In this notebook a simple water balance based on the measured discharges are set up in the Mark-Dintel-Vliet (MDV) system.

In the cells below, the various used packages are loaded and some basic variables are set correctly.

**import** urllib

**import** knmi

**import** func

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** datetime **as** dt

**import** matplotlib **as** mpl

**%matplotlib** inline

**%config** InlineBackend.figure\_format='retina'

**from** matplotlib **import** rcParams

rcParams**.**update({'figure.figsize': (12,10)})

mpl**.**style**.**use('bmh')

ts\_id **=** []

Q **=** pd**.**DataFrame()

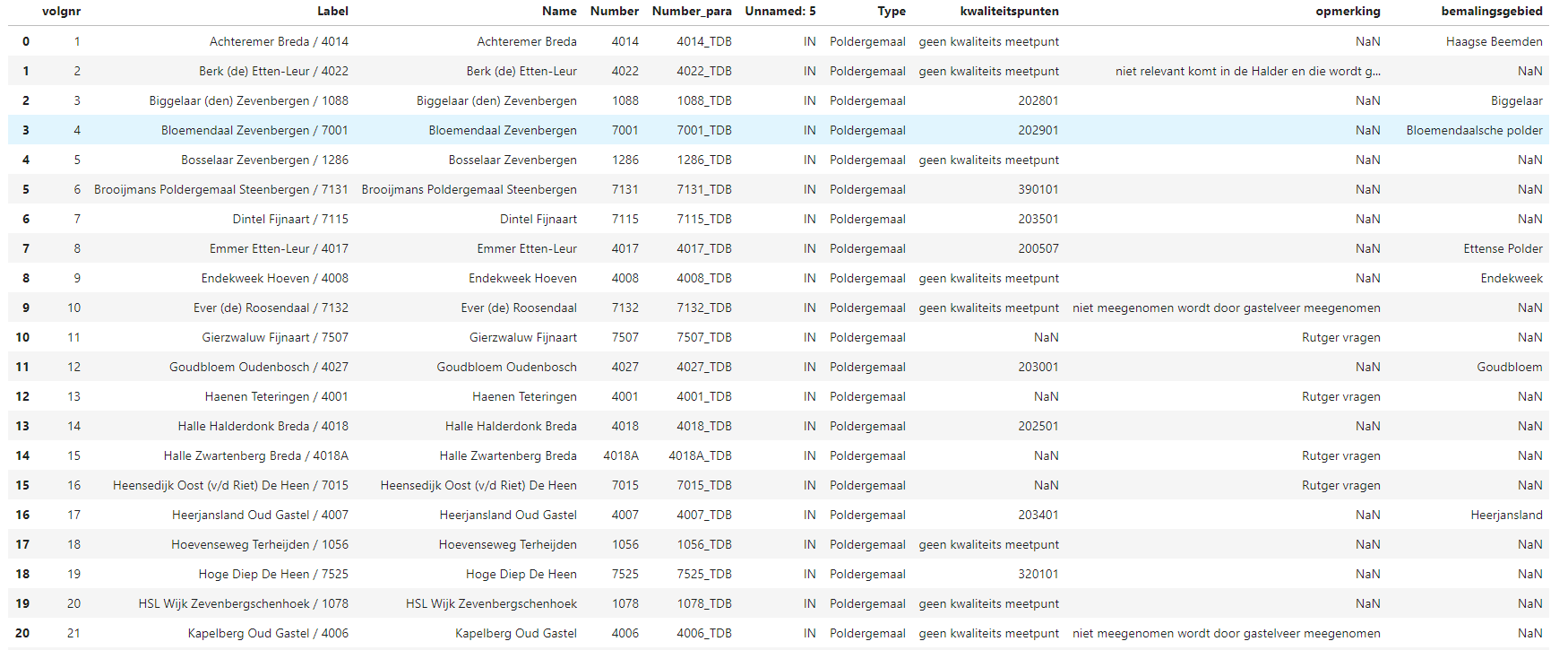
g **=** pd**.**DataFrame()

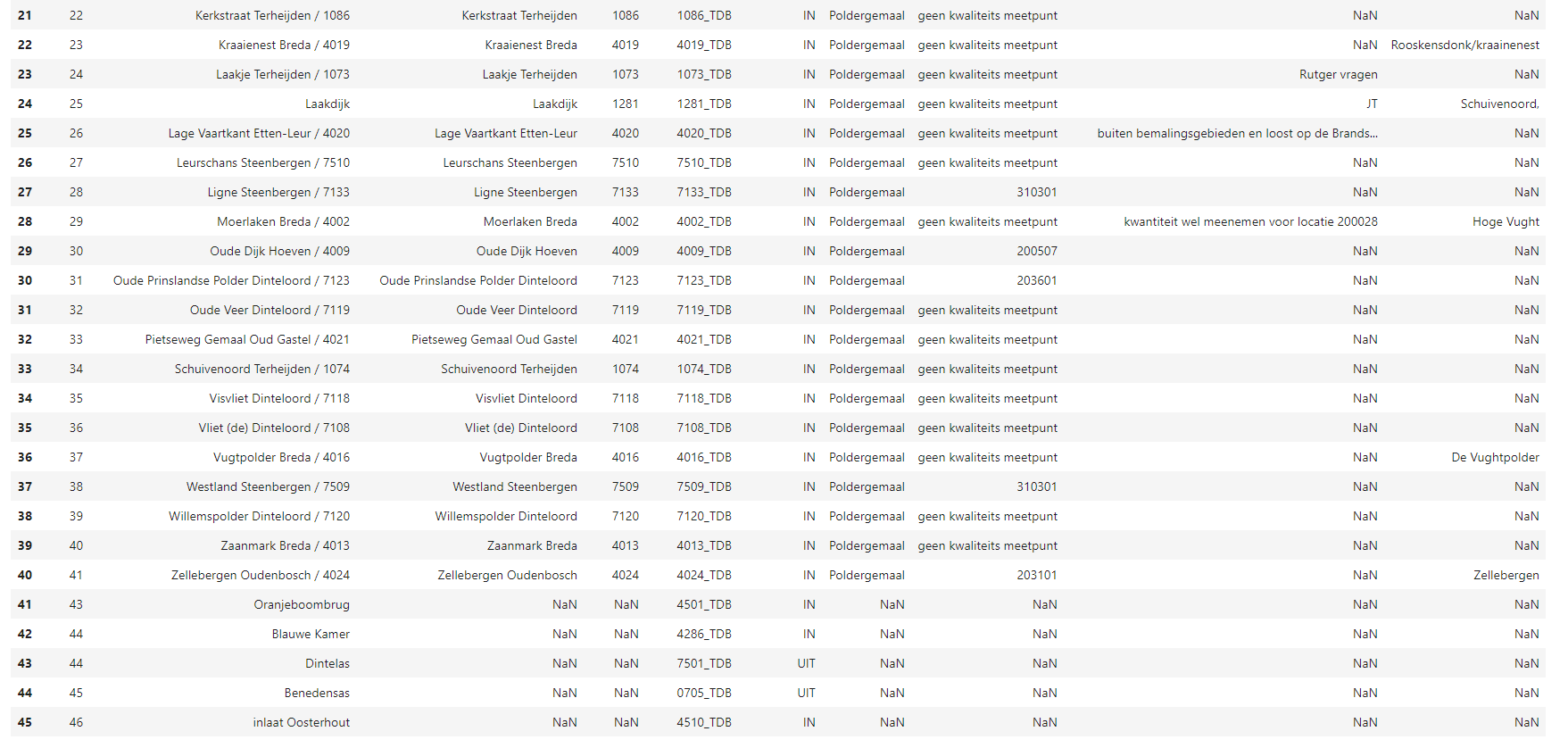
Below is a list of flow measurement points that are important for the water balance of the MDV. This list was used to perform the water balance for the water system analysis of the MDV.

MDV **=** pd**.**read\_csv('ids\_mpt.csv', sep**=**';')

MDV

Serial no label Name Number Number\_Para Type quality points remarks drainage area





station\_no **=** list(MDV['Number\_para']) *# meetstation nummer met parameterafkorting, bijv. ['4501\_TDB', '4251\_TDB']*

post **=** list(MDV['Unnamed: 5']) *# IN of UIT van meetstations hierboven, bijv. ['IN', 'UIT']*

A **=** 4540000 *# oppervlak van binnendijks MDV in m2*

The measurement series are downloaded via the WISKI API. If it concerns a pumping station, it is stored separately and later combined into a series of pumping stations. This is to keep things clear.

**for** i **in** range(len(station\_no)):

url\_ts\_id **=** 'http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getTimeseriesList&datasource=0&format=objson&stationparameter\_name=Debiet&station\_name=Totaaldebiet&ts\_name=Dag.Gem&station\_no='**+**station\_no[i]**+**'&returnfields=ts\_id'

ts\_id**.**append(pd**.**read\_json(urllib**.**request**.**urlopen(url\_ts\_id)**.**read()))

*# WISKI API opvragen alle stations namen*

url\_station\_name **=** 'http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getStationList&datasource=0&format=json&returnfields=station\_no,site\_name'

station\_name **=** pd**.**read\_json(urllib**.**request**.**urlopen(url\_station\_name)**.**read())

*# WISKI API opvragen meetreeks per timeseries ID*

**for** i **in** range(len(ts\_id)):

**try**:

**for** j **in** range(len(station\_name)):

**if** station\_name**.**iloc[j,0] **==** station\_no[i]:

name **=** station\_name**.**iloc[j,1]

str\_ts\_id **=** str(ts\_id[i]['ts\_id'][0])

url\_series **=** 'http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getTimeseriesValues&datasource=0&format=json&from=2018-01-01&period=P365D&metadata=True&dateformat=yyyy-MM-dd&ts\_id=' **+** str\_ts\_id

t **=** pd**.**DataFrame(pd**.**read\_json(urllib**.**request**.**urlopen(url\_series)**.**read())['data'][0], columns**=**['date', station\_no[i]]) *# inlezen meetreeks naar dataframe*

t['date'] **=** pd**.**to\_datetime(t['date']) *# datums parsen*

t**.**set\_index(['date'], inplace**=True**) *# datum als index*

*#t[t<0] = float('nan') # als er negatieve waardes zijn naar nan zetten*

**if** post[i] **==** 'UIT': *# de UIT posten negatief maken*

t **=** t**\*-**1

*#t = t\*86400/(A)\*1000 # van m3/s naar mm*

t**.**columns **=** [name]

**if** MDV['Type'][i] **==** 'Poldergemaal':

g **=** pd**.**concat([g,t], 1)

**else**:

Q **=** pd**.**concat([Q,t], 1) *# alle debietreeksen samenvoegen tot een dataframe*

**except**:

**continue**

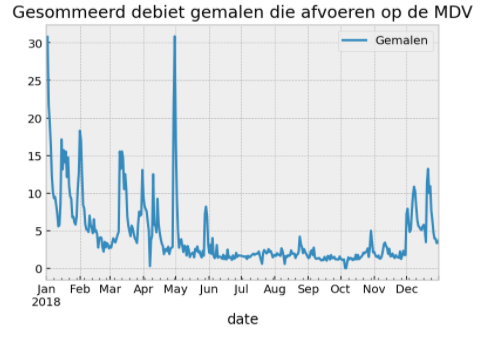
t **=** g**.**sum(axis**=**1)

t **=** pd**.**DataFrame(t)

t**.**columns **=** ['Gemalen']

t**.**plot(legend**=True**, title**=**'Summed flow rate of ground that discharges to the MDV')

Q **=** pd**.**concat([Q,t], axis**=**1)



Determination of the flow rate through the Mark lock by using the formula for stowed underflow:



with:

Q: discharge (m3 / s)

cw: lateral contraction coefficient (= 0.63) Af: wet surface (m2) g: acceleration of gravity (= 9.81 m / s2)

h1: water depth upstream from the stop (m)

h2: water depth downstream from the stop (m)

**def** getMeetreeks(station\_no, period**=**'complete'):

"" "Returns the measured Series of measurements and metadata for given station\_no

Parameters

----------

station\_no: String

Station\_no like 4510\_WTH\_BOV for which the Measurement series is gathered from the KiWis service.

Returns

-------

ho: pandas.Series

pandas. Series of the complete measured Measurement series for the selected station number.

meta: pandas.DataFrame

pandas.DataFrame with metadata of the selected station number.

"" "

# find timeseries\_id (ts\_id) for the given station

url\_ts\_id **=** ('http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getTimeseriesList'

'&datasource=0&format=objson&ts\_name=ContinueMeting.P&station\_no=%s' **%** station\_no)

ts\_id **=** pd**.**read\_json(urllib**.**request**.**urlopen(url\_ts\_id)**.**read())

*# find the x, y coordinates for the given station*

url\_xy **=** ('http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getTimeseriesList'

'&datasource=0&format=objson&ts\_name=ContinueMeting.P'

'&returnfields=station\_name,station\_no,station\_id,ts\_id,ts\_name,parametertype\_id,'

'parametertype\_name,station\_latitude,station\_longitude&station\_no=%s' **%** station\_no)

xy **=** pd**.**read\_json(urllib**.**request**.**urlopen(url\_xy)**.**read())

*# get string for the given station*

str\_ts\_id **=** str(ts\_id['ts\_id'][0])

url\_series **=** ('http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getTimeseriesValues'

'&datasource=0&format=json&from=2018-01-01&period=%s&metadata=True&ts\_id=%s'**%** (period,str\_ts\_id))

series **=** pd**.**read\_json(urllib**.**request**.**urlopen(url\_series)**.**read())

*# haal alle stations namen binnen*

url\_station\_name **=** ('http://10.10.3.126:8080/KiWIS/KiWIS?service=kisters&type=queryServices&request=getStationList'

'&datasource=0&format=json&returnfields=station\_no,site\_name')

station\_name **=** pd**.**read\_json(urllib**.**request**.**urlopen(url\_station\_name)**.**read())

**try**:

station\_no **=** series['station\_no'][0]

temp **=** series['data'][0]

header **=** ['date', station\_no]

d **=** pd**.**DataFrame(temp, columns **=** header)

d['date'] **=** pd**.**to\_datetime(d['date'])

d **=** d**.**set\_index(['date'])

*# vind stations naam voor stations nummer*

name **=** station\_name[station\_name[0] **==** station\_no][1]

**except**:

print('%s kan niet worden gevonden' **%** station\_no)

meta **=** pd**.**DataFrame()

meta **=** meta**.**append({'station\_no': ts\_id['station\_no'][0], 'name': name, 'station\_name': ts\_id['station\_name'][0], 'station\_id': xy['station\_id'][0], 'ts\_name': ts\_id['ts\_name'][0], 'lat': xy['station\_latitude'][0], 'lon': xy['station\_longitude'][0]}, ignore\_index**=True**)

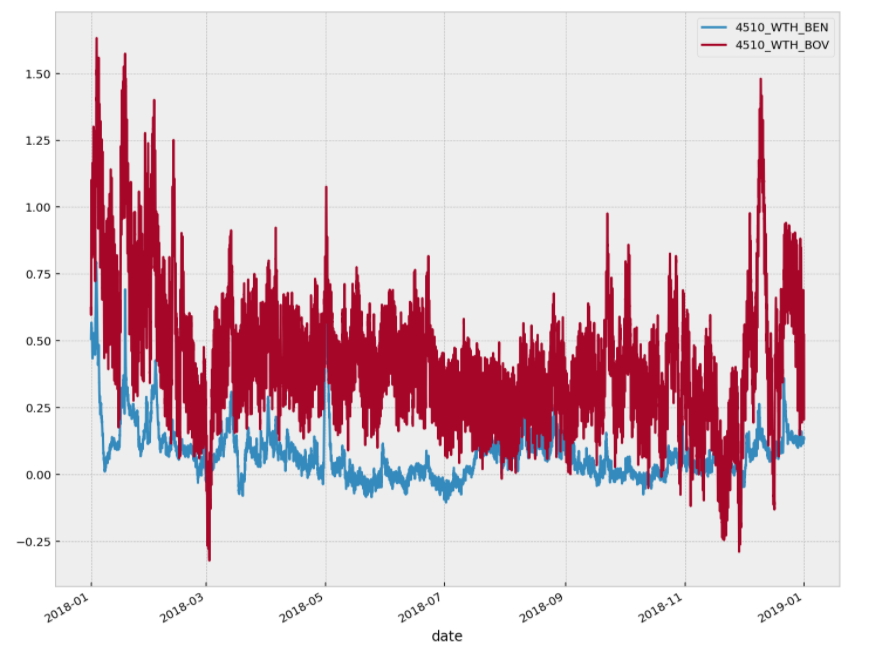
**return** d, meta

wth\_ben, meta\_wth\_ben **=** getMeetreeks(station\_no**=**'4510\_WTH\_BEN', period**=**'P1Y')

wth\_bov, meta\_wth\_ben **=** getMeetreeks(station\_no**=**'4510\_WTH\_BOV', period**=**'P1Y')

ax **=** wth\_ben**.**plot(legend**=True**)

wth\_bov**.**plot(ax**=**ax,legend**=True**);



import xml.etree.ElementTree as ET

tree = ET.parse ('hwlw-KEIZVR-Keizersveer-20180101-20181231.xml')

root = tree.getroot ()

tide = list ()

datetime = list ()

for elem in root [4]:

for subelem in elem.findall ('tide'):

tide.append (subelem.text)

for subelem in elem.findall ('datetime'):

datetime.append (pd.to\_datetime (subelem.text))

t = pd.DataFrame ({'tide': tide, 'datetime': datetime})

t.index = t.datetime.copy ()

t = t ['20180726': '20180815']

t ['datetime'] = t ['datetime']. dt.round ('15min')

t = t [t ['tide'] == 'LW']

t ['starttime'] = t ['datetime'] - pd.Timedelta (hours = 1)

t ['endtime'] = t ['datetime'] + pd.Timedelta (hours = 1)

d = pd.read\_csv ('schutten.csv', sep = ';', parse\_dates = [[0,1], [0,2]])

d.columns = ['starttime', 'endtime']

d

df **=** pd**.**concat([wth\_ben, wth\_bov], axis**=**1)

df **=** df['20180726':'20180906']**.**copy()

df['Q bov'] **=** 4 **\*** 1.3 **\*** 0.6 **\*** 0.63 **\*** 0.8 **\*** np**.**sqrt(2**\***9.81**\***(df['4510\_WTH\_BOV']**-**df['4510\_WTH\_BEN']))

df['Q onder'] **=** 4 **\*** 1.3 **\*** 0.6 **\*** 0.63 **\*** 0.7 **\*** np**.**sqrt(2**\***9.81**\***(df['4510\_WTH\_BOV']**-**df['4510\_WTH\_BEN']))

df['Q'] **=** df['Q bov'] **+** df['Q onder']

*# vanaf 26-07 tot 05-09-2018 1 uur voor en na hoogwater schutten, Q naar 0*

**for** i **in** range(len(t)):

df['Q'][t['starttime'][i]:t['endtime'][i]] **=** float('nan')

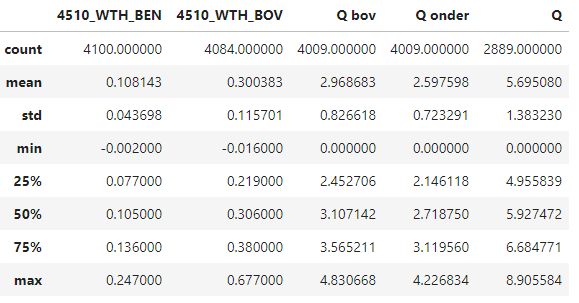
*# vanaf 16-08 tot 05-09-2018 vaste tijden schutten en spuien*

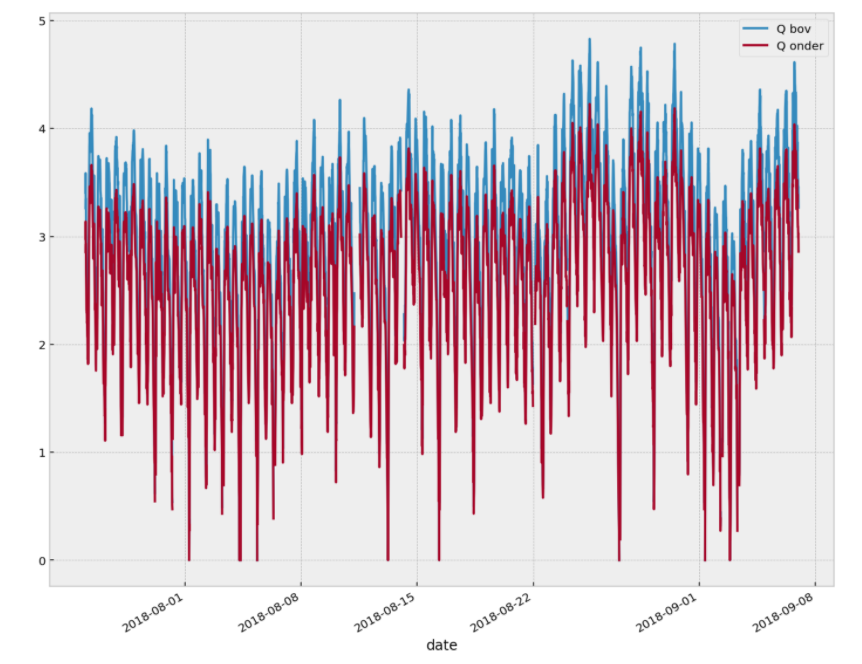
**for** i **in** range(len(d)):

df['Q'][d['starttime'][i]:d['endtime'][i]] **=** float('nan')

ax **=** df[['Q bov', 'Q onder']]**.**plot(style**=**'-')

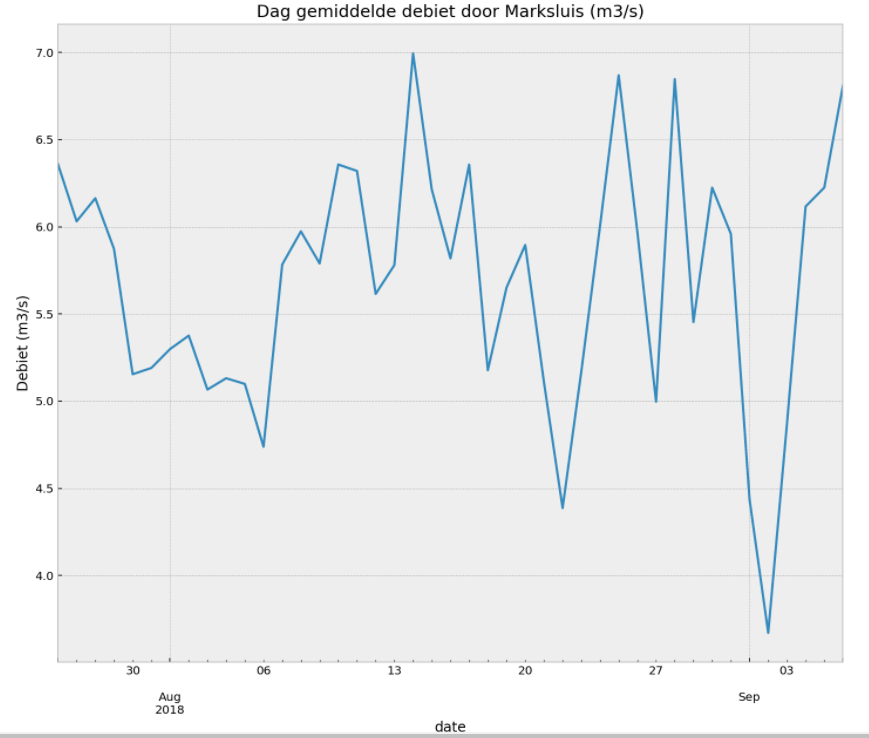
df**.**describe()





df['Q']**.**resample('D')**.**mean()**.**plot(title**=**’Day average flow through Marksluis (m3/s)')

plt**.**ylabel('Debiet (m3/s)')



Plot of the downloaded measurement series in m3 / s (where IN is positive and OUT is negative).

Q **=** Q['20180401':'20181201']**.**copy()**.**interpolate()

Q['Marksluis'] **=** df['Q']**.**resample('D')**.**mean()

Qs **=** pd**.**DataFrame(index**=**Q**.**index)

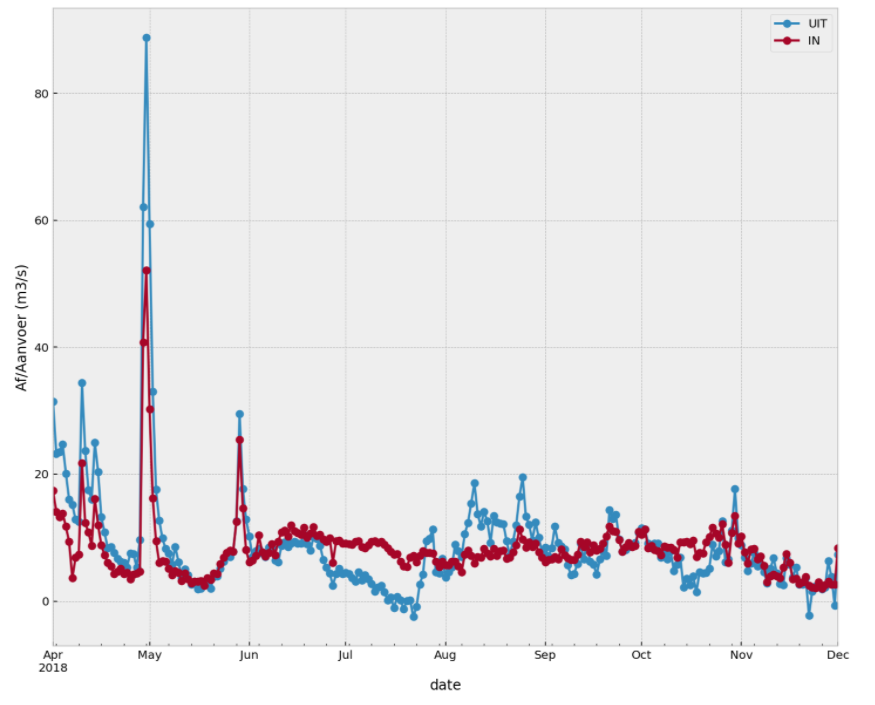
Qs['IN'] **=** Q[['Oranjeboombrug Breda','Blauwe Kamer Ulvenhout','Gemalen','Oosterhout Inlaatduiker', 'Marksluis']]**.**sum(axis**=**1)

Qs['UIT'] **=** Q[['Dintelsas Dinteloord','Benedensas De Heen']]**.**sum(axis**=**1) **\*** **-**1

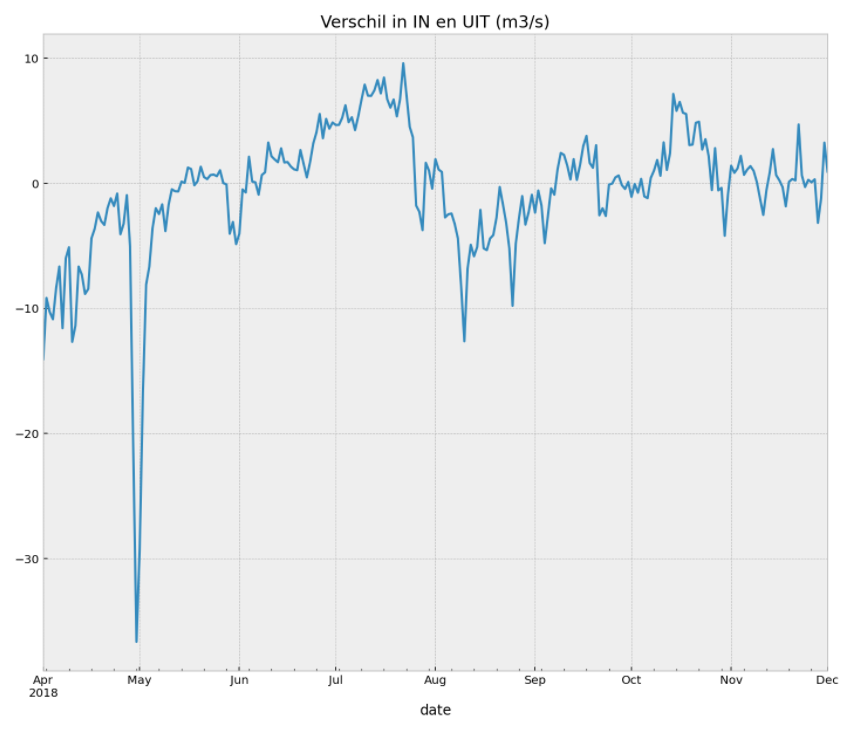
Qs['IN-UIT'] **=** Qs['IN'] **-** Qs['UIT']

Qs[['UIT', 'IN']]['2018']**.**plot(style**=**'-o')

plt**.**ylabel('Af/Aanvoer (m3/s)');



Qs['IN-UIT']**.**plot(title**=**'Difference in IN and OUT (m3/s)');



dfy **=** Q**.**resample('a')**.**sum()**\***86400

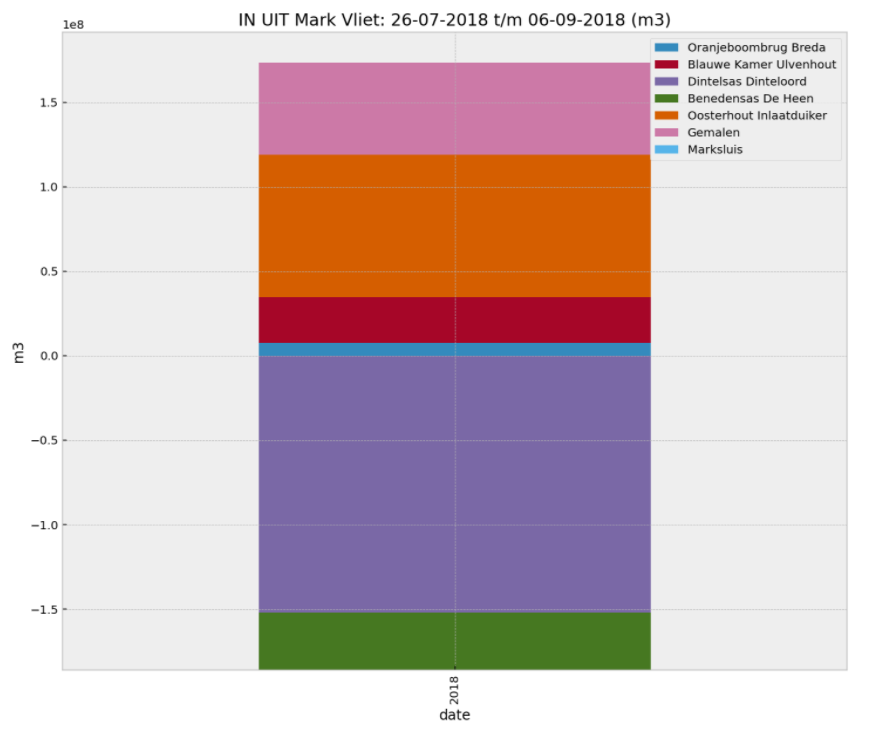
dfy**.**index **=** dfy**.**index**.**strftime('%Y')

title **=** str('IN UIT Mark Vliet: 26-07-2018 t/m 06-09-2018 (m3)')

ax1 **=** dfy**.**plot**.**bar(stacked**=True**, title**=**title)

ax1**.**set\_ylabel('m3')

ax1**.**legend();



Below, the evaporation series is retrieved for KNMI measuring station Gilze-Rijen. Then the mm / day is converted to m3 / s for the MDV boezem.

station\_nr = 350 # standard 350 Gilze Rijen for evaporation

knmi\_ = knmi.get\_day\_data\_dataframe (stations = [station\_nr], start = dt.date (1987,1,1), variables = ['EV24']) # bring in evaporation from KNMI station Gilze Rijen

knmi\_.drop ('STN', 1, inplace = True)

knmi\_.index = knmi\_.index.strftime ('% Y-% m-% d')

knmi\_.index = pd.to\_datetime (knmi\_.index)

EV = pd.DataFrame (knmi \_ ['EV24'] / - 10) / 1000 \* A / 86400 # evaporation KNMI Gilze RIjen

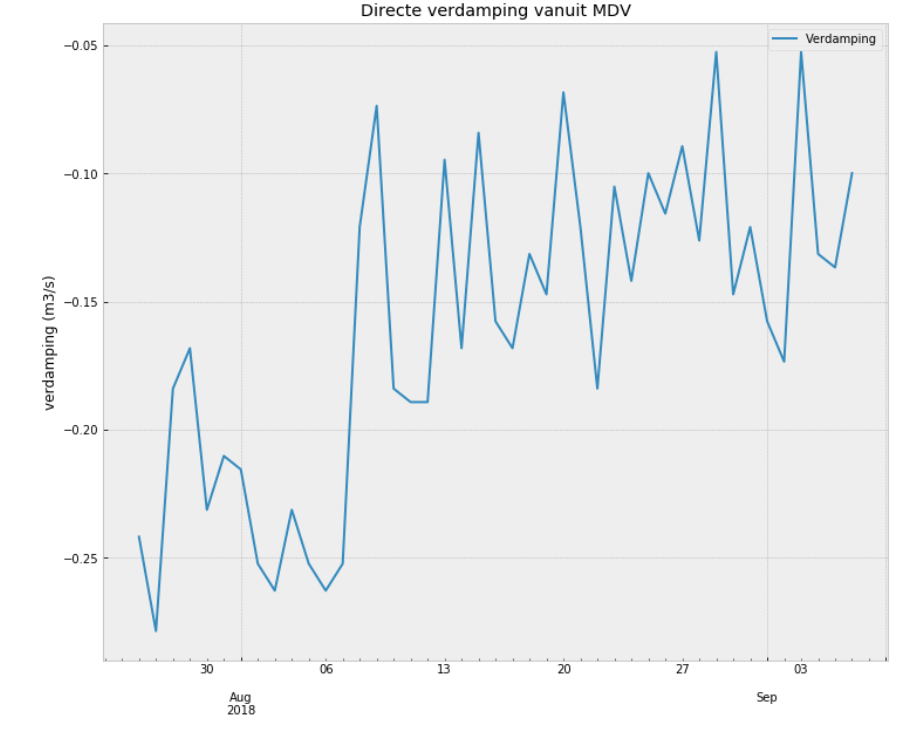
EV.columns = ['Evaporation']

EV = EV ['20180726': '20180906']. Copy ()

Plot of the evaporation series.

EV.plot (title = 'Direct evaporation from MDV')

plt.ylabel ('evaporation (m3 / s)');



Below, the precipitation series is retrieved for KNMI precipitation station Oudenbosch. Then the mm / day is converted to m3 / s for the MDV chimney breast.

station\_no = 828

knmi\_ = func.getRain (start = '20180726', end = '20180906', stns = '828')

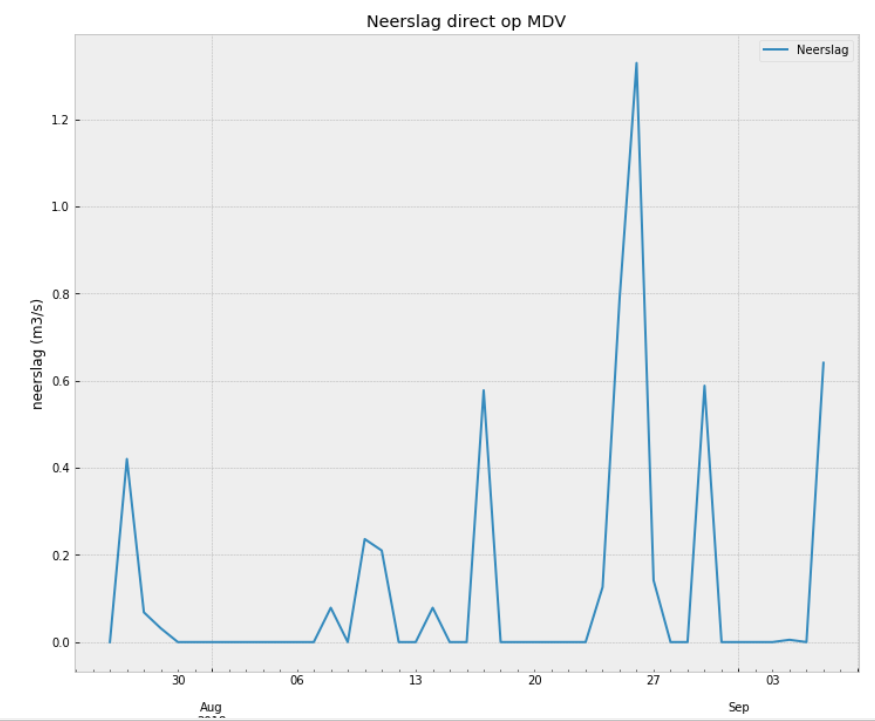
P = pd.DataFrame (knmi\_.resample ('D'). Mean ()) \* A / 86400 # precipitation KNMI GIlze Rijen

P.columns = ['Precipitation']

Plot of precipitation series.

P.plot (title = 'Precipitation directly on MDV')

plt.ylabel ('precipitation (m3 / s)');



Merging of flow series, evaporation and precipitation series. And calculate REST entry.

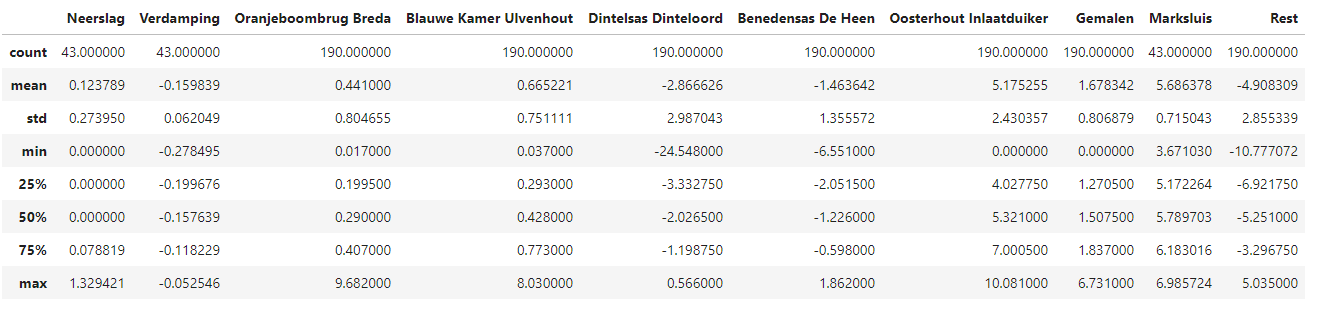
df = pd.concat ([P, EV, Q], 1) # merge all INs and OUTs into a data frame

df ['Remainder'] = df.sum (1) \* - calculate 1 # REST entry

Some general numbers (in m3 / s) of the collected data for the daily sums.

df.describe ()

Precipitation Evaporation



Suddenly

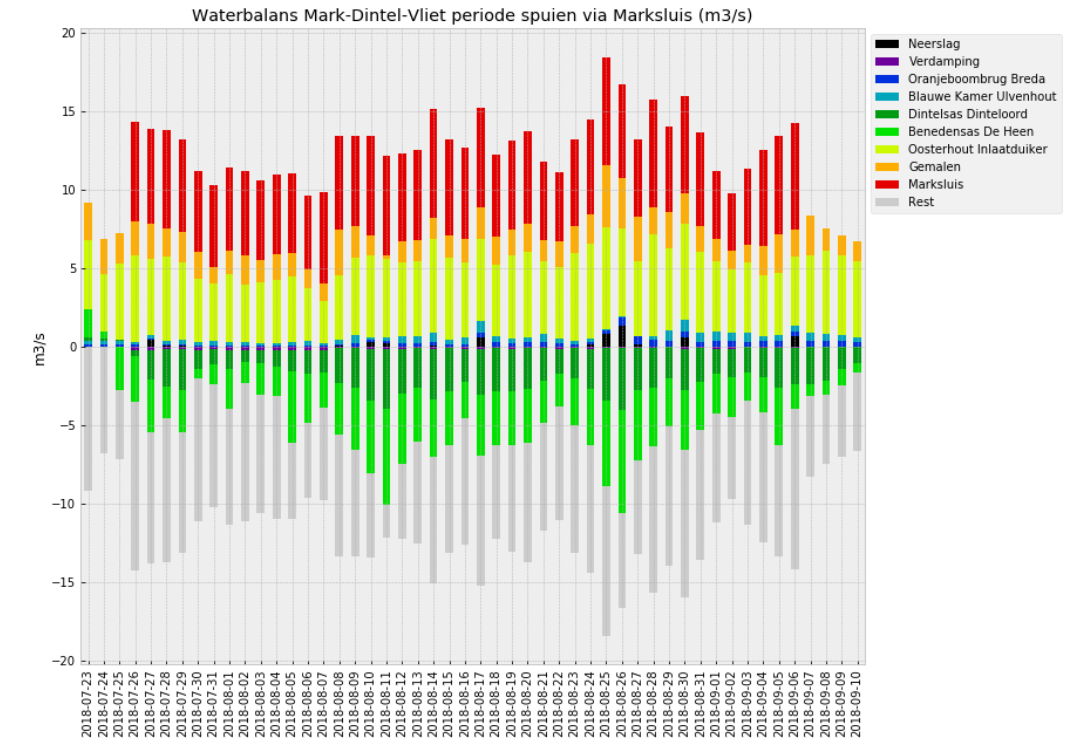
df.index = pd.to\_datetime (df.index)

df.index = df.index.strftime ('% Y-% m-% d')

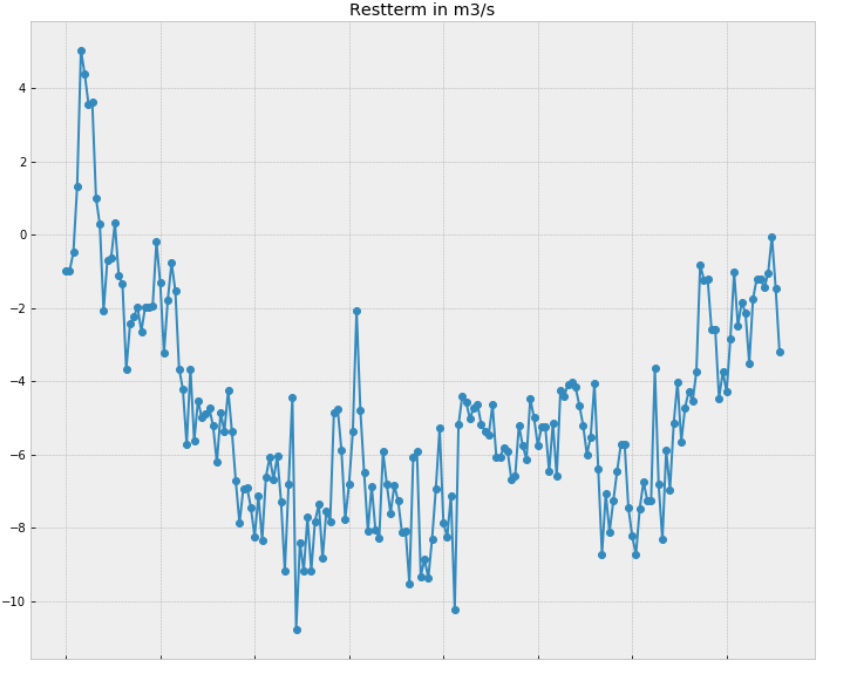
df ['2018-07-23': '2018-09-10']. plot.bar (stacked = True, title = 'Water balance Mark-Dintel-Vliet period draining via Marksluis (m3 / s)', cmap = ' nipy\_spectral ')

plt.ylabel ('m3 / s');

plt.legend (loc = 'upper left', bbox\_to\_anchor = (1, 1));



df ['Rest']. plot (style = 'o-', title = 'Rest term in m3 / s');



df.index = pd.to\_datetime (df.index)

df ['Rest (l / s / ha)'] = df ['Rest'] / 28100 \* -1000

df ['max'] = df ['Rest (l / s / ha)'] \* 1.25

df ['min'] = df ['Rest (l / s / ha)'] \* 0.75

ax = df ['Rest (l / s / ha)'] ['20180701': '20181001']. plot (style = 'o-', title = 'Estimation of consumption on the MDV in l / s / ha ', c =' navy ', label =' estimated consumption in l / s / ha ')

df ['max'] ['20180701': '20181001']. plot (ax = ax, style = '-', alpha = 0.75, c = 'royalblue', label = '\_ nolegend\_')

df ['min'] ['20180701': '20181001']. plot (ax = ax, style = '-', alpha = 0.75, c = 'royalblue', label = '\_ nolegend\_')

ax.fill\_between (df ['20180701': '20181001']. index, df ['max'] ['20180701': '20181001'], df ['min'] ['20180701': '20181001'], color = 'royalblue', alpha = 0.3, label = '25% bandwidth ')

ax.axhline (y = 0.3, linestyle = '-', c = 'k', label = 'water availability limit')

plt.legend ()

plt.xlabel ('date')

plt.ylabel ('residual term in l / s / ha')

