classes-4-differences

September 10, 2022

1 Classes 4 - Data classes round 2

Sections: 1) Introduction 2) Coding 3) Charts - visual inspection 4) Probabilities (not finished)

This notebook prepares the raw data from the 2018 sensor reference data set and uses it to construct the following random variable to analyzse how the difference between station control measurements evolve through time.

Theory and definition of random variable

Let (x_i, x_j) represent measurements from a given AWS and LCD station pair at a given point in time. Let $\overline{x_{i,j}}$ and $\sigma_{x_{i,j}}$ be the mean difference and standard deviation of this difference between aws and lcd measurements for all pairs of observations (x_i, x_j).

Define the discreet Random Variable X, where $X =$ (the number of observations that exceed 2x the standard deviation) for a given time period t_k . This can be written as $g(x_i, x_j)$ for all (x_i, x_j) in a given time t_k.

$$g(x_i, x_j) = 1 \; if \; \frac{x_i - x_j - \overline{x_{i,j}}}{\sigma_{x_{i,j}}} > 2$$

Therefore the random variable can be written as:

$$X_k = \sum g(x_i, x_j) \;\; \forall \; (x_i, x_j) \in t_k$$

Concretely, the study period runs from May 15th - September 15th. There are thus 123 study days. There are 6 readings per hour. Therefore there are $6 * 123 = 738$ observation for each hour of the day during the survey period. Therefore, the range of $X_k = 0 < i < 738 \; \forall \; i \; \in I$ for every time period $k \; \in \; K$ where K is the set of all time period groupings.

1.1 Coding

[1]:
```
# Import statements
import pandas as pd
import os
import datetime as dt
from datetime import timedelta
import seaborn as sns
```

```
from matplotlib import pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.ticker import AutoLocator
import itertools
from itertools import product
import numpy as np
import csv
import operator

# Directories for reading and writing data
thedir = os.getcwd()
writedir = os.path.abspath(os.path.join(os.path.dirname(thedir), '..', 'data/
 ↪interim'))
extdir = os.path.abspath(os.path.join(os.path.dirname(thedir), '..', 'data/
 ↪external'))
figdir = os.path.abspath(os.path.join(os.path.dirname(thedir), '..', 'figures'))

# load data (reference data set of lcd and aws stations from 2018)
data_test = pd.read_csv(F'{extdir}/ref_2018.csv')
data_ = data_test[['Date_time_CET','BOLL_LCD_TEMP',
       'BOLL_AWS_TEMP', 'BOLL_PRECIP', 'BOLL_RADIATION_GLOBAL',
       'BOLL_WIND_SPEED_mean', 'ZOLL_STEVENSON_LCD_TEMP', 'ZOLL_2m_LCD_TEMP',
       'ZOLL_3m_LCD_TEMP', 'ZOLL_AWS_TEMP', 'ZOLL_RADIATION_GLOBAL',
       'ZOLL_SUNSHINE', 'ZOLL_PRECIP', 'ZOLL_WIND_SPEED_MEAN', 'AFU_LCD_TEMP',
       'AFU_AWS_TEMP', 'AFU_WIND_SPEED_MEAN', 'AFU_RADIATION_GLOBAL']]
data_.set_index('Date_time_CET',inplace=True)

# dictionaries for data
names = ['BOLL_LCD_TEMP','BOLL_AWS_TEMP', 'BOLL_PRECIP',␣
 ↪'BOLL_RADIATION_GLOBAL','BOLL_WIND_SPEED_mean', 'ZOLL_STEVENSON_LCD_TEMP',
        'ZOLL_2m_LCD_TEMP','ZOLL_3m_LCD_TEMP', 'ZOLL_AWS_TEMP',␣
 ↪'ZOLL_RADIATION_GLOBAL','ZOLL_SUNSHINE', 'ZOLL_PRECIP',␣
 ↪'ZOLL_WIND_SPEED_MEAN',
        'AFU_LCD_TEMP', 'AFU_AWS_TEMP', 'AFU_WIND_SPEED_MEAN',␣
 ↪'AFU_RADIATION_GLOBAL']
period_keys = {'hour':'%H','month':'%m','year':'%Y','day':'%j'}
qty = ['temp','wind','rad','sun','prec']
station = ['bol','zol','afu']
key = period_keys['day']
```

[2]:
```
# define class (update value func isn't necessary...)

class Thing:
    def __init__(self, value, index, header):
        self.name = header
        self.time = dt.datetime.strptime(index,'%d.%m.%Y %H:%M')
        self.value = value
```

```
        self.station = header[:3].lower()
        if 'LCD' in header:
            self.stype = 'lcd'
        else:
            self.stype = 'aws'
        if 'TEMP' in header:
            self.qty = 'temp'
        elif 'SUN' in header:
            self.qty = 'sun'
        elif 'RADI' in header:
            self.qty = 'rad'
        elif 'WIND' in header:
            self.qty = 'wind'
        elif 'PREC' in header:
            self.qty = 'prec'

    def get_period(self, strfkey):
        return dt.datetime.strftime(self.time,strfkey)

    def update_value(self,value):
        self.value = value
```

1.1.1 Create list of objects

[3]:
```
cols = data_.columns
mylist = []
for index,rows in data_.iterrows():
        mylist.append([Thing(data_.at[index,index_c],index,index_c) for index_c␣
 ↪in cols])
ad = [x for y in mylist for x in y]
ad = sorted(ad, key=operator.attrgetter('time'))
times = list(set([x.time for x in ad]))
times = sorted(times)
```

1.1.2 Fix incorrect data

Bad data is keyed as -9999, following code replaces the bad valeus with the mean from the previous and next hour (excluding any potential bad values that also occur in that hour).

[4]:
```
wrong = [x for x in ad if x.value < -50]
for i in wrong:
    lower = [x.value for x in ad if (x.name == i.name) & (i.time -␣
 ↪timedelta(hours=1) < x.time < i.time) & (x.value > -50)]
    higher = [x.value for x in ad if (x.name == i.name) & (i.time < x.time < i.
 ↪time + timedelta(hours=1)) & (x.value > -50)]
```

```
    new_val = (sum(lower) + sum(higher))/(len(lower) + len(higher))
    i.update_value(new_val)
```

1.1.3 Get values

Get the differences for each lcd-aws pair.

[5]:
```
# get values from Thing class (Sensor reading)
aws = [x.value for x in ad if x.name == 'BOLL_AWS_TEMP']
lcd = [x.value for x in ad if x.name == 'BOLL_LCD_TEMP']
# subtract aws - lcd for readings
diff = [x - y for x, y in zip(aws, lcd)]
# turn into pd dataframe, turn time pd dataframe
diff = pd.DataFrame(diff)
md = pd.DataFrame(times)
# use cumsum function to create a new cumsum df
cumsum = diff.cumsum()
# concat dfs together
bvals = pd.concat([cumsum,diff,md],axis = 1)
# reindex columns and rows
bvals.columns = ['Boll_sum','Boll_diff','time']
bvals.set_index('time',inplace=True,drop=True)

## same as above for each lcd-aws sensor pair. Should be turned into a loop but␣
 ↪I got stuck making it...

aws = [x.value for x in ad if x.name == 'AFU_AWS_TEMP']
lcd = [x.value for x in ad if x.name == 'AFU_LCD_TEMP']
diff = [x - y for x, y in zip(aws, lcd)]
diff = pd.DataFrame(diff)
md = pd.DataFrame(times)
cumsum = diff.cumsum()
avals = pd.concat([cumsum,diff,md],axis = 1)
avals.columns = ['afu_sum','afu_diff','time']
avals.set_index('time',inplace=True,drop=True)

aws = [x.value for x in ad if x.name == 'ZOLL_AWS_TEMP']
lcd = [x.value for x in ad if x.name == 'ZOLL_STEVENSON_LCD_TEMP']
diff = [x - y for x, y in zip(aws, lcd)]
diff = pd.DataFrame(diff)
md = pd.DataFrame(times)
cumsum = diff.cumsum()
zvals = pd.concat([cumsum,diff,md],axis = 1)
zvals.columns = ['zoll_sum_s','zoll_diff_s','time']
zvals.set_index('time',inplace=True,drop=True)
```

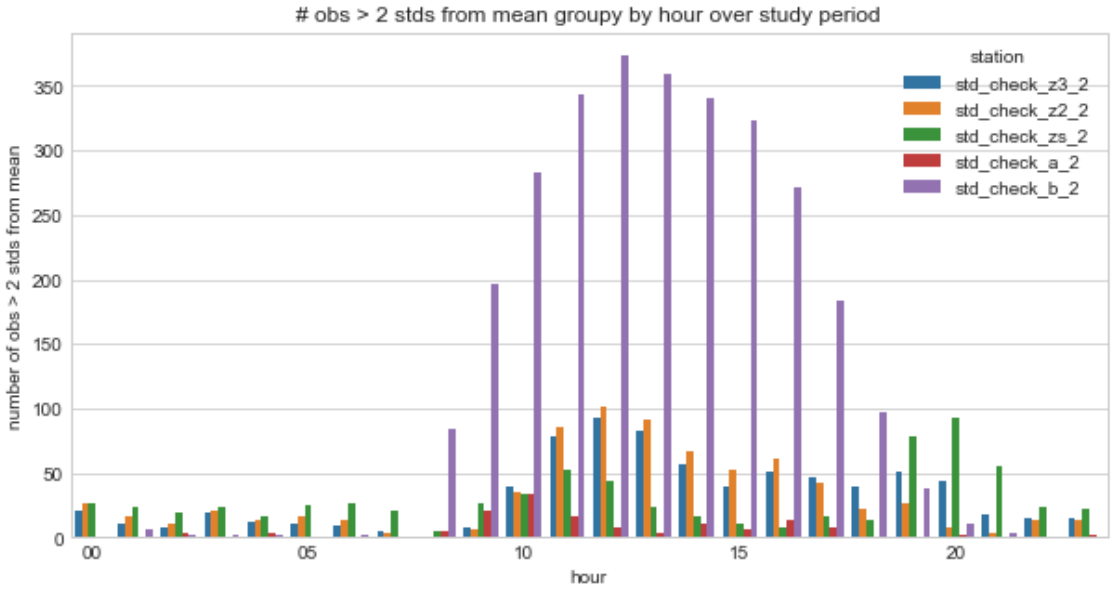
```
aws = [x.value for x in ad if x.name == 'ZOLL_AWS_TEMP']
lcd = [x.value for x in ad if x.name == 'ZOLL_2m_LCD_TEMP']
diff = [x - y for x, y in zip(aws, lcd)]
diff = pd.DataFrame(diff)
md = pd.DataFrame(times)
cumsum = diff.cumsum()
zvals2 = pd.concat([cumsum,diff,md],axis = 1)
zvals2.columns = ['zoll_sum_2','zoll_diff_2','time']
zvals2.set_index('time',inplace=True,drop=True)

aws = [x.value for x in ad if x.name == 'ZOLL_AWS_TEMP']
lcd = [x.value for x in ad if x.name == 'ZOLL_3m_LCD_TEMP']
diff = [x - y for x, y in zip(aws, lcd)]
diff = pd.DataFrame(diff)
md = pd.DataFrame(times)
cumsum = diff.cumsum()
zvals3 = pd.concat([cumsum,diff,md],axis = 1)
zvals3.columns = ['zoll_sum_3m','zoll_diff_3m','time']
zvals3.set_index('time',inplace=True,drop=True)

# concat the df from each sensor into 1
cumsums = [zvals3,zvals2,zvals,bvals,avals]
df = pd.concat(cumsums,axis = 1)

# get standard deviation and mean for all reading differences
stds = df.std()
means = df.mean()

a = [stds[x] for x in stds.index if 'diff' in x]
b = [x for x in stds.index if 'diff' in x]
std = dict(zip(b,a))

a = [means[x] for x in means.index if 'diff' in x]
b = [x for x in means.index if 'diff' in x]
mean = dict(zip(b,a))

# get the difference between mean and observation and divide by standard␣
 ↪deviation.
def check(x,col):
    return abs((x - mean[col]) * (1/std[col]))

df['std_check_z3'] = df.zoll_diff_3m.apply(lambda x: check(x,'zoll_diff_3m'))
df['std_check_z2'] = df.zoll_diff_2.apply(lambda x: check(x,'zoll_diff_2'))
df['std_check_zs'] = df.zoll_diff_s.apply(lambda x: check(x,'zoll_diff_s'))
df['std_check_a'] = df.Boll_diff.apply(lambda x: check(x,'afu_diff'))
df['std_check_b'] = df.afu_diff.apply(lambda x: check(x,'Boll_diff'))
```

```
# check if the observation is more than 2 standard deviations away from the mean
def check_2(x):
    if x > 2:
        return True
    else:
        return False

col = [x for x in df.columns if 'std' in x]

for x in col:
    df[F'{x}_2'] = df[x].apply(lambda x: check_2(x))
df.reset_index(inplace=True,drop=False)

# get time periods to groupby for analysis
period_keys = {'hour':'%H','month':'%m','year':'%Y','day':'%j'}

df['hours'] = df.time.apply(lambda x: x.strftime('%H'))
df['days'] =  df.time.apply(lambda x: x.strftime('%j'))

# group by time period and count the number of obs that exceed 2 standard␣
 ↪deviations in each time period.
mylist = []
cols  = [x for x in df.columns if ('_2' in x) & ('check' in x)]
for i in cols:
    df_ = df.groupby(['days','hours'])[i].sum()
    df_.columns = [i]
    # re-index resulting multilevel index so day,hour index key becomes␣
 ↪'dayhor' eg. day 105, hour 04 becomes 10504
    df_.index = df_.index.map(''.join)
    mylist.append([i,df_])
```

1.2 Charts

[6]:
```
mylist = []
cols  = [x for x in df.columns if ('_2' in x) & ('check' in x)]
for i in cols:
    df_ = df.groupby(['hours'])[i].sum()
    df_.columns = [i]
    df_.index = df_.index.map(''.join)
    mylist.append(df_)
data_ = pd.concat(mylist,axis = 1)
data_.reset_index(inplace=True,drop=False)
melted = data_.melt('hours',var_name = 'station', value_name = '# obs > 2 stds␣
 ↪from mean', )
```

[7]:
```
plt.rcParams.update(plt.rcParamsDefault)
%matplotlib inline
plt.style.use('seaborn-whitegrid')

fig,axs = plt.subplots(figsize= (10,5))
sns.barplot(data=melted,x = melted.hours,y = '# obs > 2 stds from mean',hue =␣
 ↪'station')
axs.set_title('# obs > 2 stds from mean groupy by hour over study period')
axs.set_xlabel('hour')
axs.set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs.xaxis.set_major_locator(loc)
```

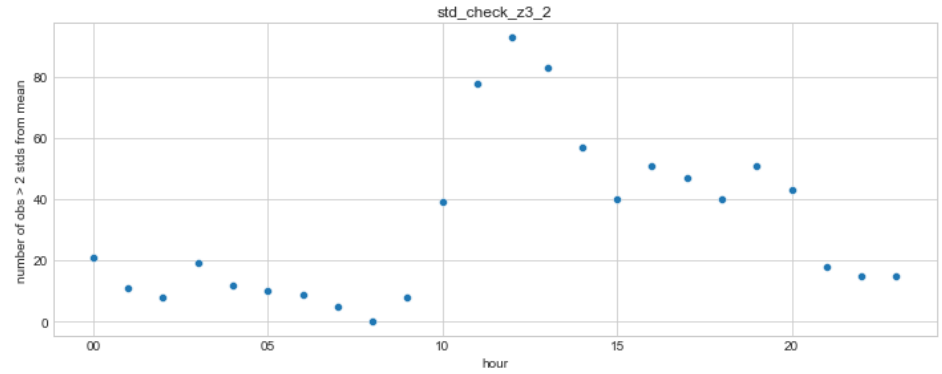


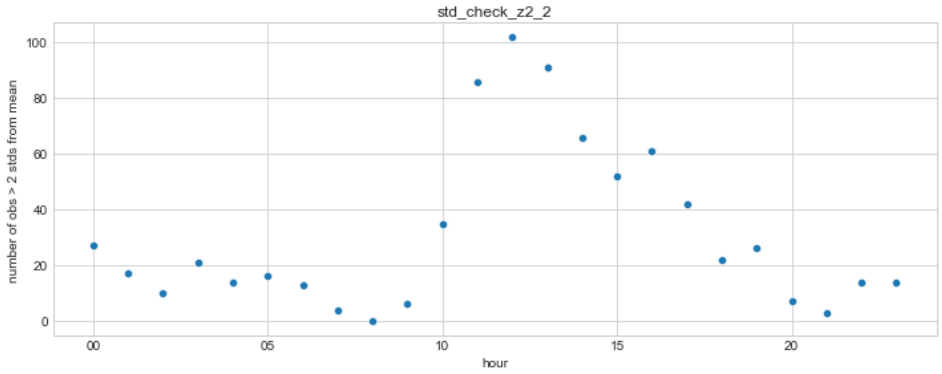
1.3 Scatter Plots - individual station pairs

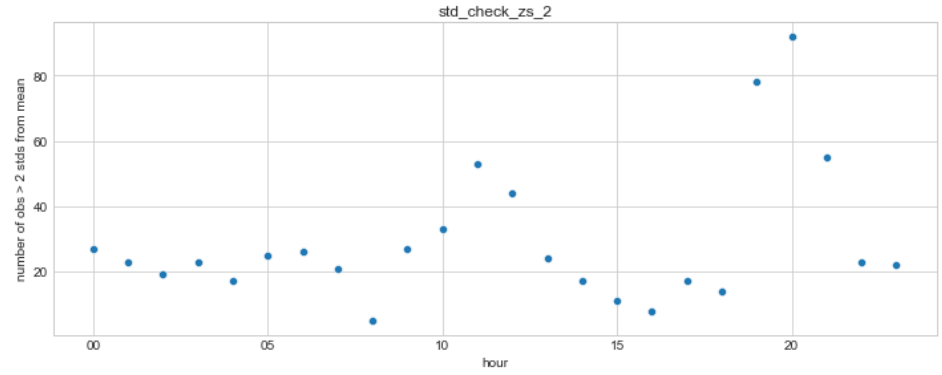
Hourly and daily differences from the mean that exceed 2 * standard deviation are counted and returned for each station. For clarity, each station is graphed separately.

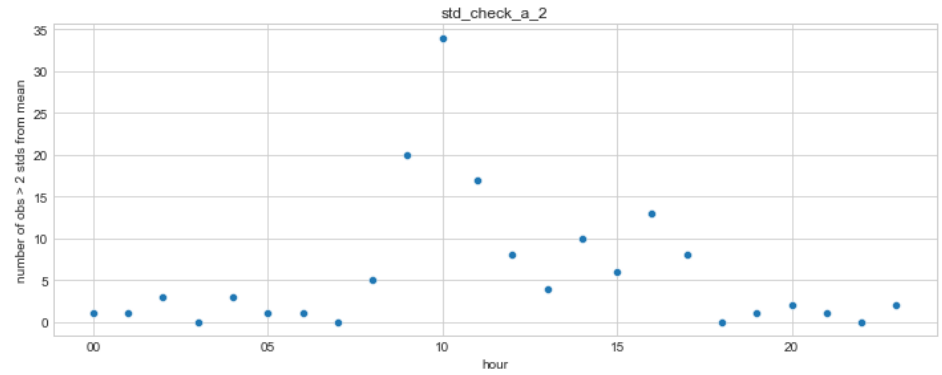
[8]:
```
mylist = []
cols  = [x for x in df.columns if ('_2' in x) & ('check' in x)]
for i in cols:
    df_ = df.groupby(['hours'])[i].sum()
    df_.columns = [i]
    df_.index = df_.index.map(''.join)
    mylist.append([i,df_])
```

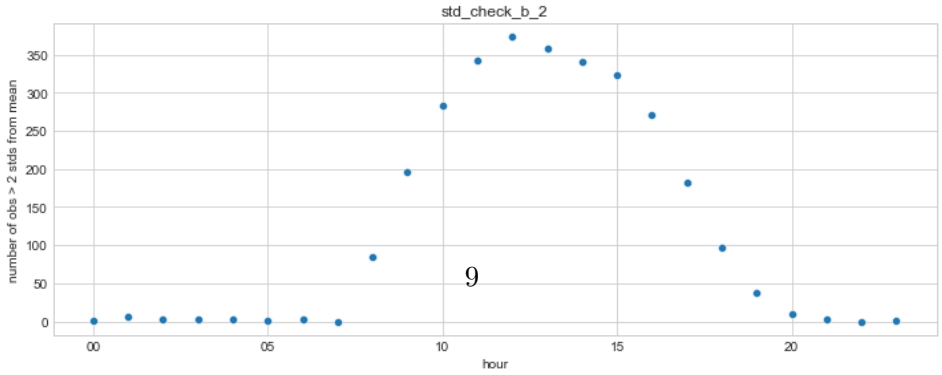
```
plt.rcParams.update(plt.rcParamsDefault)
%matplotlib inline
plt.style.use('seaborn-whitegrid')

fig,axs = plt.subplots(5,1,figsize= (10,20))
data_ = mylist[0][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[0])
axs[0].set_title(mylist[0][0])
axs[0].set_xlabel('hour')
axs[0].set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs[0].xaxis.set_major_locator(loc)

data_ = mylist[1][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[1])
axs[1].set_title(mylist[1][0])
axs[1].set_xlabel('hour')
axs[1].set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs[1].xaxis.set_major_locator(loc)

data_ = mylist[2][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[2])
axs[2].set_title(mylist[2][0])
axs[2].set_xlabel('hour')
axs[2].set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs[2].xaxis.set_major_locator(loc)

data_ = mylist[3][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[3])
axs[3].set_title(mylist[3][0])
axs[3].set_xlabel('hour')
axs[3].set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs[3].xaxis.set_major_locator(loc)

data_ = mylist[4][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[4])
axs[4].set_title(mylist[4][0])
axs[4].set_xlabel('hour')
axs[4].set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs[4].xaxis.set_major_locator(loc)
plt.tight_layout()
plt.savefig(F'{figdir}/obs_stds_above_mean_hours_2018.png')
```


std_check_z3_2
number of obs > 2 stds from mean
0
20
40
60
80
00
05
10
15
20
hour


std_check_z2_2
number of obs > 2 stds from mean
0
20
40
60
80
100
00
05
10
15
20
hour


std_check_zs_2
number of obs > 2 stds from mean
20
40
60
80
00
05
10
15
20
hour

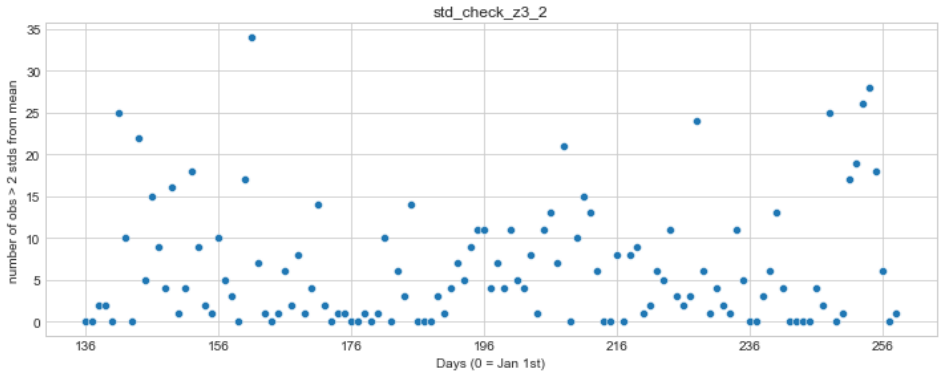

std_check_a_2
number of obs > 2 stds from mean
0
5
10
15
20
25
30
35
00
05
10
15
20
hour

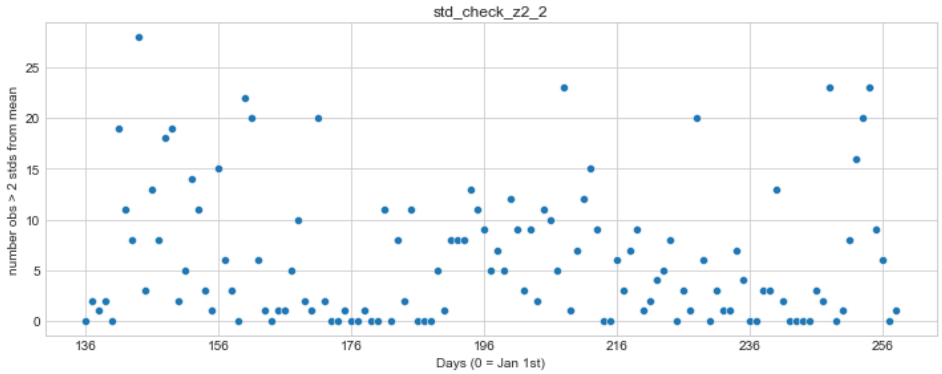

std_check_b_2
number of obs > 2 stds from mean
0
50
100
150
200
250
300
350
00
05
10
15
20
hour

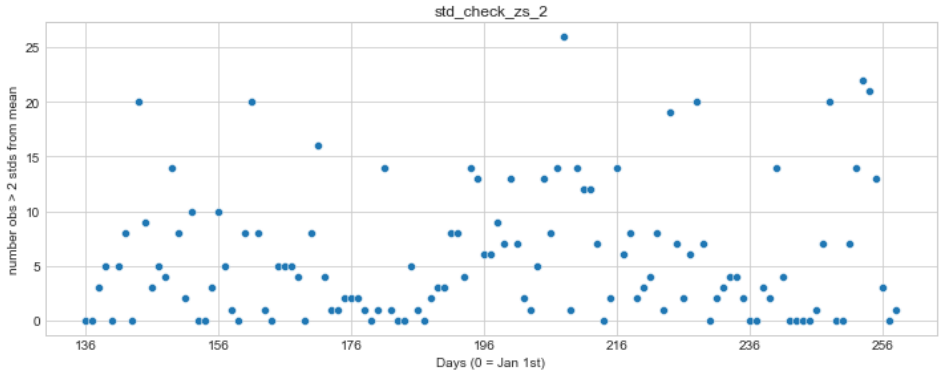
Daily values (0 is January1st) are checked to see if the incidence of extreme values changes over the measurement period (May15th to September 15th)

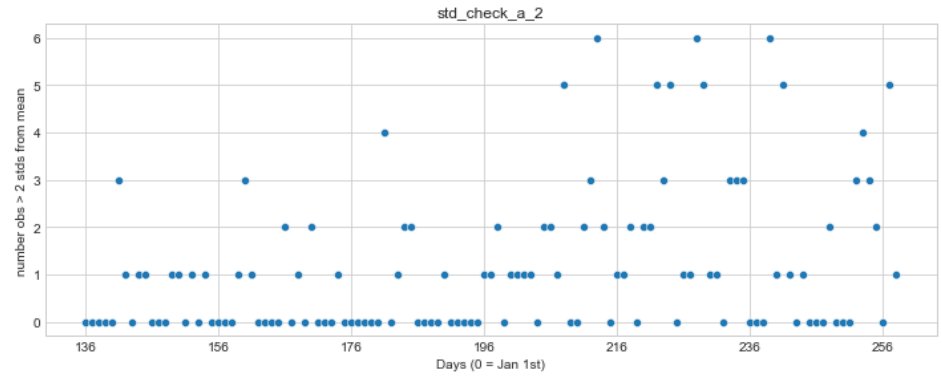
[9]:
```
mylist = []
cols  = [x for x in df.columns if ('_2' in x) & ('check' in x)]
for i in cols:
    df_ = df.groupby(['days'])[i].sum()
    df_.columns = [i]
    df_.index = df_.index.map(''.join)
    mylist.append([i,df_])

plt.rcParams.update(plt.rcParamsDefault)
%matplotlib inline
plt.style.use('seaborn-whitegrid')

fig,axs = plt.subplots(5,1,figsize= (10,20))
data_ = mylist[0][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[0])
axs[0].set_title(mylist[0][0])
axs[0].set_xlabel('Days (0 = Jan 1st)')
axs[0].set_ylabel('number of obs > 2 stds from mean')
loc = AutoLocator()
axs[0].xaxis.set_major_locator(loc)

data_ = mylist[1][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[1])
axs[1].set_title(mylist[1][0])
axs[1].set_xlabel('Days (0 = Jan 1st)')
axs[1].set_ylabel('number obs > 2 stds from mean')
loc = AutoLocator()
axs[1].xaxis.set_major_locator(loc)

data_ = mylist[2][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[2])
axs[2].set_title(mylist[2][0])
axs[2].set_xlabel('Days (0 = Jan 1st)')
axs[2].set_ylabel('number obs > 2 stds from mean')
loc = AutoLocator()
axs[2].xaxis.set_major_locator(loc)

data_ = mylist[3][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[3])
axs[3].set_title(mylist[3][0])
axs[3].set_xlabel('Days (0 = Jan 1st)')
axs[3].set_ylabel('number obs > 2 stds from mean')
```

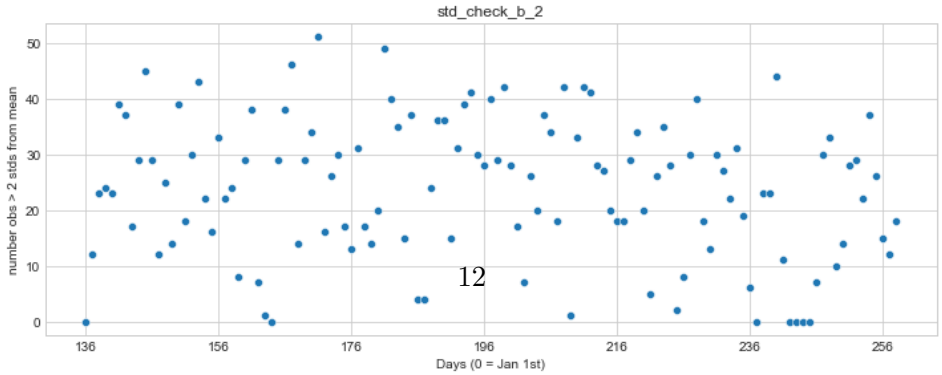
```
loc = AutoLocator()
axs[3].xaxis.set_major_locator(loc)

data_ = mylist[4][1]
sns.scatterplot(data=data_,x = data_.index,y = data_,ax = axs[4])
axs[4].set_title(mylist[4][0])
axs[4].set_xlabel('Days (0 = Jan 1st)')
axs[4].set_ylabel('number obs > 2 stds from mean')
loc = AutoLocator()
axs[4].xaxis.set_major_locator(loc)
plt.tight_layout()
plt.savefig(F'{figdir}/obs_stds_above_mean_days_2018.png')
```


std_check_z3_2
number of obs > 2 stds from mean
Days (0 = Jan 1st)


std_check_z2_2
number obs > 2 stds from mean
Days (0 = Jan 1st)


std_check_zs_2
number obs > 2 stds from mean
Days (0 = Jan 1st)


std_check_a_2
number obs > 2 stds from mean
Days (0 = Jan 1st)


std_check_b_2
number obs > 2 stds from mean
Days (0 = Jan 1st)

1.4 Probability Work

NOTE: missing values identified and removed in this section. In the next update these values will be removed in a separate notebook.

[10]:
```
# correct missing values
rsq = df[['time','zoll_diff_3m',␣
 ↪'zoll_diff_2','zoll_diff_s','Boll_diff','afu_diff']]
rsq = pd.concat([rsq,dfr],axis = 1)
a = rsq.isna()
b = a[(a.zoll_diff_3m == False) & (a.zoll_diff_2 == False) & (a.zoll_diff_s ==␣
 ↪False) & (a.Boll_diff == False) & (a.afu_diff == False)]
indy = b.index
rsq_ = rsq[rsq.index.isin(indy)]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      1 # correct missing values
      2 rsq = df[['time','zoll_diff_3m',␣
 ↪'zoll_diff_2','zoll_diff_s','Boll_diff','afu_diff']]
----> 3 rsq = pd.concat([rsq,dfr],axis = 1)
      4 a = rsq.isna()
      5 b = a[(a.zoll_diff_3m == False) & (a.zoll_diff_2 == False) & (a.
 ↪zoll_diff_s == False) & (a.Boll_diff == False) & (a.afu_diff == False)]

NameError: name 'dfr' is not defined
```

[ ]:
```
```