), (8, 'a2d_std'), (9, 'a2d_mean'), (10, 'a2d_max'), (11, 'a4d_ma
x')]
```

In [25]:
```
#a4d_mean: Need to add this
#a2l_range, a4l_range: Already added this based on Variance Threshold analysis
#a7l_range, a7d_max, a7d_std, a7d_mean: This is already in baseline
algo_str = "QuadraticDiscriminantAnalysis"
scols = ["a2l_range", "a2d_std", "a7d_std","a7l_range","a4d_max", "a4l_range", "a4d_mean"]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.79
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.55
Cross-val-score(precision)= 0.64
Cross-val-score(f1)       = 0.64
```

Summary for RFE

Adding a4d_mean definitely boosting recall, without too much impact on precision

In [ ]:

Correlation of failure with all variables

In [27]:
```
fcdf = pd.DataFrame(tdf.corr()["failure"])
fcdf.loc[:,"failure_abs"] = fcdf.failure.map(lambda x: abs(x))
fcdf.sort_values(by="failure_abs", ascending=False, inplace=True)
fcdf.drop(["failure_abs"], axis=1, inplace=True)
fcdf
```

Out[27]:

	failure
failure	1.000000
a4l_range	0.482712
a7d_std	0.453876
a7l_range	0.447489
a7d_mean	-0.421852
a2l_range	0.386144
a2d_std	0.346241
a2d_max	0.319640
a2d_mean	-0.287751
a4d_std	0.247814
a7d_max	0.203912
a4d_max	0.159966
a4d_mean	0.051375

In [ ]:

For QDA, what matters most is difference in correlation between different classes

In [29]:
```
tdf[tdf["failure"] == 0].corr()
```

Out[29]:

	a2d_max	a2d_mean	a2d_std	a2l_range	a7d_max	a7d_mean	a7d_std	a7l_range	a4d_max	a4d_mean	a4d_s
a2d_max	1.000000	0.876467	0.979157	0.924475	0.280718	0.004785	0.116430	0.155355	0.011553	0.055736	0.029
a2d_mean	0.876467	1.000000	0.830472	0.820134	0.405954	-0.008970	0.178529	0.236265	0.013675	0.064890	0.039
a2d_std	0.979157	0.830472	1.000000	0.959579	0.248684	0.012098	0.098964	0.130937	0.009871	0.057031	0.026
a2l_range	0.924475	0.820134	0.959579	1.000000	0.421380	-0.018684	0.189667	0.254386	0.019428	0.066123	0.047
a7d_max	0.280718	0.405954	0.248684	0.421380	1.000000	-0.158516	0.497531	0.706093	0.006534	-0.027512	0.066
a7d_mean	0.004785	-0.008970	0.012098	-0.018684	-0.158516	1.000000	-0.925136	-0.782458	0.000540	0.014731	-0.011
a7d_std	0.116430	0.178529	0.098964	0.189667	0.497531	-0.925136	1.000000	0.957153	0.002103	-0.018053	0.033
a7l_range	0.155355	0.236265	0.130937	0.254386	0.706093	-0.782458	0.957153	1.000000	0.003069	-0.030069	0.047
a4d_max	0.011553	0.013675	0.009871	0.019428	0.006534	0.000540	0.002103	0.003069	1.000000	0.971616	0.985
a4d_mean	0.055736	0.064890	0.057031	0.066123	-0.027512	0.014731	-0.018053	-0.030069	0.971616	1.000000	0.938
a4d_std	0.029822	0.039523	0.026434	0.047078	0.066296	-0.011642	0.033861	0.047045	0.985479	0.938878	1.000
a4l_range	0.115258	0.153533	0.105819	0.163158	0.276579	-0.045010	0.139098	0.194575	0.365307	0.264671	0.436
failure	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [30]:
```
tdf[tdf["failure"] == 1][fcols].corr()
```

Out[30]:

	a2d_max	a2d_mean	a2d_std	a2l_range	a7d_max	a7d_mean	a7d_std	a7l_range	a4d_max	a4d_mean	a4d
a2d_max	1.000000	-0.187700	0.379686	0.565935	0.100255	-0.111395	0.163974	0.172496	0.129601	0.096996	0.18
a2d_mean	-0.187700	1.000000	-0.976761	-0.908298	0.010513	0.222996	-0.192964	-0.170859	0.094696	0.195499	-0.0
a2d_std	0.379686	-0.976761	1.000000	0.973922	0.022193	-0.222135	0.207586	0.188264	-0.060944	-0.157794	0.07
a2l_range	0.565935	-0.908298	0.973922	1.000000	0.044539	-0.223800	0.221222	0.207453	-0.028156	-0.125853	0.10
a7d_max	0.100255	0.010513	0.022193	0.044539	1.000000	-0.127484	0.322978	0.454068	0.077046	0.277936	0.17
a7d_mean	-0.111395	0.222996	-0.222135	-0.223800	-0.127484	1.000000	-0.962195	-0.910083	-0.200717	-0.166550	-0.2
a7d_std	0.163974	-0.192964	0.207586	0.221222	0.322978	-0.962195	1.000000	0.985359	0.229747	0.242056	0.30
a7l_range	0.172496	-0.170859	0.188264	0.207453	0.454068	-0.910083	0.985359	1.000000	0.228868	0.267453	0.31
a4d_max	0.129601	0.094696	-0.060944	-0.028156	0.077046	-0.200717	0.229747	0.228868	1.000000	0.852552	0.89
a4d_mean	0.096996	0.195499	-0.157794	-0.125853	0.277936	-0.166550	0.242056	0.267453	0.852552	1.000000	0.80
a4d_std	0.189036	-0.032450	0.071098	0.104044	0.172314	-0.258298	0.301807	0.312119	0.894497	0.802666	1.00
a4l_range	0.291658	-0.391560	0.430994	0.448053	0.154870	-0.212284	0.218112	0.223580	0.197314	-0.078440	0.39

In [32]:
```
#Based on: a2d_std Vs a2d_mean: +ve corr for good. -ve corr for bad
#Possibly no improvement because of lack of signal in this selection.
#May need to cross validate
algo_str = "QuadraticDiscriminantAnalysis"
scols = ["a2l_range", "a2d_std", "a2d_mean", "a7d_std","a7l_range","a4d_max", "a4l_range", "a4d_mean"]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.79
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.53
Cross-val-score(precision)= 0.64
Cross-val-score(f1)       = 0.64
```

In [35]:
```
algo_str = "QuadraticDiscriminantAnalysis"
scols = tdf.columns[:-1]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.78
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.56
Cross-val-score(precision)= 0.65
Cross-val-score(f1)       = 0.65
```

OVERALL SUMMARY

Looks like need to use all Features derived from a2, a7 and a4

In [ ]:
```
```

Step 8: Overall Recommendations

In [2]:
```
import pandas as pd
import sys
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np
%matplotlib inline
cmap_bold = ListedColormap(['#00FF00','#FF0000'])
sys.path.append('../utils')
import DataAggregation as da
import AlgoUtils as au
```

In [3]:
```
algos_dd = {
    "LogisticRegression": {"C": 1e9},
    "LogisticRegressionB": {"C": 1e9, "class_weight":'balanced'},
    "KNeighborsClassifier": {"n_neighbors": 7},
    "LinearDiscriminantAnalysis": {},
    "QuadraticDiscriminantAnalysis": {},
    "SVC": {}
}

fcols = ["d_mean:d_std:d_max:l_range",
         "d_mean:d_std:l_range",
         "d_std:l_range",
         "l_range",
         "d_std",
         "d_max"]
algos_str = ["LogisticRegression",
             "LogisticRegressionB",
             "KNeighborsClassifier",
             "LinearDiscriminantAnalysis",
             "QuadraticDiscriminantAnalysis"]
```

In [4]:
```
a2 = da.GetFrames("../data/device_failure.csv", "a2")
a7 = da.GetFrames("../data/device_failure.csv", "a7")
a4 = da.GetFrames("../data/device_failure.csv", "a4", ldays=-30, lday_strict=False)
tdf = a2.df_sfeature.drop("failure", axis=1).join(a7.df_sfeature.drop("failure", axis=1)).join(a4.df_s
feature)
```

All models and recommendations need further validation at scale!

In [ ]:
```
```

Model 1: Definite Action Model

If this model detects failure, take action

Ofcourse still need to do validation at scale

See analysis in Step3_a2_analysis.ipynb, Step4_a7_analysis.ipynb for more info on why we are confident about this recommendation

In [6]:
```
algo_str = "QuadraticDiscriminantAnalysis"
scols = ["a2l_range", "a2d_std", "a2d_mean", "a2d_max",
         "a7l_range", "a7d_std", "a7d_mean", "a7d_max"]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

```
Cross-val-score(roc_auc) = 0.74
Cross-val-score(accuracy) = 0.94
Cross-val-score(recall)   = 0.46
Cross-val-score(precision)= 0.76
Cross-val-score(f1)       = 0.76
```

In [ ]:

In [ ]:

Model 2:

If Model 1 does not detect failure, but this model detects failure

Recommend Inspection of device

Do Hypothesis testing from field:

How many days to actual failure once this model detected fail

Refer to analysis in Step5_a4_analysis.ipynb for explanation

In [7]:
```
algo_str = "QuadraticDiscriminantAnalysis"
scols = tdf.columns[:-1]
analysisdf = au.do_clf_validate_new(tdf, algo_str,algos_dd[algo_str], scols, "failure")
```

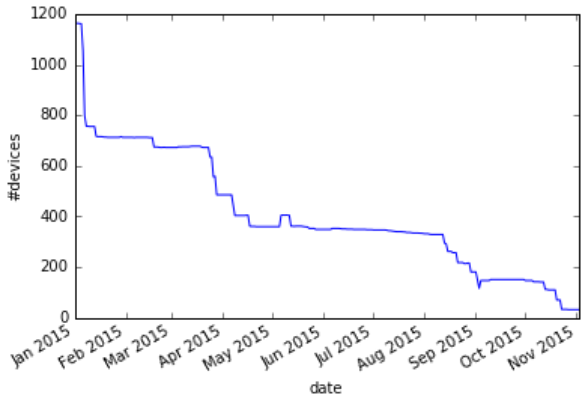
```
Cross-val-score(roc_auc) = 0.78
Cross-val-score(accuracy) = 0.93
Cross-val-score(recall)   = 0.56
Cross-val-score(precision)= 0.65
Cross-val-score(f1)       = 0.65
```

In [ ]:

Data Quality Improvement Recommendations

In [16]:
```
df = pd.read_csv("../data/device_failure.csv")
df.loc[:,'date'] = pd.to_datetime(df['date'])
df.groupby(["date"]).count()["device"].plot.line()
plt.ylabel("#devices")
```

Out[16]: <matplotlib.text.Text at 0x7f4fd5213850>



NumDevices/day that were observed --> points to systemic ingestion/sampling problem

This graph should remain relatively flat over time for same number of devices that are getting monitored

Refer to Step1_EDA.ipynb for more information

In [ ]:

Other Data Engineering related recommendations

Possible ways to store raw and transformed (log and log differentials)

i. Druid or some other timeseries databases

ii. Elastic / Solr (for quick exploration via Kibana/(Banana_Twigkit)

iii. orc/parquet in hdfs with daily partitions

Note in orc/parquet: Want to make sure device id be used as columns so that values for each device end up in columnar format. This makes aggregations/manipulations extremely fast.

In [ ]:

Thank You!

It was fun working on this dataset.

Need to be honest (to set your expectations)

This data challenge took me close to ~ 15 hours which includes submission time.

It took me good 4-6 hours or so to explore the data and just to get idea that I need to use logarithms and logarithm differentials as the values and ranges of devices were all over the place.

It was really when I saw the clear seperation of good vs bad devices in scatter plots that I thought I was in right path

I cannot wait to share these findings with my buddies over at HMS (https://www.hms-networks.com/about (https://www.hms-networks.com/about))

Also request you to read my article I just published (https://www.linkedin.com/pulse/what-fastest-sailboat-taught-me-data-science-viswanath-puttagunta (https://www.linkedin.com/pulse/what-fastest-sailboat-taught-me-data-science-viswanath-puttagunta))

I look forward to the interview. My last interview with Amazon in 2012 was fantastic!

In [ ]: