hydropy_tutorial

May 5, 2015

```
In [1]: %matplotlib inline
```

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set_style('whitegrid')

        from IPython.display import HTML
```

```
In [3]: flowdata = pd.read_pickle("./data/FlowData")
        raindata = pd.read_pickle("./data/RainData")
```

1 Hydropy-package

Go to the hydropy website for more information on the package and the code on Github.

```
In [4]: #Loading the hydropy package
        import hydropy as hp
```

We have a Dataframe with river discharge at different locations in the Maarkebeek basin (Belgium):

```
In [5]: HTML('<iframe src=http://biomath.ugent.be/~stvhoey/maarkebeek_data/ width=700 height=350></ifra
```

```
Out[5]: <IPython.core.display.HTML object>
```

Data downloaded from http://www.waterinfo.be/, made available by the Flemish Environmental Agency (VMM).

```
In [6]: flowdata.head()
```

```
Out[6]:                      L06_347  LS06_347  LS06_348  LS06_34C  LS06_34D  \
        Time
        2008-01-01 00:15:00    0.229     0.229     0.021     0.122     0.014
        2008-01-01 00:30:00    0.229     0.229     0.021     0.122     0.014
        2008-01-01 00:45:00    0.229     0.229     0.021     0.122     0.014
        2008-01-01 01:00:00    0.229     0.229     0.021     0.122     0.015
        2008-01-01 01:15:00    0.229     0.229     0.021     0.122     0.015

                             LS06_34E  LS06_34G
        Time
        2008-01-01 00:15:00     0.030       NaN
        2008-01-01 00:30:00     0.029       NaN
        2008-01-01 00:45:00     0.029       NaN
        2008-01-01 01:00:00     0.029       NaN
        2008-01-01 01:15:00     0.029       NaN
```

```
In [7]: flowdata.tail()
```

Out[7]:

Time	L06_347	LS06_347	LS06_348	LS06_34C	LS06_34D	LS06_34E	LS06_34G
2013-01-01 23:00:00	0.883	0.883	0.075	NaN	0.119	0.034	NaN
2013-01-01 23:15:00	0.875	0.875	0.075	NaN	0.118	0.034	NaN
2013-01-01 23:30:00	0.872	0.872	0.074	NaN	0.119	0.034	NaN
2013-01-01 23:45:00	0.873	0.873	0.075	NaN	0.116	0.035	NaN
2013-01-02 00:00:00	0.860	0.860	0.075	NaN	0.114	0.036	NaN

```
In [8]: print len(flowdata), 'records', 'from', flowdata.index[0], 'till', flowdata.index[-1]
```

```
175488 records from 2008-01-01 00:15:00 till 2013-01-02 00:00:00
```

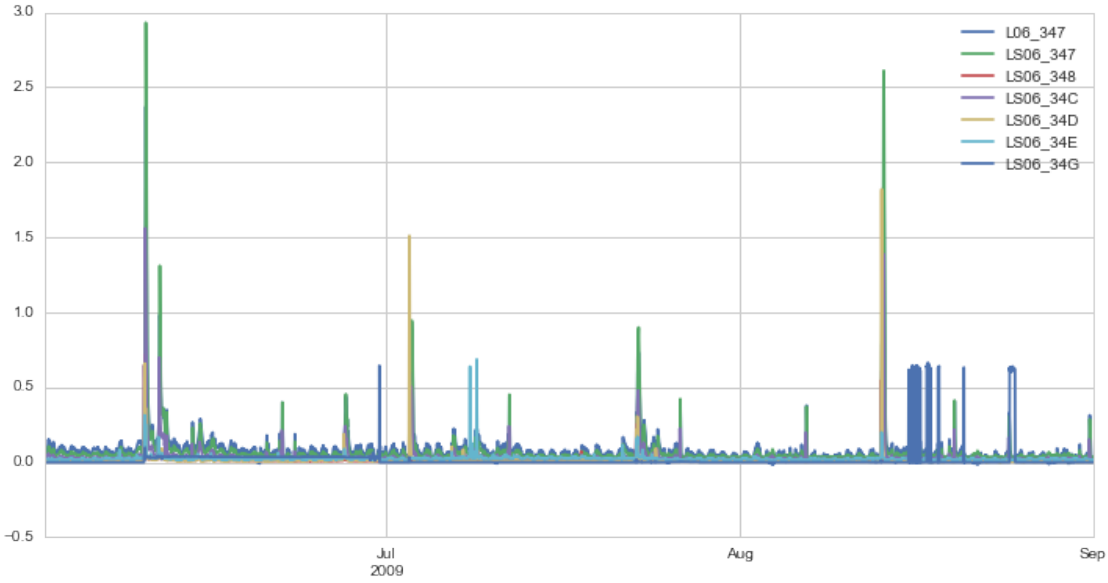
Converting the dataframe to a hydropy time series datatype, provides extra functionalities:

```
In [9]: myflowserie = hp.HydroAnalysis(flowdata)
```

1.0.1 Select the summer of 2009:

```
In [10]: myflowserie.get_year('2009').get_season('summer').plot(figsize=(12,6))
```

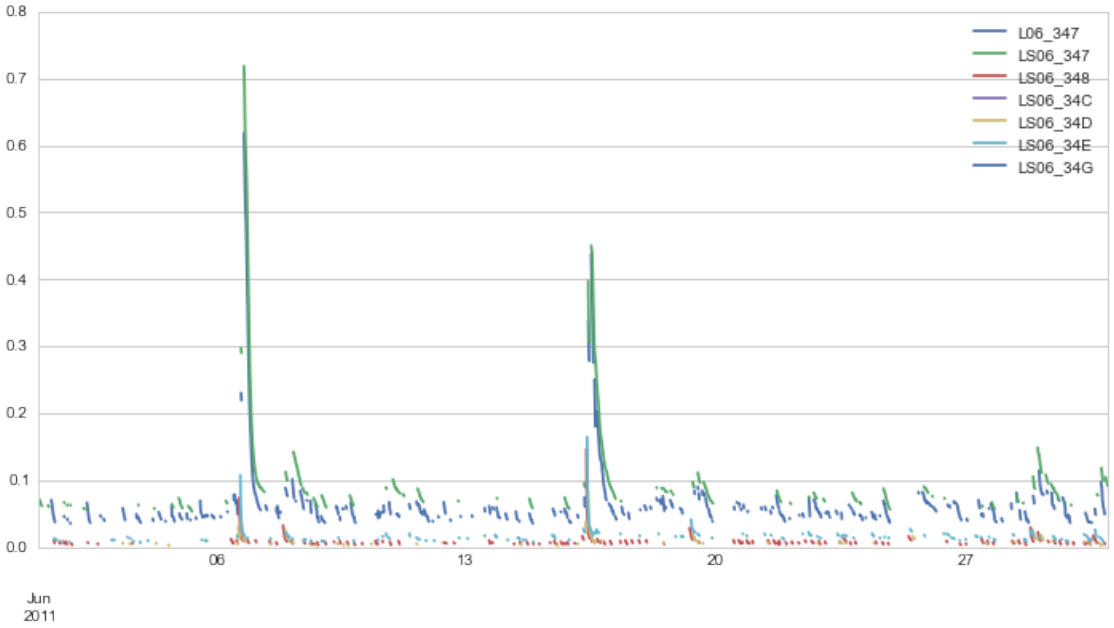
```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe9306458d0>
```



1.0.2 Select only the recession periods of the discharges (catchment drying) in June 2011:

```
In [11]: myflowserie.get_year('2011').get_month("Jun").get_recess().plot(figsize=(12,6))
```

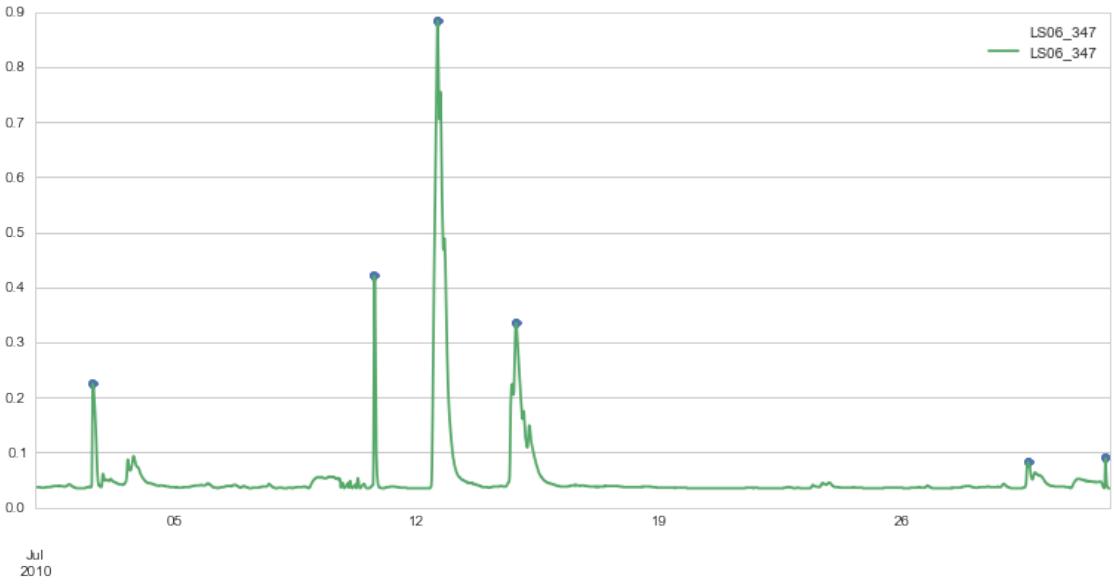
```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe92e43fed0>
```



1.0.3 Peak values above the 90th percentile for the station LS06_347 in July 2010:

```
In [12]: fig, ax = plt.subplots(figsize=(13, 6))
         myflowserie['LS06_347'].get_year('2010').get_month("Jul").get_highpeaks(150, above_percentile=
         myflowserie['LS06_347'].get_year('2010').get_month("Jul").plot(ax=ax)
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe92e07b0d0>
```

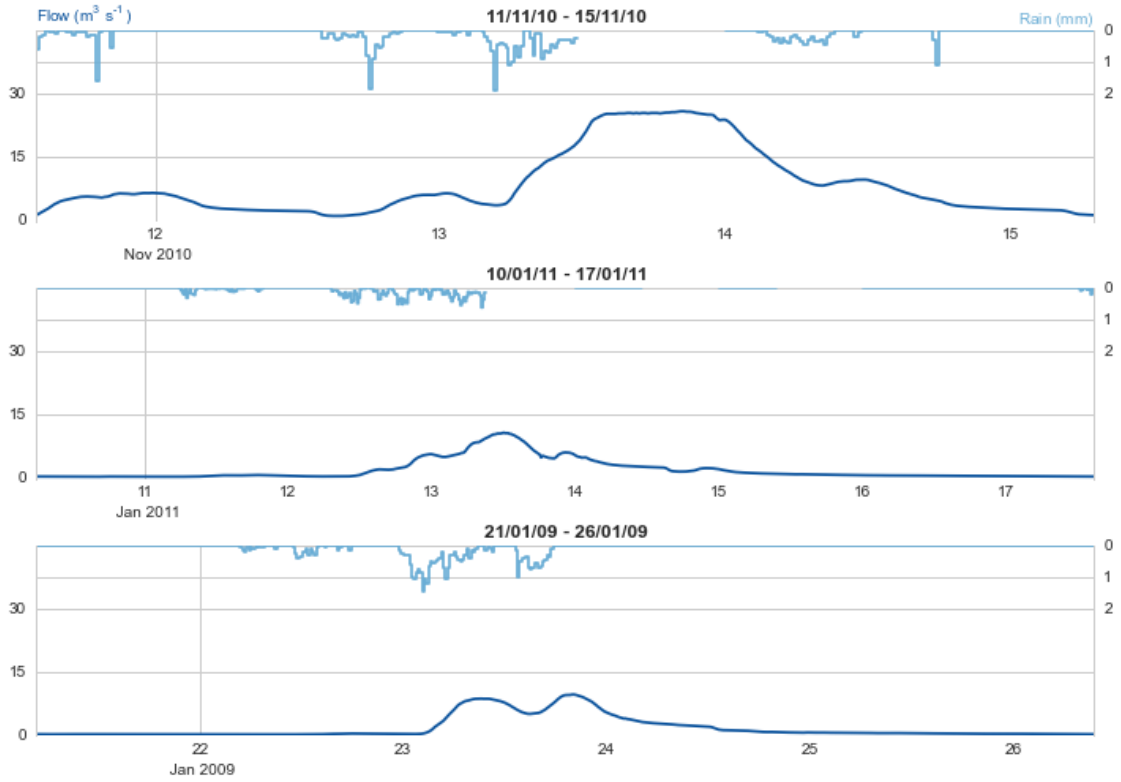


1.1 Select storms and make plot

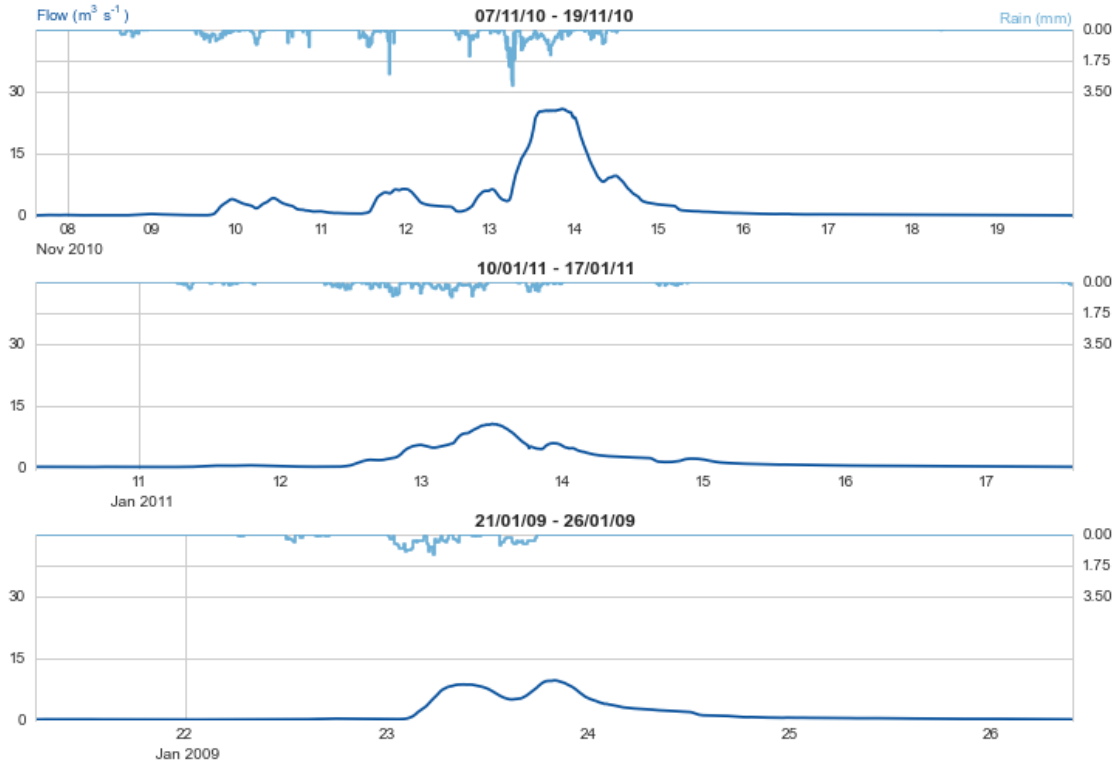
```
In [13]: raindata.columns
```

```
Out[13]: Index([u'P05_019', u'P05_038', u'P05_039', u'P06_014', u'P06_040', u'P07_006', u'P07_021', u'P07
```

```
In [14]: storms = myflowserie.derive_storms(raindata['P05_019'], 'LS06_347',
                                            number_of_storms=3, drywindow=50,
                                            makeplot=True)
```




```
In [30]: storms = myflowserie.derive_storms(raindata['P06_014'], 'LS06_347',
                                            number_of_storms=3, drywindow=96,
                                            makeplot=True)
```



1.2 Season averages (Pandas!)

```
In [16]: myflowserie.data.groupby('season').mean()
```

Out[16]:

season	L06_347	LS06_347	LS06_348	LS06_34C	LS06_34D	LS06_34E	LS06_34G
Autumn	0.238284	0.246378	0.022630	0.182139	0.026797	0.026666	0.017467
Spring	0.225966	0.239476	0.023696	0.144322	0.026558	0.029317	0.055004
Summer	0.097924	0.094836	0.012179	0.048987	0.012076	0.023207	0.136304
Winter	0.474153	0.480252	0.044169	0.219861	0.055192	0.034974	0.034519

1.3 Goals

- Use the power of Python **Pandas**
- Provide **domain specific** (hydrological) functionalities
- Provide **intuitive** interface for hydrological time series (main focus on flow series)
- Combine different earlier written loose functionalities in **package**
- Independent, but useful in global **Phd**-developed framework: enables the user to quickly look at different properties of model behaviour

1.4 Where?

- Code : https://github.com/stijnvanhoey/hydropy –> **Fork** and contribute
- Website : https://stijnvanhoey.github.io/hydropy/

1.4.1 How to start?

1. Fork the github repo
2. Get the code on your computer

   ```
   git clone https://github.com/yourname/hydropy
   ```

3. Run the python setup script (install as development package):

   ```
   python setup.py develop
   ```

4. Improve implementation, add functionalities,...
   - Make a new branch
   - Make improvements on this branch
   - push the branch towards the repo and perform a push request

In [ ]:

1.5 Functionalities extended

In [17]:
```
import hydropy as hp
flowdata = pd.read_pickle("./data/FlowData")
raindata = pd.read_pickle("./data/RainData")
myflowserie = hp.HydroAnalysis(flowdata)
```

1.5.1 Forwarding Pandas functionalities

In [18]:
```
# Data inspection
myflowserie.summary() #head(), tail(),
```

Out[18]:

	L06_347	LS06_347	LS06_348	LS06_34C
count	175465.000000	172153.000000	163312.000000	104486.000000
mean	0.258319	0.264058	0.025398	0.148447
std	0.730894	0.732851	0.047489	0.444993
min	-0.020000	-0.019000	0.001000	-0.010000
25%	0.059000	0.061000	0.009000	0.030000
50%	0.107000	0.116000	0.013000	0.065000
75%	0.240000	0.244000	0.025000	0.137000
max	25.900000	25.900000	1.400000	13.800000

	LS06_34D	LS06_34E	LS06_34G
count	174054.000000	171812.000000	93985.000000
mean	0.030074	0.028579	0.065588
std	0.079334	0.027651	0.156632
min	0.000000	0.002000	0.000000
25%	0.008000	0.017000	0.000000
50%	0.014000	0.025000	0.000000
75%	0.028000	0.033000	0.001000
max	3.720000	1.650000	0.691000

In [19]:
```
# Resampling frequencies
temp1 = myflowserie.frequency_resample('7D', 'mean')  # 7 day means
temp1.head()
```

Out[19]:

Time	L06_347	LS06_347	LS06_348	LS06_34C	LS06_34D	LS06_34E	LS06_34G	season
2008-01-01 00:15:00	0.407740	0.407740	0.045302	0.218046	0.045960	0.039046	0.000877	Winter
2008-01-08 00:15:00	0.463927	0.463927	0.046653	0.247088	0.052054	0.043190	0.000057	Winter
2008-01-15 00:15:00	0.473644	0.473644	0.046685	0.252646	0.058049	0.041683	0.000124	Winter
2008-01-22 00:15:00	0.267629	0.267629	0.026110	0.142237	0.023388	0.032665	0.000004	Winter
2008-01-29 00:15:00	0.278516	0.278516	0.027170	0.148635	0.024737	0.035676	0.000009	Winter

In [20]:
```
temp2 = myflowserie.frequency_resample("M", "max")  # Monthly maxima
temp2.head()
```

Out[20]:

Time	L06_347	LS06_347	LS06_348	LS06_34C	LS06_34D	LS06_34E	LS06_34G	season
2008-01-31	1.730	1.730	0.312	0.924	0.452	0.164	0.006	Winter
2008-02-29	0.747	0.747	0.274	0.398	0.218	0.168	0.691	Winter
2008-03-31	7.430	7.430	0.879	3.960	1.260	0.829	0.018	Spring
2008-04-30	1.010	1.010	0.160	0.541	0.216	0.133	0.001	Spring
2008-05-31	0.427	0.427	0.075	0.228	0.069	0.106	0.691	Spring

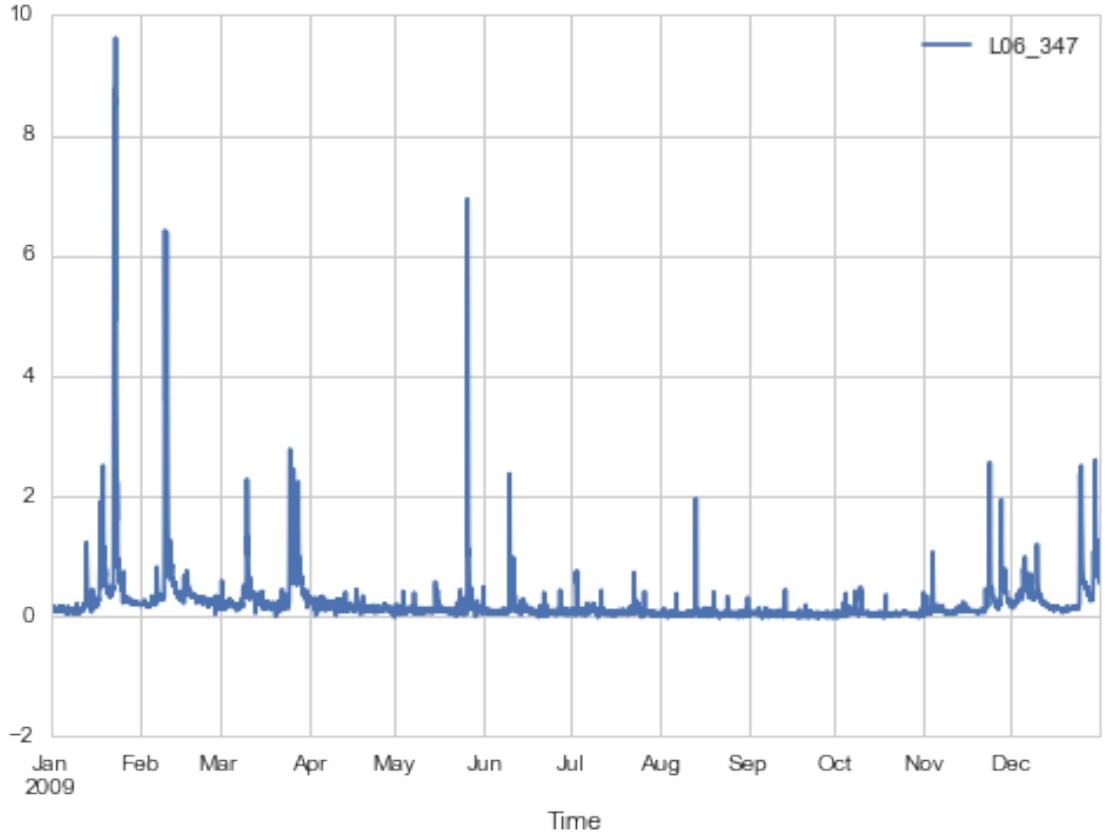
In [21]:
```
temp3 = myflowserie.frequency_resample("A", 'sum')  # Yearly sums
temp3.head(6)
```

Out[21]:

Time	L06_347	LS06_347	LS06_348	LS06_34C	LS06_34D	LS06_34E	LS06_34G	season
2008-12-31	9053.371	9051.862	997.975	4826.749	939.998	1167.250	926.223	Winter
2009-12-31	7683.888	7672.038	859.617	4092.716	1036.323	1053.482	319.755	Winter
2010-12-31	11933.623	12356.573	906.725	6588.184	1294.821	1212.729	4918.342	Winter
2011-12-31	8458.819	8299.722	598.294	NaN	826.962	710.203	NaN	Winter
2012-12-31	8029.035	7910.918	771.414	3.002	1111.521	760.598	NaN	Winter
2013-12-31	167.232	167.232	13.726	NaN	24.894	5.903	NaN	None

In [22]:
```
#slicing of the dataframes
myflowserie['L06_347']['2009'].plot()
```

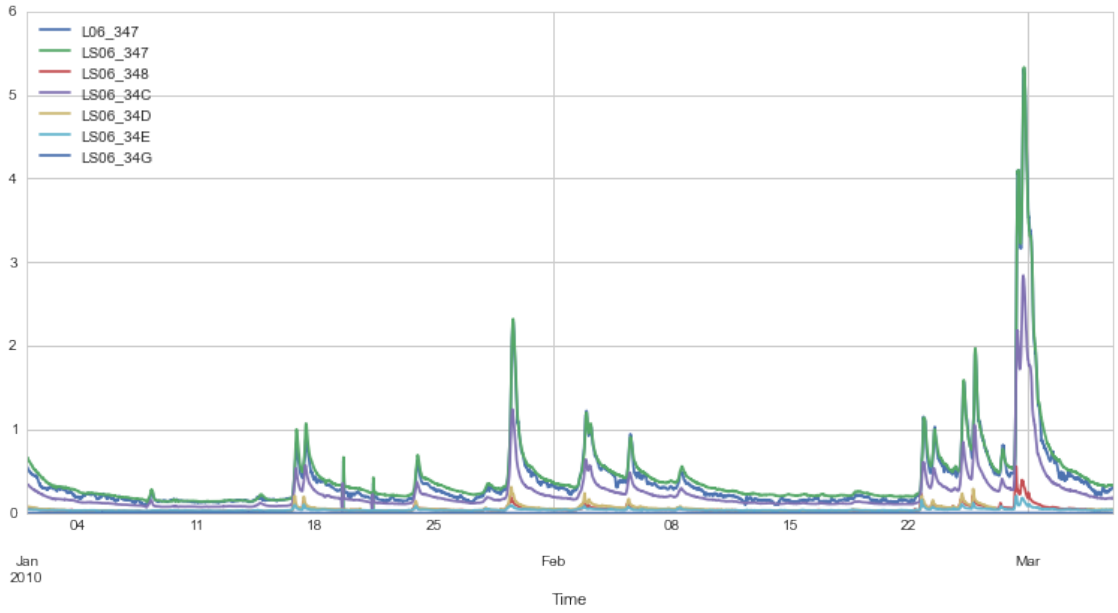
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe92de17710>



1.5.2 Easy period selection

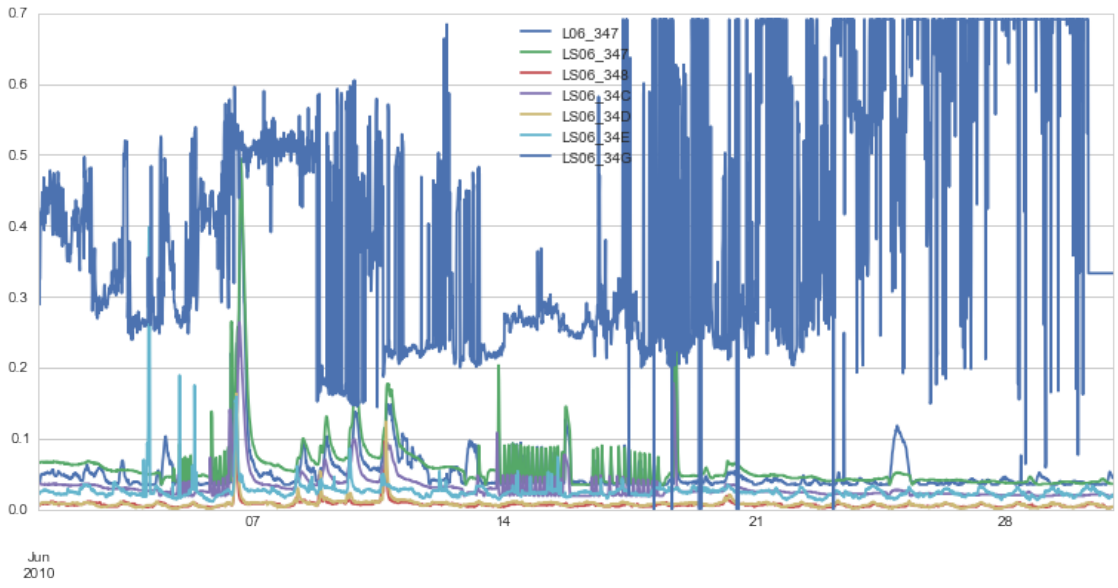
In [23]:
```
# get_month, get_year, get_season, get_date_range
myflowserie.get_date_range("01/01/2010","03/05/2010").plot(figsize=(13, 6))
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe92dd2e950>




```
In [24]: # or combine different statements:
         myflowserie.get_year('2010').get_month(6).plot(figsize=(13, 6))
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe92db25290>
```



For the seasons some options are available: Meteorologic (first of the month) or astrologic (21st of the month)

```
In [25]: myflowserie.current_season_dates()
```

```
Out[25]: {'Autumn': '0901', 'Spring': '0301', 'Summer': '0601', 'Winter': '1201'}
```

In [26]:
```
myflowserie.info_season_dates('north', 'astro')
```

Out[26]:
```
{'Autumn': '0921', 'Spring': '0321', 'Summer': '0621', 'Winter': '1221'}
```

1.5.3 'Hydrological' selections

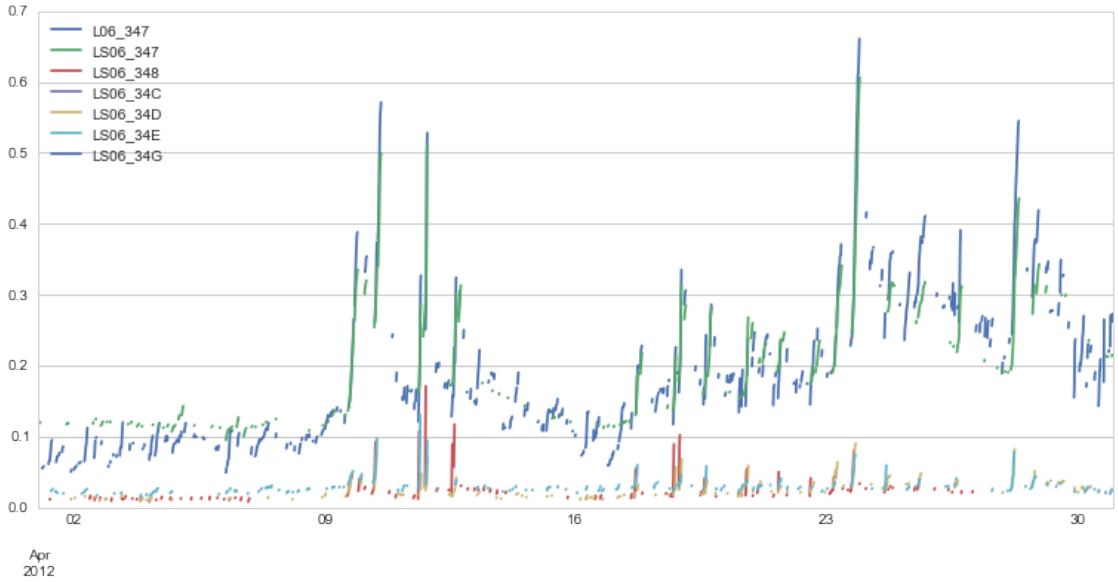
In [27]:
```
# Peaks (high or low)
myflowserie['LS06_348'].get_year('2012').get_highpeaks(60, above_percentile=0.8).data.dropna()
```

Out[27]:

Time	LS06_348	season
2012-01-01 23:15:00	0.199	Winter
2012-01-03 17:15:00	0.351	Winter
2012-01-04 23:45:00	0.415	Winter
2012-01-07 07:45:00	0.070	Winter
2012-01-09 10:45:00	0.031	Winter

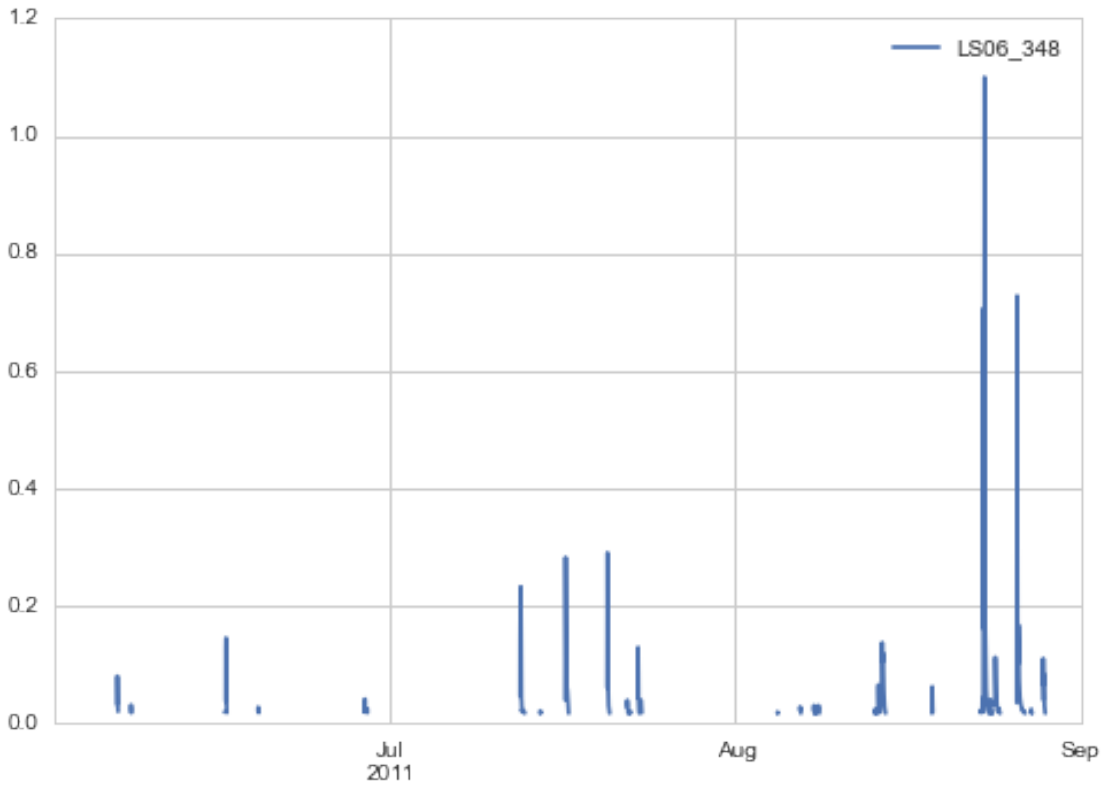
In [28]:
```
# Recessions and climbing periods  get_recess, get_climbing
myflowserie.get_year("2012").get_month("april").get_climbing().plot(figsize=(13, 6))
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe92dd0c510>
```



In [29]:
```
# above/below certain percentile values
myflowserie["LS06_348"].get_above_percentile(0.6).get_year('2011').get_season('summer').plot()
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe92c3365d0>
```



Furthermore:

- get_storms_per_year (in combination with rain-data)
- get_above_baseflow (in combination with baseflow-data) => next on the list
- ...

In [ ]:

In [ ]: