hydropandas

Release 0.11.1

Artesia

Mar 25, 2024

CONTENTS:

Hydropandas is a Python package for reading, analyzing and writing (hydrological) timeseries data. Users can store a timeseries and metadata in a single object. This object inherits from a pandas DataFrame, with all its wonderful features, and is extended with custom methods and attributes related to hydrological timeseries.

CHAPTER

ONE

SUPPORTED DATA SOURCES

Currently supported datasources that can be read:

- BRO
- DINOLoket CSV
- FEWS PI-XML
- IMOD groundwater models
- KNMI data
- Lizard
- MODFLOW groundwater models
- Pastastore, for managing Pastas timeseries and models
- Waterinfo
- WISKI csv files

ObsCollection can be exported to:

- Excel (with one tab for each time series and a single tab with all metadata)
- Shapefile
- Pastastore

See the table of contents to get started with *hydropandas*.

1.1 Getting Started

On this page you will find all the information to get started with *hydropandas*.

1.1.1 Getting Python

To install *hydropandas*, a working version of Python 3.7 or higher has to be installed on your computer. We recommend using the Anaconda Distribution of Python.

1.1.2 Installing hydropandas

Install the module by typing:

```
pip install hydropandas
```

Please note that some of the dependencies cannot be installed automatically on Windows. If you do not have these packages already you can install them manually using the following instructions:

Download these packages from Christoph Gohlke's website :

- GDAL
- Fiona
- Shapely
- Python-snappy
- Fastparquet

Use CTRL+F to find the download link on the page. Be sure to download the correct version of the package. The Python version should match your Python version. Also the architecture should match (i.e. 64bits vs 32bits). For example:

- GDAL-3.1.4-cp38-cp38-win_amd64.whl

This is the GDAL version for Python *3.8* (as can be seen from the *cp38* in the name), for *64-bits* Python (as derived from the *amd64* in the name).

Once you have downloaded the correct files, open Anaconda Prompt, and navigate to the directory in which you saved your downloads. Now use the following commands to install the packages (the order is important):

```
pip install GDAL-3.1.4-cp38-cp38-win_amd64.whl
pip install Fiona-1.8.17-cp38-cp38-win_amd64.whl
pip install Shapely-1.7.1-cp38-cp38-win_amd64.whl
pip install python_snappy-0.5.4-cp38-cp38-win_amd64.whl
pip install fastparquet-0.4.1-cp38-cp38-win_amd64.whl
```

After you've done this you can install hydropandas using:

```
pip install hydropandas
```

For installing in development mode, clone the repository and install by typing the following from the module root directory:

```
pip install -e .
```

1.1.3 Using *hydropandas*

Start Python and import the module:

```
import hydropandas as hpd
```

1.1.4 Dependencies

This module has several optional dependencies that have to be installed. These include:

- pastastore (create pastas models of an ObsCollection)
- folium and bokeh (make an interactive map of an ObsCollection)
- xarray and netCDF4 (get regis layers for groundwater observations)
- flopy (interaction with modflow data)

See the *Examples* section for some quick examples on how to get started.

1.2 Examples

This page provides some short example code snippets and links to Jupyter Notebooks with more detailed examples.

1.2.1 Example snippets

Importing a single CSV-file downloaded from DINOLoket:

```
import hydropandas as hpd
fname = './tests/data/2019-Dino-test/Grondwaterstanden_Put/B33F0080001_1.csv'
gw = hpd.GroundwaterObs.from_dino(path=fname)
```

Or for a zipfile:

```
import hydropandas as hpd
dinozip = './tests/data/2019-Dino-test/dino.zip'
dino_gw = hpd.ObsCollection.from_dino(dirname=dinozip,
                                      subdir='Grondwaterstanden_Put',
                                      suffix='1.csv',
                                      ObsClass=hpd.GroundwaterObs,
                                      keep_all_obs=False)
```

1.2.2 Example notebooks

The links below link to Jupyter Notebooks with explanation and examples of the usage of the *hydropandas* module:

Reading groundwater observations

This notebook introduces how to use the `hydropandas` package to read, process and visualise groundwater data from Dino and Bro databases.

Notebook contents

[1]:
```
import hydropandas as hpd

import logging
from IPython.display import HTML

import logging
```

[2]:
```
hpd.util.get_color_logger("INFO")
```

GroundwaterObs

The hydropandas package has several functions to read groundwater observations at a measurement well. These include reading data from: - dino (from csv-files). - bro (using the bro-api) - fews (xml dumps from the fews database) - wiski (dumps from the wiski database)

[3]:
```
# reading a dino csv file
path = "data/Grondwaterstanden_Put/B33F0080001_1.csv"
gw_dino = hpd.GroundwaterObs.from_dino(path=path)
gw_dino
```

```
INFO:hydropandas.io.dino:reading -> B33F0080001_1
```

[4]:
```
# reading the same filter from using the bro api. Specify a groundwater monitoring id␣
↪(GMW00...) and a filter number (1)
gw_bro = hpd.GroundwaterObs.from_bro("GMW000000041261", 1)
```

```
INFO:hydropandas.io.bro:reading bro_id GMW000000041261
INFO:hydropandas.io.bro:GLD000000009378 contains 36 duplicates (of 66447). Keeping only␣
↪first values.
```

Now we have an `GroundwaterObs` object named `gw_bro` and `gw_dino`. Both objects are from the same measurement well in different databases. A `GroundwaterObs` object inherits from a pandas `DataFrame` and has the same attributes and methods.

[5]:
```
gw_bro.describe()
```

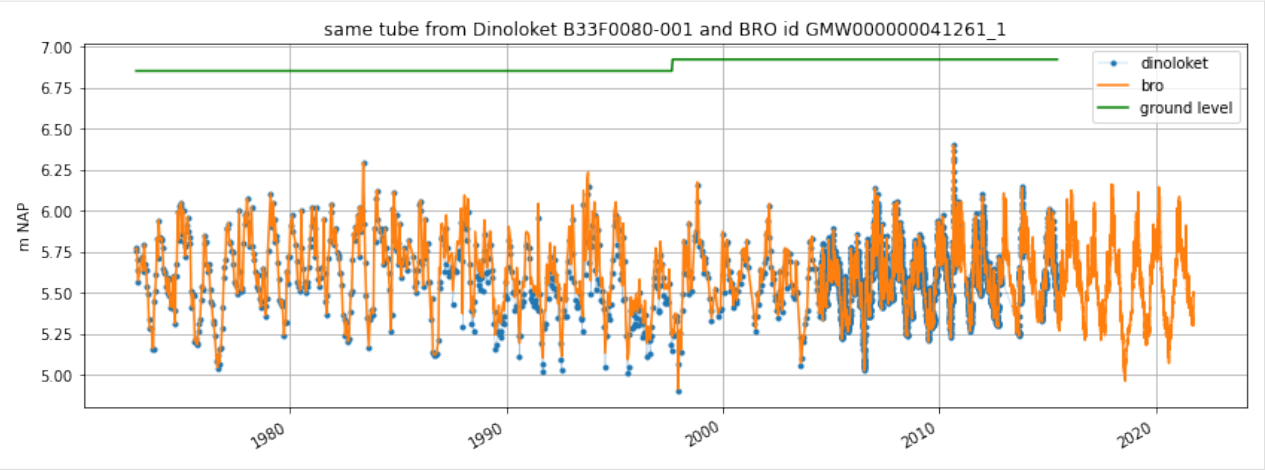
[5]:

	values
count	66411.000000
mean	5.562630
std	0.222262
min	4.913000
25%	5.380000
50%	5.569000
75%	5.721000
max	6.397000

[6]:
```
gw_bro
```

[6]:
```
GroundwaterObs GMW000000041261_1
-----metadata------
name : GMW000000041261_1
x : 213268.0
y : 473910.0
filename :
source : BRO
unit : m NAP
monitoring_well : GMW000000041261
tube_nr : 1
screen_top : 4.05
screen_bottom : 3.05
ground_level : 6.9
tube_top : 7.173
metadata_available : True

-----time series------
                     values    qualifier
1972-11-28 00:00:00   5.763  goedgekeurd
1972-12-07 00:00:00   5.773  goedgekeurd
1972-12-14 00:00:00   5.703  goedgekeurd
1972-12-21 00:00:00   5.643  goedgekeurd
1972-12-28 00:00:00   5.573  goedgekeurd
...                     ...          ...
2021-10-08 07:00:00   5.486  goedgekeurd
2021-10-08 08:00:00   5.485  goedgekeurd
2021-10-08 09:00:00   5.486  goedgekeurd
2021-10-08 09:47:00   5.491  goedgekeurd
2021-10-08 10:00:00   5.485  goedgekeurd

[66411 rows x 2 columns]
```

[7]:
```
ax = gw_dino["stand_m_tov_nap"].plot(
    label="dinoloket", figsize=(14, 5), legend=True, marker=".", lw=0.2
)
gw_bro["values"].plot(ax=ax, label="bro", legend=True, ylabel=gw_bro.unit)
gw_dino["ground_level"].plot(
    ax=ax,
    label="ground level",
    legend=True,
    grid=True,
    color="green",
    ylabel=gw_dino.unit,
)

ax.set_title(f"same tube from Dinoloket {gw_dino.name} and BRO id {gw_bro.name}")
```



GroundwaterObs Attributes

Besides the standard `DataFrame` attributes a `GroundwaterObs` has the following additional attributes: - x, y: x- and y-coordinates of the observation point - name: str with the name - filename: str with the filename (only available when the data was loaded from a file) - monitoring_well: the name of the monitoring_well. One monitoring well can have multiple tubes. - tube_nr: the number of the tube. The combination of monitoring_well and tube_nr should be unique - screen_top: the top of the tube screen (bovenkant filter in Dutch) - screen_bottom: the bottom of the tube screen (onderkant filter in Dutch) - ground_level: surface level (maaiveld in Dutch) - tube_top: the top of the tube - metadata_available: boolean indicating whether metadata is available for this observation point - meta: dictionary with additional metadata

When dowloading from Dinoloket all levels are in meters NAP.

[8]:
```
print(gw_bro)
```

```
GroundwaterObs GMW000000041261_1
-----metadata------
name : GMW000000041261_1
x : 213268.0
y : 473910.0
filename :
source : BRO
unit : m NAP
monitoring_well : GMW000000041261
tube_nr : 1
screen_top : 4.05
screen_bottom : 3.05
ground_level : 6.9
tube_top : 7.173
metadata_available : True

-----time series------
                     values    qualifier
1972-11-28 00:00:00   5.763  goedgekeurd
1972-12-07 00:00:00   5.773  goedgekeurd
1972-12-14 00:00:00   5.703  goedgekeurd
1972-12-21 00:00:00   5.643  goedgekeurd
```

(continues on next page)

(continued from previous page)

```
1972-12-28 00:00:00   5.573  goedgekeurd
...                     ...          ...
2021-10-08 07:00:00   5.486  goedgekeurd
2021-10-08 08:00:00   5.485  goedgekeurd
2021-10-08 09:00:00   5.486  goedgekeurd
2021-10-08 09:47:00   5.491  goedgekeurd
2021-10-08 10:00:00   5.485  goedgekeurd

[66411 rows x 2 columns]
```

GroundwaterObs methods

Besides the standard `DataFrame` methods a `GroundwaterObs` has additional methods. This methods are accessible through submodules: - `geo.get_lat_lon()`, to obtain latitude and longitude - `gwobs.get_modellayer()`, to obtain the modellayer of a modflow model using the filter depth - `stats.get_seasonal_stat()`, to obtain seasonal statistics - `stats.obs_per_year()`, to obtain the number of observations per year - `stats.consecutive_obs_years()`, to obtain the number of consecutive years with more than a minimum number of observations - `plots.interactive_plot()`, to obtain a bokeh plot

Get latitude and longitude with `gw.geo.get_lat_lon()`:

[9]:
```
print(f"latitude and longitude -> {gw_bro.geo.get_lat_lon()}")
```

```
latitude and longitude -> (52.25014706738195, 6.24047994529121)
```

[10]:
```
gw_bro.stats.get_seasonal_stat(stat="mean")
```

[10]:
```
                    winter_mean  summer_mean
GMW000000041261_1      5.723791      5.40682
```

[11]:
```
p = gw_bro.plots.interactive_plot("figure")
HTML(filename="figure/{}.html".format(gw_bro.name))
```

[11]:
```
<IPython.core.display.HTML object>
```

ObsCollections

`ObsCollections` are a combination of multiple observation objects. The easiest way to construct an `ObsCollections` is from a list of observation objects.

[12]:
```
path1 = "data/Grondwaterstanden_Put/B33F0080001_1.csv"
path2 = "data/Grondwaterstanden_Put/B33F0133001_1.csv"
gw1 = hpd.GroundwaterObs.from_dino(path=path1)
gw2 = hpd.GroundwaterObs.from_dino(path=path2)

# create ObsCollection
oc = hpd.ObsCollection([gw1, gw2], name="Dino groundwater")
oc
```

```
INFO:hydropandas.io.dino:reading -> B33F0080001_1
INFO:hydropandas.io.dino:reading -> B33F0133001_1
```

[12]:
```
                     x         y       filename source   unit monitoring_well
name
B33F0080-001  213260.0  473900.0  B33F0080001_1   dino  m NAP        B33F0080  \
B33F0133-001  210400.0  473366.0  B33F0133001_1   dino  m NAP        B33F0133

              tube_nr  screen_top  screen_bottom  ground_level  tube_top
name
B33F0080-001      1.0        3.85           2.85          6.92      7.18  \
B33F0133-001      1.0      -67.50         -70.00          6.50      7.14

              metadata_available
name
B33F0080-001                True  \
B33F0133-001                True

                                                          obs
name
B33F0080-001  GroundwaterObs B33F0080-001
-----metadata-----...
B33F0133-001  GroundwaterObs B33F0133-001
-----metadata-----...
```

Now we have an `ObsCollection` object named `oc`. The `ObsCollection` contains all the data from the two `GroundwaterObs` objects. It also stores a reference to the `GroundwaterObs` objects in the 'obs' column. An `ObsCollection` object also inherits from a pandas `DataFrame` and has the same attributes and methods.

[13]:
```
# get columns
oc.columns
```

[13]:
```
Index(['x', 'y', 'filename', 'source', 'unit', 'monitoring_well', 'tube_nr',
       'screen_top', 'screen_bottom', 'ground_level', 'tube_top',
       'metadata_available', 'obs'],
      dtype='object')
```

[14]:
```
# get individual GroundwaterObs object from an ObsCollection
o = oc.loc["B33F0133-001", "obs"]
o
```

[14]:
```
GroundwaterObs B33F0133-001
-----metadata------
name : B33F0133-001
x : 210400.0
y : 473366.0
filename : B33F0133001_1
source : dino
unit : m NAP
monitoring_well : B33F0133
tube_nr : 1.0
screen_top : -67.5
screen_bottom : -70.0
ground_level : 6.5
tube_top : 7.14
metadata_available : True
```

(continues on next page)

(continued from previous page)

```
-----time series------
            stand_m_tov_nap   locatie  filternummer  stand_cm_tov_mp
1989-12-14             1.20  B33F0133             1            582.0  \
1990-01-15             1.57  B33F0133             1            545.0
1990-01-29             1.70  B33F0133             1            532.0
1990-02-14             1.53  B33F0133             1            549.0
1990-03-01             1.56  B33F0133             1            546.0
...                     ...       ...           ...              ...
2011-01-14             3.57  B33F0133             1            357.0
2011-01-15             3.60  B33F0133             1            354.0
2011-01-16             3.61  B33F0133             1            353.0
2011-01-17             3.61  B33F0133             1            353.0
2011-01-18             3.63  B33F0133             1            351.0

            stand_cm_tov_mv  stand_cm_tov_nap bijzonderheid opmerking
1989-12-14            530.0             120.0           NaN       NaN  \
1990-01-15            493.0             157.0           NaN       NaN
1990-01-29            480.0             170.0           NaN       NaN
1990-02-14            497.0             153.0           NaN       NaN
1990-03-01            494.0             156.0           NaN       NaN
...                     ...               ...           ...       ...
2011-01-14            293.0             357.0           NaN       NaN
2011-01-15            290.0             360.0           NaN       NaN
2011-01-16            289.0             361.0           NaN       NaN
2011-01-17            289.0             361.0           NaN       NaN
2011-01-18            287.0             363.0           NaN       NaN

            tube_top
1989-12-14      7.02
1990-01-15      7.02
1990-01-29      7.02
1990-02-14      7.02
1990-03-01      7.02
...              ...
2011-01-14      7.14
2011-01-15      7.14
2011-01-16      7.14
2011-01-17      7.14
2011-01-18      7.14

[2212 rows x 9 columns]
```

[15]:
```
# get statistics
oc.describe()
```

[15]:
```
                   x              y  tube_nr  screen_top  screen_bottom  \
count       2.000000       2.000000      2.0    2.000000       2.000000
mean   211830.000000  473633.000000      1.0  -31.825000     -33.575000
std      2022.325394     377.595021      0.0   50.452069      51.512729
min    210400.000000  473366.000000      1.0  -67.500000     -70.000000
25%    211115.000000  473499.500000      1.0  -49.662500     -51.787500
50%    211830.000000  473633.000000      1.0  -31.825000     -33.575000
```

(continues on next page)

(continued from previous page)

```
75%    212545.000000  473766.500000      1.0  -13.987500     -15.362500
max    213260.000000  473900.000000      1.0    3.850000       2.850000

       ground_level  tube_top
count      2.000000  2.000000
mean       6.710000  7.160000
std        0.296985  0.028284
min        6.500000  7.140000
25%        6.605000  7.150000
50%        6.710000  7.160000
75%        6.815000  7.170000
max        6.920000  7.180000
```

ObsCollection methods

Besides the methods of a pandas `DataFrame` an `ObsCollection` has additional methods stored in submodules.

`geo`: - `get_bounding_box` -> get a tuple with (xmin, ymin, xmax, ymax) - `get_extent` -> get a tule with (xmin, xmax, ymin, ymax) - `get_lat_lon` -> to get the lattitudes and longitudes from the x and y coordinates - `within_polygon` -> to select only the observations that lie within a polygon

`gwobs`: - `set_tube_nr` -> to set the tube numbers based on the tube screen depth when there are multiple tubes at one monitoring well - `set_tube_nr_monitoring_well` -> find out which observations are at the same location with a different screen depth. Set monitoring_well and tube_nr attributes accordingly.

`plots`: - `interactive_figures` -> create bokeh figures for each observation point. - `interactive_map` -> create a folium map with observation points and bokeh figures for each observation point. - `section_plot` -> create a plot of multiple observations and a plot of the well layout.

`stats`: - `get_first_last_obs_date()` -> get the first and the last date of the observations for each observation point - `get_no_of_observations()` -> get the number of observations - `get_seasonal_stat()` -> get seasonal stats of the observations

E.g. get the bounding box with `gw.geo.get_bounding_box()`:

[16]:
```
print(f"bounding box -> {oc.geo.get_bounding_box()}")
```

```
bounding box -> (210400.0, 473366.0, 213260.0, 473900.0)
```

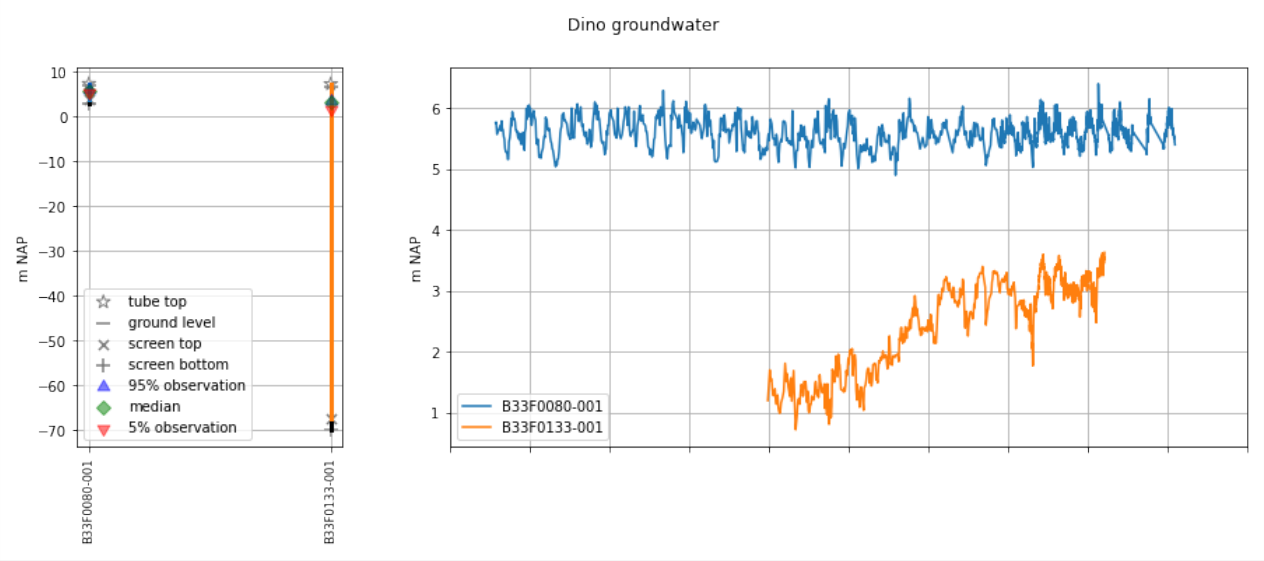
[17]:
```
oc.geo.set_lat_lon()
oc.plots.interactive_map(plot_dir="figure")
```

[17]:
```
<folium.folium.Map at 0x1ba4f0a1040>
```

We can get an overview of the well layout and observations via `plots.section_plot`:

[18]:
```
oc.plots.section_plot()
```

```
INFO:hydropandas.extensions.plots:created sectionplot -> B33F0133-001
```



ObsCollection Attributes

An `ObsCollection` also has additional attributes: - name, a str with the name of the collection - meta, a dictionary with additional metadata

[19]:
```
print(f"name is -> {oc.name}")
print(f"meta is -> {oc.meta}")
```

```
name is -> Dino groundwater
meta is -> {}
```

Read ObsCollections

Instead of creating the ObsCollection from a list of observation objects. It is also possible to read the data from a source into an ObsCollection at once. The following sources can be read as an ObsCollection:

- bro (using the api)
- dino (from files)
- fews (dumps from the fews database)
- wiski (dumps from the wiski database)
- menyanthes (a .men file)
- modflow (from the heads of a modflow model)
- imod (from the heads of an imod model)

This notebook won't go into detail on all the sources that can be read. Only the two options for reading data from Dino and BRO are shown below.

[20]:
```
# read using a .zip file with data
dinozip = "data/dino.zip"
dino_gw = hpd.read_dino(dirname=dinozip, keep_all_obs=False)
dino_gw
```

```
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B02H0092001_1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B02H0092001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B02H1007001_1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B02H1007001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B04D0032002_1.csv
WARNING:hydropandas.io.dino:could not read metadata -> Grondwaterstanden_Put/B04D0032002_
↪1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B04D0032002_1.csv
INFO:root:not added to collection -> Grondwaterstanden_Put/B04D0032002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B27D0260001_1.csv
WARNING:hydropandas.io.dino:could not read metadata -> Grondwaterstanden_Put/B27D0260001_
↪1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B27D0260001_1.csv
INFO:root:not added to collection -> Grondwaterstanden_Put/B27D0260001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0080001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0080002_1.csv
```

```
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0133001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0133002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B37A0112001_1.csv
WARNING:hydropandas.io.dino:could not read metadata -> Grondwaterstanden_Put/B37A0112001_
↪1.csv
WARNING:hydropandas.io.dino:could not read measurements -> Grondwaterstanden_Put/
↪B37A0112001_1.csv
INFO:root:not added to collection -> Grondwaterstanden_Put/B37A0112001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003003_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003004_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0092004_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0092005_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0102001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0167001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0212001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B22D0155001_1.csv
```

[20]:

name	x	y	filename	\
B02H0092-001	219890.0	600030.0	Grondwaterstanden_Put/B02H0092001_1.csv	
B02H1007-001	219661.0	600632.0	Grondwaterstanden_Put/B02H1007001_1.csv	
B33F0080-001	213260.0	473900.0	Grondwaterstanden_Put/B33F0080001_1.csv	
B33F0080-002	213260.0	473900.0	Grondwaterstanden_Put/B33F0080002_1.csv	
B33F0133-001	210400.0	473366.0	Grondwaterstanden_Put/B33F0133001_1.csv	
B33F0133-002	210400.0	473366.0	Grondwaterstanden_Put/B33F0133002_1.csv	
B42B0003-001	38165.0	413785.0	Grondwaterstanden_Put/B42B0003001_1.csv	
B42B0003-002	38165.0	413785.0	Grondwaterstanden_Put/B42B0003002_1.csv	
B42B0003-003	38165.0	413785.0	Grondwaterstanden_Put/B42B0003003_1.csv	
B42B0003-004	38165.0	413785.0	Grondwaterstanden_Put/B42B0003004_1.csv	
B58A0092-004	186924.0	372026.0	Grondwaterstanden_Put/B58A0092004_1.csv	
B58A0092-005	186924.0	372026.0	Grondwaterstanden_Put/B58A0092005_1.csv	

(continues on next page)

(continued from previous page)

```
B58A0102-001  187900.0  373025.0  Grondwaterstanden_Put/B58A0102001_1.csv
B58A0167-001  185745.0  371095.0  Grondwaterstanden_Put/B58A0167001_1.csv
B58A0212-001  183600.0  373020.0  Grondwaterstanden_Put/B58A0212001_1.csv
B22D0155-001  233830.0  502530.0  Grondwaterstanden_Put/B22D0155001_1.csv

             source   unit monitoring_well  tube_nr  screen_top
name
B02H0092-001   dino  m NAP        B02H0092      1.0         NaN  \
B02H1007-001   dino  m NAP        B02H1007      1.0         NaN
B33F0080-001   dino  m NAP        B33F0080      1.0        3.85
B33F0080-002   dino  m NAP        B33F0080      2.0      -10.15
B33F0133-001   dino  m NAP        B33F0133      1.0      -67.50
B33F0133-002   dino  m NAP        B33F0133      2.0     -104.20
B42B0003-001   dino  m NAP        B42B0003      1.0       -2.00
B42B0003-002   dino  m NAP        B42B0003      2.0      -34.00
B42B0003-003   dino  m NAP        B42B0003      3.0      -60.00
B42B0003-004   dino  m NAP        B42B0003      4.0     -107.00
B58A0092-004   dino  m NAP        B58A0092      4.0     -115.23
B58A0092-005   dino  m NAP        B58A0092      5.0     -134.23
B58A0102-001   dino  m NAP        B58A0102      1.0       -3.35
B58A0167-001   dino  m NAP        B58A0167      1.0       23.33
B58A0212-001   dino  m NAP        B58A0212      1.0       26.03
B22D0155-001   dino  m NAP        B22D0155      1.0        7.80

              screen_bottom  ground_level  tube_top  metadata_available
name
B02H0092-001            NaN           NaN       NaN                True  \
B02H1007-001            NaN          1.92       NaN                True
B33F0080-001           2.85          6.92      7.18                True
B33F0080-002         -12.15          6.92      7.17                True
B33F0133-001         -70.00          6.50      7.14                True
B33F0133-002        -106.20          6.50      7.12                True
B42B0003-001          -3.00          6.50      6.99                True
B42B0003-002         -35.00          6.50      6.99                True
B42B0003-003         -61.00          6.50      6.95                True
B42B0003-004        -108.00          6.50      6.97                True
B58A0092-004        -117.23         29.85     29.61                True
B58A0092-005        -137.23         29.84     29.62                True
B58A0102-001          -8.35         29.65     29.73                True
B58A0167-001          22.33         30.50     30.21                True
B58A0212-001          25.53         28.49     28.53                True
B22D0155-001           6.80          8.91      9.94                True

                                                                   obs
name
B02H0092-001  GroundwaterObs B02H0092-001
-----metadata-----...
B02H1007-001  GroundwaterObs B02H1007-001
-----metadata-----...
B33F0080-001  GroundwaterObs B33F0080-001
-----metadata-----...
B33F0080-002  GroundwaterObs B33F0080-002
```

(continues on next page)

(continued from previous page)

```
-----metadata-----...
B33F0133-001  GroundwaterObs B33F0133-001
-----metadata-----...
B33F0133-002  GroundwaterObs B33F0133-002
-----metadata-----...
B42B0003-001  GroundwaterObs B42B0003-001
-----metadata-----...
B42B0003-002  GroundwaterObs B42B0003-002
-----metadata-----...
B42B0003-003  GroundwaterObs B42B0003-003
-----metadata-----...
B42B0003-004  GroundwaterObs B42B0003-004
-----metadata-----...
B58A0092-004  GroundwaterObs B58A0092-004
-----metadata-----...
B58A0092-005  GroundwaterObs B58A0092-005
-----metadata-----...
B58A0102-001  GroundwaterObs B58A0102-001
-----metadata-----...
B58A0167-001  GroundwaterObs B58A0167-001
-----metadata-----...
B58A0212-001  GroundwaterObs B58A0212-001
-----metadata-----...
B22D0155-001  GroundwaterObs B22D0155-001
-----metadata-----...
```

[21]:
```
# read from bro using an extent (Schoonhoven zuid-west)
oc = hpd.read_bro(extent=(117850, 118180, 439550, 439900), keep_all_obs=False)
oc
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
INFO:hydropandas.io.bro:reading bro_id GMW000000036319
INFO:hydropandas.io.bro:GLD000000012818 contains 720 duplicates (of 22303). Keeping only␣
↪first values.
 25%|       | 1/4 [00:04<00:14,  4.84s/it]
INFO:hydropandas.io.bro:reading bro_id GMW000000036327
INFO:hydropandas.io.bro:GLD000000012821 contains 720 duplicates (of 17856). Keeping only␣
↪first values.
 50%|     | 2/4 [00:08<00:08,  4.02s/it]
INFO:hydropandas.io.bro:reading bro_id GMW000000036365
INFO:hydropandas.io.bro:GLD000000012908 contains 329 duplicates (of 12793). Keeping only␣
↪first values.
 75%|  | 3/4 [00:11<00:03,  3.58s/it]
INFO:hydropandas.io.bro:reading bro_id GMW000000049567
INFO:hydropandas.io.bro:no groundwater level dossier for GMW000000049567 and tube number␣
↪1
WARNING:hydropandas.io.bro:no measurements found for gmw_id GMW000000049567 and tube␣
↪number1
```

(continues on next page)

(continued from previous page)

```
INFO:hydropandas.io.bro:reading bro_id GMW000000049567
INFO:hydropandas.io.bro:no groundwater level dossier for GMW000000049567 and tube number␣
↪2
WARNING:hydropandas.io.bro:no measurements found for gmw_id GMW000000049567 and tube␣
↪number2
```

```
100%|| 4/4 [00:12<00:00,  3.03s/it]
```

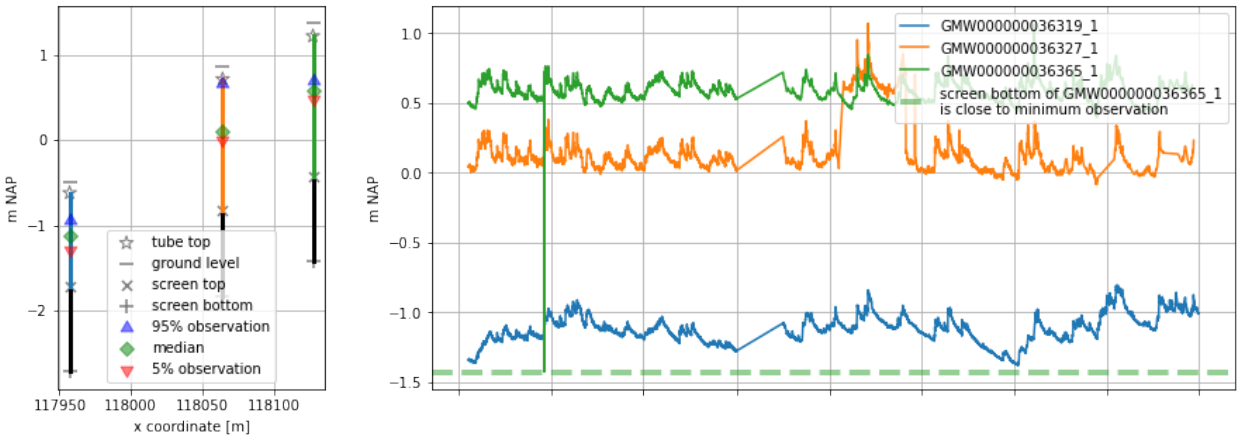
[21]:

```
                            x           y filename source   unit
name
GMW000000036319_1  117957.010  439698.236             BRO  m NAP  \
GMW000000036327_1  118064.196  439799.968             BRO  m NAP
GMW000000036365_1  118127.470  439683.136             BRO  m NAP

                   monitoring_well  tube_nr  screen_top  screen_bottom
name
GMW000000036319_1  GMW000000036319        1      -1.721         -2.721  \
GMW000000036327_1  GMW000000036327        1      -0.833         -1.833
GMW000000036365_1  GMW000000036365        1      -0.429         -1.428

                   ground_level  tube_top  metadata_available
name
GMW000000036319_1        -0.501    -0.621                True  \
GMW000000036327_1         0.856     0.716                True
GMW000000036365_1         1.371     1.221                True

                                                          obs
name
GMW000000036319_1  GroundwaterObs GMW000000036319_1
-----metadata...
GMW000000036327_1  GroundwaterObs GMW000000036327_1
-----metadata...
GMW000000036365_1  GroundwaterObs GMW000000036365_1
-----metadata...
```

[22]:

```
# plot wells, use x-coordinate in section plot
oc.plots.section_plot(section_colname_x="x", section_label_x="x coordinate [m]")
```

```
INFO:hydropandas.extensions.plots:created sectionplot -> GMW000000036365_1
```



Write ObsCollections

Sometimes reading ObsCollections can be time consuming, especially when you need to download a lot of data. It can be worth to save the ObsCollection to a file and read it later instead of going through the full read process again. There are two basic ways to do this: 1. Write the ObsCollection to a pickle 2. Write the ObsCollection to an excel file

	pickle	excel
extension	.pklz	.xlsx
human readable	No	Yes
data lost	None	Some metadata, see `hpd.to_excel` docstring

Pickling is used to store Python objects into a binary file that is not human readable. Writing and reading a pickle is fast and returns an exact copy of the ObsCollection. Exchanging pickles between machines can be troublesome because of machine settings and differences between package versions.

[23]:
```
oc.to_pickle("test.pklz")
```

[24]:
```
oc_pickled = hpd.read_pickle("test.pklz")
```

An ObsCollection can be written to Excel file. An Excel file with multiple sheets is created. One sheet with the metdata and another sheet for each observation in the ObsCollection. Writing to an excel file is considerably slower than writing a pickle but it does give you a human readable file format that can be easily exchanged between machines.

[25]:
```
oc.to_excel("test.xlsx")
```

```
WARNING:py.warnings:C:\Users\oebbe\02_python\hydropandas\hydropandas\obs_collection.py:
↪1933: UserWarning: Pandas requires version '1.4.3' or newer of 'xlsxwriter' (version
↪'1.3.8' currently installed).
  with pd.ExcelWriter(path) as writer:
```

[26]:
```
oc_excelled = hpd.read_excel("test.xlsx")
```

Read KNMI observations using hydropandas

This notebook introduces how to use the `hydropandas` package to read, process and visualise KNMI data.

Martin & Onno - 2022

Notebook contents

[1]:
```
import hydropandas as hpd
from hydropandas.io import knmi
from IPython.display import display

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
from scipy.interpolate import RBFInterpolator, NearestNDInterpolator

import logging
```

```
c:\Users\vonkm\miniconda3\envs\hpd_env\lib\site-packages\tqdm\auto.py:21: TqdmWarning:␣
↪IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.
↪readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[2]:
```
hpd.util.get_color_logger("INFO")
```

[2]:
```
<RootLogger root (INFO)>
```

Observation types

The hydropandas package has a function to read all kinds of KNMI observations. These are stored in an `Obs` object. There are three types of observations you can obtain from the KNMI: - `EvaporationObs`, for evaporation time series - `PrecipitationObs`, for precipitation time series - `MeteoObs`, for all the other meteorological time series

With the code below we get the Evaporation in [m/day] for KNMI station 344 (Rotterdam Airport).

[3]:
```
o = hpd.EvaporationObs.from_knmi(stn=344, start="2022")
display(o.head())
o.plot()
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable EV24from 2022-
↪01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi EV24 data from station 344-ROTTERDAM between 2022-
↪01-01 00:00:00 and None
```

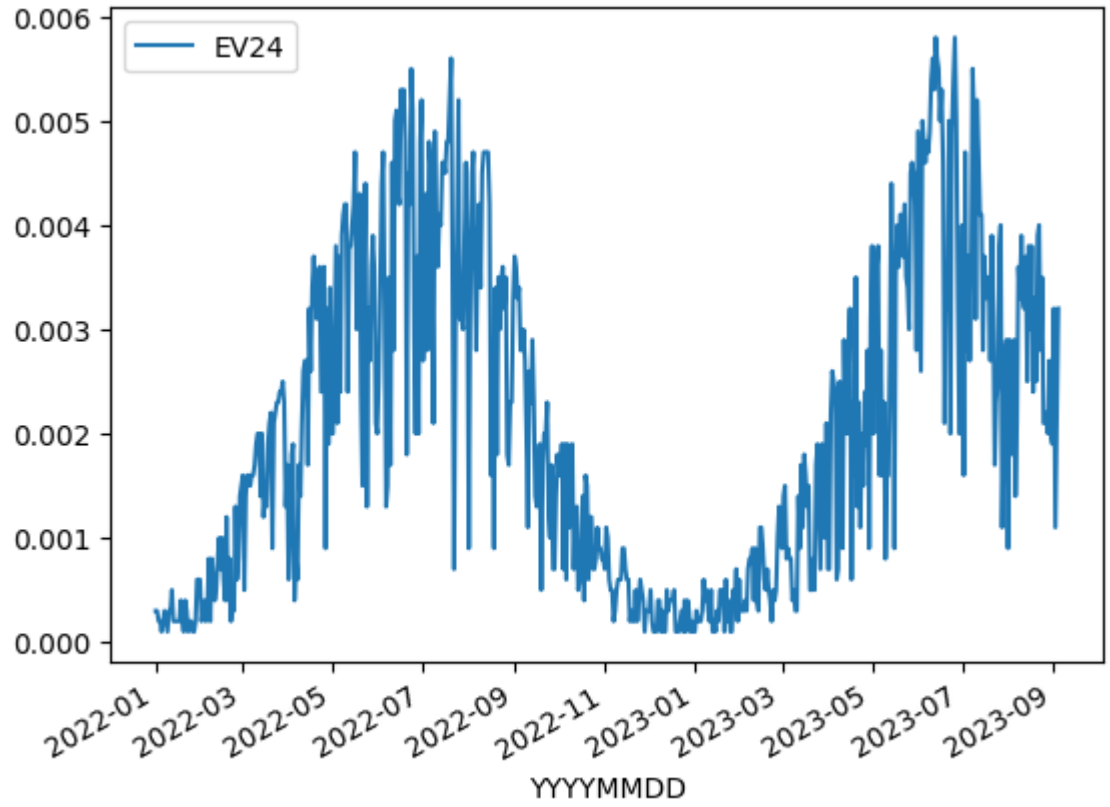
```
EvaporationObs EV24_ROTTERDAM
-----metadata------
name : EV24_ROTTERDAM
x : 90061.9308913925
y : 441636.8542853104
filename :
source : KNMI
unit :
station : 344
meteo_var : EV24

-----time series------
                     EV24
YYYYMMDD
2022-01-01 01:00:00  0.0003
```

(continues on next page)

(continued from previous page)

```
2022-01-02 01:00:00  0.0003
2022-01-03 01:00:00  0.0002
2022-01-04 01:00:00  0.0002
2022-01-05 01:00:00  0.0001
```

[3]:
```
<Axes: xlabel='YYYYMMDD'>
```



[4]:
```
o = hpd.PrecipitationObs.from_knmi(stn=344, start="2022")
display(o.head())
o.plot()
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable RHfrom 2022-
↪01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi RH data from station 344-ROTTERDAM between 2022-
↪01-01 00:00:00 and None
```

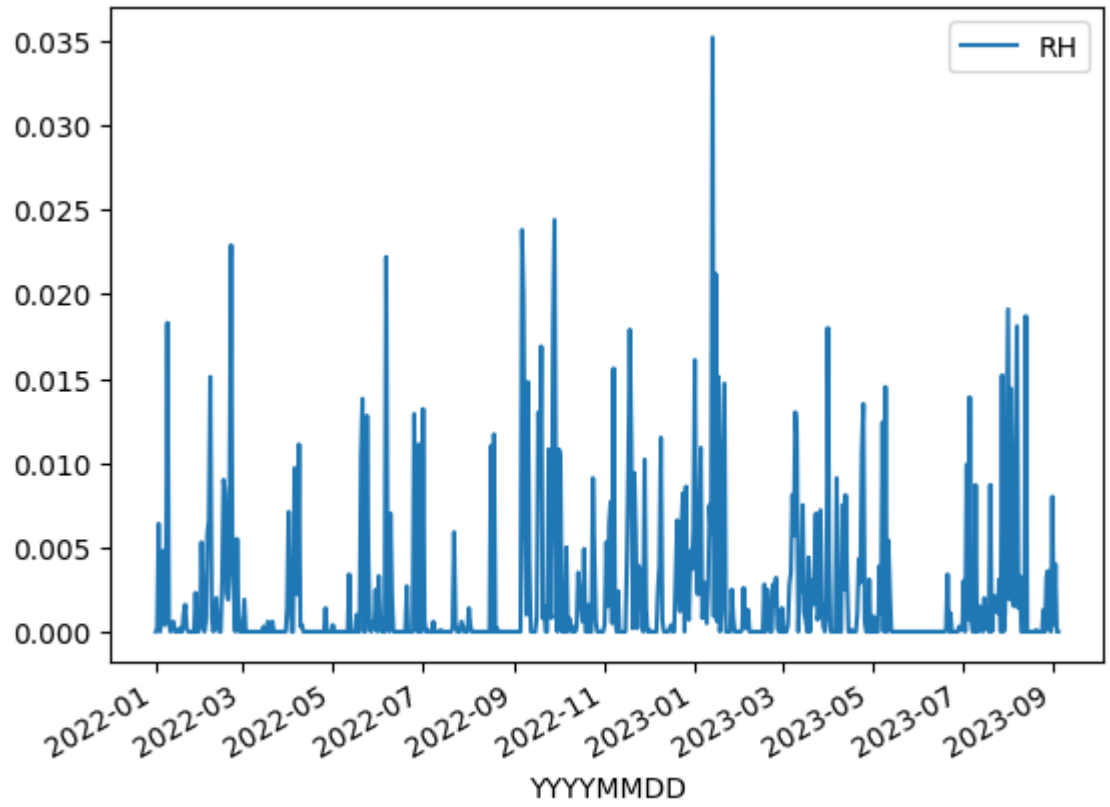
```
PrecipitationObs RH_ROTTERDAM
-----metadata------
name : RH_ROTTERDAM
x : 90061.9308913925
y : 441636.8542853104
filename :
source : KNMI
unit :
station : 344
meteo_var : RH
```

(continues on next page)

(continued from previous page)

```
-----time series------
                         RH
YYYYMMDD
2022-01-01 01:00:00  0.0000
2022-01-02 01:00:00  0.0002
2022-01-03 01:00:00  0.0064
2022-01-04 01:00:00  0.0000
2022-01-05 01:00:00  0.0008
```

[4]: `<Axes: xlabel='YYYYMMDD'>`



attributes

A `MeteoObs` object has the following attributes:

- `name`: station name and variable
- `x`: x-coordinate in m RD
- `y`: y-coordinate in m RD
- `station`: station number
- `unit`: measurement unit
- `meta`: dictionary with other metadata

[5]:
```
print(f"name: {o.name}")
print(f"x,y: {(o.x, o.y)}")
print(f"station: {o.station}")
print(f"unit", o.unit)
print("metadata:")
for key, item in o.meta.items():
    print(f"    {key}: {item}")
```

```
name: RH_ROTTERDAM
x,y: (90061.9308913925, 441636.8542853104)
station: 344
unit
metadata:
    LON_east: 4.447
    LAT_north: 51.962
    ALT_m: -4.3
    NAME: Rotterdam
    RH: Etmaalsom van de neerslag (in m) (0 voor <0.05mm) / Daily precipitation amount␣
↪(in m) (0 for <0.05mm)
```

Get KNMI data

There are 2 main method to obtain meteorological data:

1. `from_knmi`
2. `from_knmi_nearest_xy`

Below you can see how they can be called to obtain the precipitation data. Notice that they return the same data because station 344 is nearest to the given xy coordinates.

[6]:
```
o1 = hpd.PrecipitationObs.from_knmi(stn=344)
o2 = hpd.PrecipitationObs.from_knmi(xy=(90600, 442800))
o1.equals(o2)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable RHfrom None␣
↪to None
INFO:hydropandas.io.knmi:download knmi RH data from station 344-ROTTERDAM between None␣
↪and None
INFO:hydropandas.io.knmi:get KNMI data from station nearest to coordinates (90600,␣
↪442800) and meteovariable RH
INFO:hydropandas.io.knmi:download knmi RH data from station 344-ROTTERDAM between None␣
↪and None
```

[6]:
```
True
```

read options

The `MeteoObs.from_knmi` method contains the following keyword arguments:

- `stn`: station number.
- `startdate`: the start date of the time series you want, default is 1st of January 2019.
- `enddate`: the end date of the time series you want, default is today.

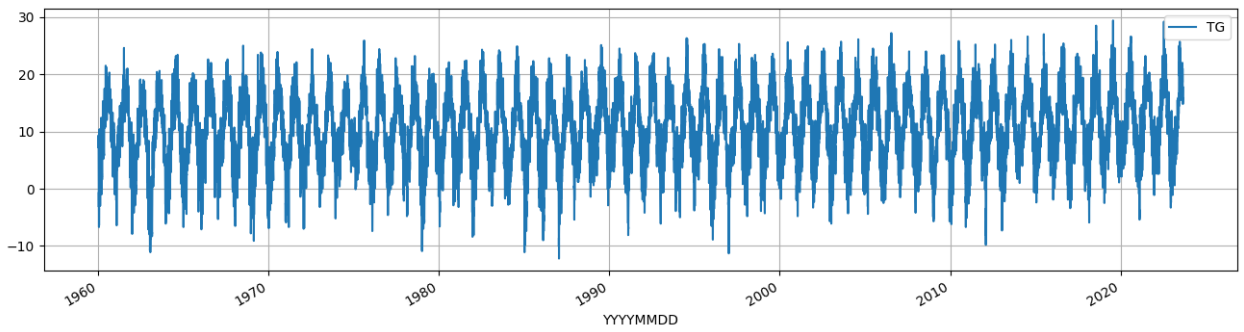
- `fill_missing_obs`: option to fill missing values with values from the nearest KNMI station. If measurements are filled an extra column is added to the time series in which the station number is shown that was used to fill a particular missing value.
- `interval`: time interval of the time series, default is 'daily'
- `raise_exception`: option to raise an error when the requested time series is empty. ***

The 3 examples below give a brief summary of these options

[7]:
```
# example 1 get daily average temperature (TG) from 1900 till now
o_t = hpd.MeteoObs.from_knmi("TG", stn=344, start="1960")
o_t.plot(figsize=(16, 4), grid=True)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable TGfrom 1960-
↪01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi TG data from station 344-ROTTERDAM between 1960-
↪01-01 00:00:00 and None
```

[7]:
```
<Axes: xlabel='YYYYMMDD'>
```



[8]:
```
# example 2 get daily average precipitation from 1972 with and without filling missing␣
↪measurements
o_rd = hpd.PrecipitationObs.from_knmi(
    "RD", stn=892, start="1972", end="2023", fill_missing_obs=False
)
o_rd.plot(figsize=(16, 4), grid=True)

o_rd_filled = hpd.PrecipitationObs.from_knmi(
    "RD", stn=892, start="1972", end="2023", fill_missing_obs=True
)

fig, ax = plt.subplots(figsize=(16, 4))
o_rd_filled.loc[o_rd_filled["station"] == "892", "RD"].plot(
    ax=ax, grid=True, label="oorspronkelijke metingen"
)
o_rd_filled.loc[o_rd_filled["station"] != "892", "RD"].plot(
    ax=ax, color="orange", label="opgevulde metingen"
)

ax.legend()
```

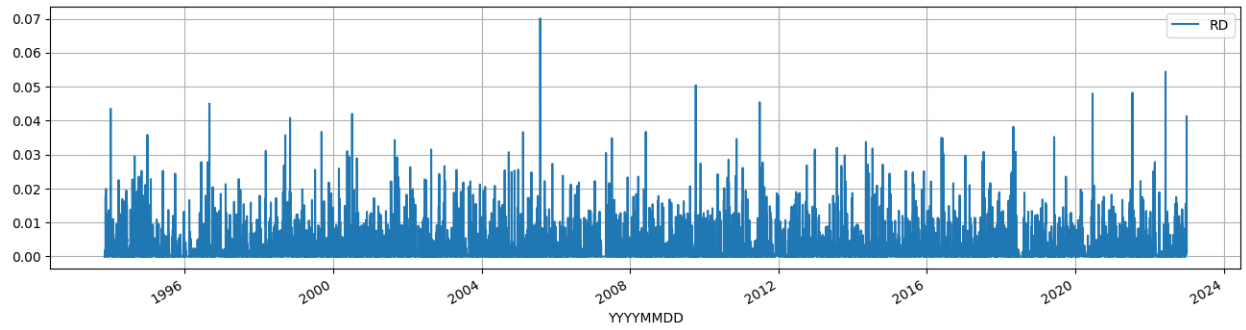
```
INFO:hydropandas.io.knmi:get KNMI data from station 892 and meteo variable RDfrom 1972-
↪01-01 00:00:00 to 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:download knmi RD data from station 892-GIERSBERGEN between 1972-
```

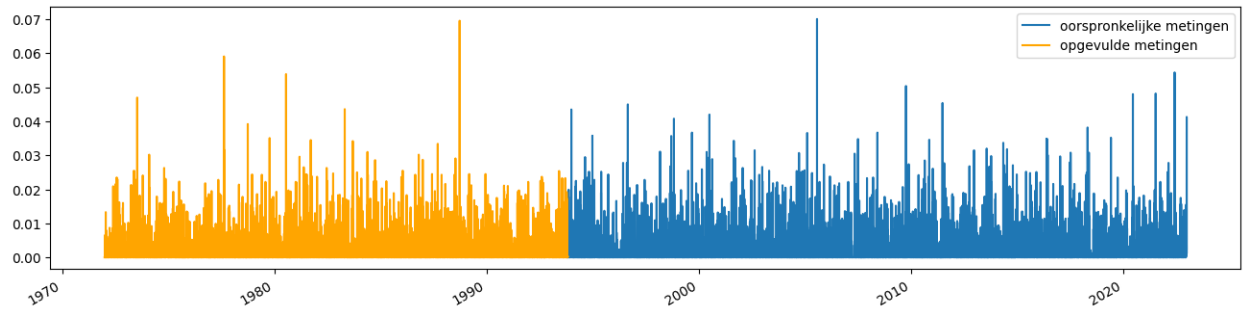
(continues on next page)

(continued from previous page)

```
↪01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 892 and meteo variable RDfrom 1972-
↪01-01 00:00:00 to 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:changing end_date to 2023-01-01
INFO:hydropandas.io.knmi:download knmi RD data from station 892-GIERSBERGEN between 1972-
↪01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:station 892 has no measurements before 1993-11-01 09:00:00
INFO:hydropandas.io.knmi:station 892 has 7975 missing measurements
INFO:hydropandas.io.knmi:trying to fill 7975 measurements with station [827]
INFO:hydropandas.io.knmi:download knmi RD data from station 827-TILBURG between 1972-01-
↪01 00:00:00 and 2023-01-01 00:00:00
```

[8]:
```
<matplotlib.legend.Legend at 0x1ba3821bac0>
```





[9]:
```
# see the station_opvulwaarde
display(o_rd.head())
display(o_rd_filled.head())
```

```
PrecipitationObs RD_GIERSBERGEN
-----metadata------
name : RD_GIERSBERGEN
x : 138500
y : 407500.0
filename :
source : KNMI
unit : m
station : 892
meteo_var : RD

-----time series------
                          RD
```

(continues on next page)

(continued from previous page)

```
YYYYMMDD
1993-11-01 09:00:00  0.0000
1993-11-02 09:00:00  0.0000
1993-11-03 09:00:00  0.0005
1993-11-04 09:00:00  0.0000
1993-11-05 09:00:00  0.0000
```

```
PrecipitationObs RD_GIERSBERGEN
-----metadata------
name : RD_GIERSBERGEN
x : 138500
y : 407500.0
filename :
source : KNMI
unit : m
station : 892
meteo_var : RD

-----time series------
                      RD station
1972-01-01 09:00:00  0.0     827
1972-01-02 09:00:00  0.0     827
1972-01-03 09:00:00  0.0     827
1972-01-04 09:00:00  0.0     827
1972-01-05 09:00:00  0.0     827
```

[10]:
```
# example 3 get evaporation
logging.getLogger().getEffectiveLevel()
logging.getLogger().setLevel(logging.INFO)

o_ev = hpd.EvaporationObs.from_knmi(
    stn=344, start="1972", end="2023", fill_missing_obs=True
)
o_ev
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable EV24from 1972-
↪01-01 00:00:00 to 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 344-ROTTERDAM between 1972-
↪01-01 00:00:00 and 2023-01-01 00:00:00
```

```
INFO:hydropandas.io.knmi:station 344 has no measurements before 1987-09-12 01:00:00
INFO:hydropandas.io.knmi:station 344 has 5809 missing measurements
INFO:hydropandas.io.knmi:trying to fill 5809 measurements with station [215]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 215-VOORSCHOTEN between␣
↪1972-01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:trying to fill 5809 measurements with station [330]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 330-HOEK-VAN-HOLLAND␣
↪between 1972-01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:trying to fill 5809 measurements with station [210]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 210-VALKENBURG between␣
↪1972-01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:trying to fill 5563 measurements with station [348]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 348-CABAUW-MAST between␣
```

(continues on next page)

(continued from previous page)

```
↪1972-01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:trying to fill 5499 measurements with station [240]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 240-SCHIPHOL between 1972-
↪01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:trying to fill 5499 measurements with station [356]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 356-HERWIJNEN between 1972-
↪01-01 00:00:00 and 2023-01-01 00:00:00
INFO:hydropandas.io.knmi:trying to fill 5499 measurements with station [260]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 260-DE-BILT between 1972-
↪01-01 00:00:00 and 2023-01-01 00:00:00
```

[10]:
```
EvaporationObs EV24_ROTTERDAM
-----metadata------
name : EV24_ROTTERDAM
x : 90061.9308913925
y : 441636.8542853104
filename :
source : KNMI
unit :
station : 344
meteo_var : EV24

-----time series------
                      EV24 station
1972-01-01 01:00:00  0.0002     260
1972-01-02 01:00:00  0.0002     260
1972-01-03 01:00:00  0.0002     260
1972-01-04 01:00:00  0.0000     260
1972-01-05 01:00:00  0.0000     260
...                     ...     ...
2022-12-28 01:00:00  0.0003     344
2022-12-29 01:00:00  0.0001     344
2022-12-30 01:00:00  0.0002     344
2022-12-31 01:00:00  0.0001     344
2023-01-01 01:00:00  0.0001     344

[18629 rows x 2 columns]
```

Get ObsCollections

It is also possible to read multiple `Observation` objects at once and store them in an `ObsCollection` object. For this we use the `ObsCollection.from_knmi()` method. Below an example to obtain precipitation (RH) and evaporation (EV24) from the KNMI station of Rotterdam and De Bilt.

[11]:
```
oc = hpd.read_knmi(stns=[344, 260], meteo_vars=["RH", "EV24"])
oc
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable RHfrom None␣
↪to None
INFO:hydropandas.io.knmi:download knmi RH data from station 344-ROTTERDAM between None␣
↪and None
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable RHfrom None␣
```

(continues on next page)

(continued from previous page)

```
↪to None
INFO:hydropandas.io.knmi:download knmi RH data from station 260-DE-BILT between None and␣
↪None
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable EV24from None␣
↪to None
INFO:hydropandas.io.knmi:download knmi EV24 data from station 344-ROTTERDAM between None␣
↪and None
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable EV24from None␣
↪to None
INFO:hydropandas.io.knmi:download knmi EV24 data from station 260-DE-BILT between None␣
↪and None
```

[11]:

```
                             x              y filename source unit  station  \
name
RH_ROTTERDAM     90061.930891  441636.854285            KNMI           344
RH_DE-BILT      140565.152506  456790.999119            KNMI           260
EV24_ROTTERDAM   90061.930891  441636.854285            KNMI           344
EV24_DE-BILT    140565.152506  456790.999119            KNMI           260

               meteo_var                                                 obs
name
RH_ROTTERDAM          RH  PrecipitationObs RH_ROTTERDAM
-----metadata---...
RH_DE-BILT            RH  PrecipitationObs RH_DE-BILT
-----metadata-----...
EV24_ROTTERDAM      EV24  EvaporationObs EV24_ROTTERDAM
-----metadata---...
EV24_DE-BILT        EV24  EvaporationObs EV24_DE-BILT
-----metadata-----...
```

Besides giving a list of stations it is also possible to: - specify `locations` as a dataframe with x, y coordinates (RD_new), the function will find the nearest KNMI station for every location. - specify `xmid` and `ymid` which are 2 arrays corresponding to a structured grid to obtain the nearest KNMI station for every cell in the grid.

[12]:

```
location = pd.DataFrame(index=["Rotterdam"], data={"x": 77500, "y": 399500})
hpd.read_knmi(locations=location, meteo_vars=["RH"])
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 340 and meteo variable RHfrom None␣
↪to None
INFO:hydropandas.io.knmi:download knmi RH data from station 340-WOENSDRECHT between None␣
↪and None
```

[12]:

```
                            x              y filename source unit  station  \
name
RH_WOENSDRECHT  82342.243033  384814.046707            KNMI           340

               meteo_var                                                 obs
name
RH_WOENSDRECHT        RH  PrecipitationObs RH_WOENSDRECHT
-----metadata-...
```

[13]:

```
hpd.read_knmi(xy=((77500, 399500),), meteo_vars=["RH"])
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 340 and meteo variable RHfrom None␣
↪to None
INFO:hydropandas.io.knmi:download knmi RH data from station 340-WOENSDRECHT between None␣
↪and None
```

[13]:
```
                          x             y filename source unit  station  \
name
RH_WOENSDRECHT  82342.243033  384814.046707            KNMI           340

               meteo_var                                                 obs
name
RH_WOENSDRECHT        RH  PrecipitationObs RH_WOENSDRECHT
-----metadata-...
```

Precipitation

The KNMI database has three different precipitation products: 1. Daily data from a meteorological station 2. Daily data from a neerslag (precipitation) station 3. Hourly data from a meteorological station

All three products can be obtained using the `from_knmi` method. Product 1 and 2 can also be accessed without the api.

If you want to access the data from a neerslag (precipitation) station you should add `stn_type='precipitation'` to the `PrecipitationObs.from_knmi()` method.

[14]:
```
# daily meteo station
precip1 = hpd.PrecipitationObs.from_knmi(stn=260, start="2010-1-1", end="2010-1-10")
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable RHfrom 2010-
↪01-01 00:00:00 to 2010-01-10 00:00:00
INFO:hydropandas.io.knmi:download knmi RH data from station 260-DE-BILT between 2010-01-
↪01 00:00:00 and 2010-01-10 00:00:00
```

[15]:
```
# daily neerslag station
precip2 = hpd.PrecipitationObs.from_knmi(
    meteo_var="RD", stn=550, start="2010-1-1", end="2010-1-10"
)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 550 and meteo variable RDfrom 2010-
↪01-01 00:00:00 to 2010-01-10 00:00:00
INFO:hydropandas.io.knmi:download knmi RD data from station 550-DE-BILT between 2010-01-
↪01 00:00:00 and 2010-01-10 00:00:00
```

[16]:
```
# hourly meteo station (only works with api)
precip3 = hpd.PrecipitationObs.from_knmi(
    stn=260,
    start="2010-1-1",
    end="2010-1-10",
    interval="hourly",
)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable RHfrom 2010-
↪01-01 00:00:00 to 2010-01-10 00:00:00
```

(continues on next page)

(continued from previous page)

```
INFO:hydropandas.io.knmi:download knmi RH data from station 260-DE-BILT between 2010-01-
↪01 00:00:00 and 2010-01-10 00:00:00
```

[17]:
```
# daily meteo station without api
precip4 = hpd.PrecipitationObs.from_knmi(
    stn=260,
    start="2010-1-1",
    end="2010-1-10",
    use_api=False,
)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable RHfrom 2010-
↪01-01 00:00:00 to 2010-01-10 00:00:00
INFO:hydropandas.io.knmi:download knmi RH data from station 260-DE-BILT between 2010-01-
↪01 00:00:00 and 2010-01-10 00:00:00
```

[18]:
```
# daily neerslag station without api
precip5 = hpd.PrecipitationObs.from_knmi(
    meteo_var="RD",
    stn=550,
    start="2010-1-1",
    end="2010-1-10",
    use_api=False,
)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 550 and meteo variable RDfrom 2010-
↪01-01 00:00:00 to 2010-01-10 00:00:00
INFO:hydropandas.io.knmi:download knmi RD data from station 550-DE-BILT between 2010-01-
↪01 00:00:00 and 2010-01-10 00:00:00
```

Below are the differences between the precipitation estimates from different station types.

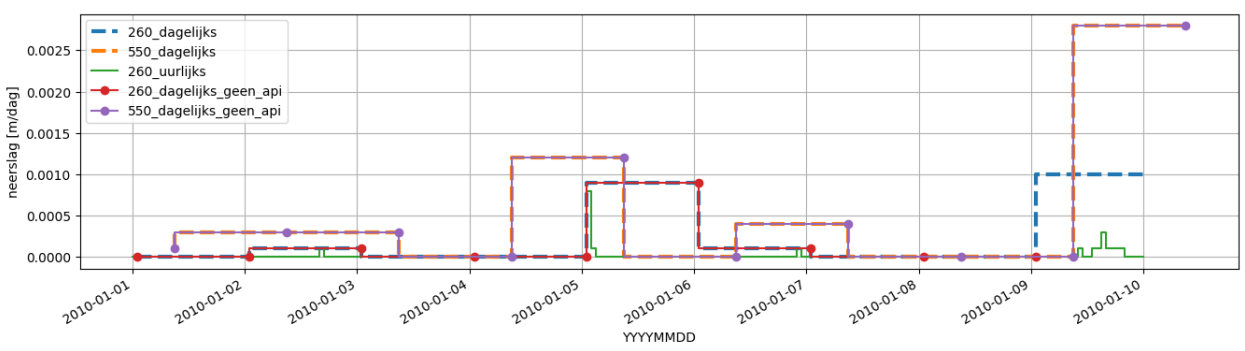
[19]:
```
fig, ax = plt.subplots(figsize=(16, 4))
precip1["RH"].plot(
    ax=ax,
    drawstyle="steps",
    ls="--",
    lw=3,
    label=str(precip1.station) + "_dagelijks",
)

precip2["RD"].plot(
    ax=ax,
    drawstyle="steps",
    ls="--",
    lw=3,
    label=str(precip2.station) + "_dagelijks",
)

precip3["RH"].plot(
    ax=ax,
    drawstyle="steps",
    label=str(precip3.station) + "_uurlijks",
```

(continues on next page)

(continued from previous page)

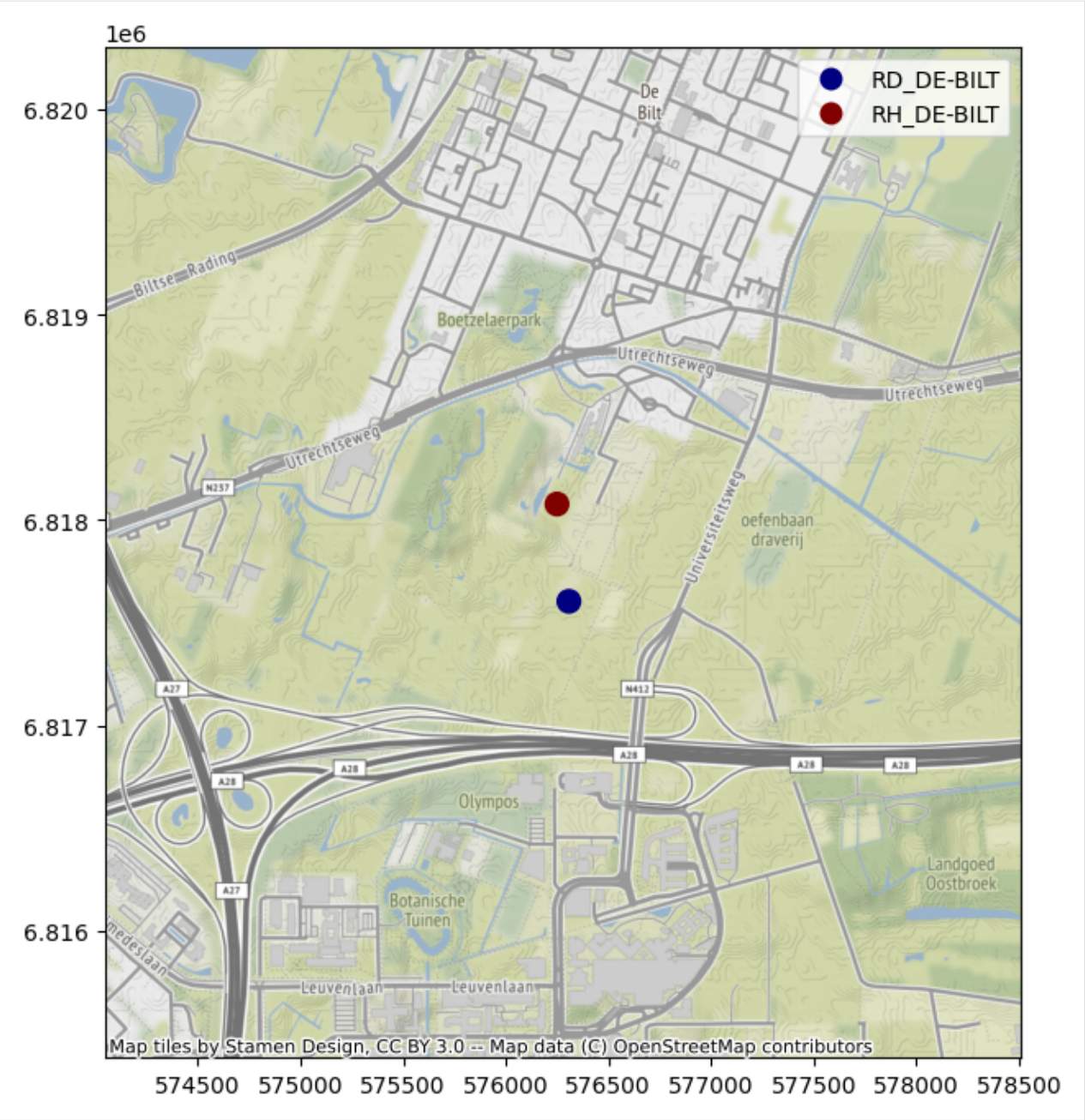
```
)

precip4["RH"].plot(
    ax=ax,
    drawstyle="steps",
    marker="o",
    label=str(precip4.station) + "_dagelijks_geen_api",
)

precip5["RD"].plot(
    ax=ax,
    drawstyle="steps",
    marker="o",
    label=str(precip5.station) + "_dagelijks_geen_api",
)


ax.legend()
ax.grid()
ax.set_ylabel("neerslag [m/dag]")
```

[19]:
```
Text(0, 0.5, 'neerslag [m/dag]')
```



The locations of the stations can be plotted onto a map using the contextily package.

[20]:
```
import contextily as cx

oc = hpd.ObsCollection([precip1, precip2])
gdf = oc.to_gdf()
gdf = gdf.set_crs(28992)
gdf["name"] = gdf.index
ax = gdf.buffer(2000).plot(alpha=0, figsize=(8, 8))
gdf.plot("name", ax=ax, cmap="jet", legend=True, markersize=100)
cx.add_basemap(ax, crs=28992)
```



Evaporation

KNMI provides the Makking reference evaporation (meteo_var EV24). Hydropandas provides a posssibility to calculate three different types of reference evaporation from data of KNMI meteo stations: - Penman - Hargreaves - Makkink (in the same way as KNMI)

These three types of reference evaporation are calculated the same way as described by Allen et al. 1990 and STOWA rapport. Be aware that the last report is written in Dutch and contains errors in the units.

The following variables from the KNMI are used for each reference evaporation type: - Penman: average (TG), minimum (TN) and maximum (TX) temperature, de global radiation (Q), de windspeed (FG) en de relative humidity (PG).

- Hargreaves: average (TG), minimum (TN) and maximum (TX) temperature - Makkink: average temperature (TG) and global radiation (Q)

Comparison Makkink

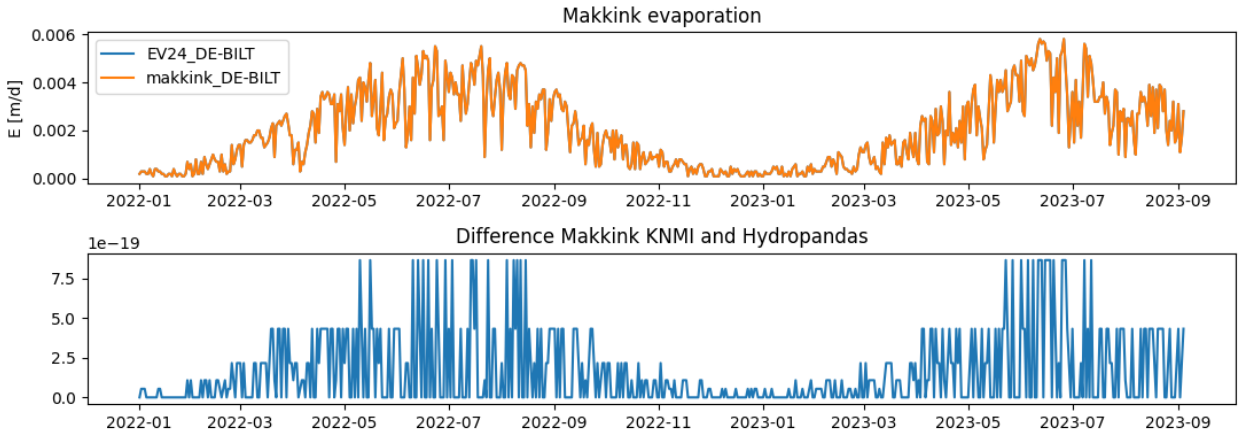
Lets compare Hypdropandas Makkink verdamping evaporation with the EV24 Makkink verdamping of the KNMI. When Hydropandas Makkink evaporation is rounded (on 4 decimals), the estimate is the same as for the KNMI estimate.

[21]:
```
ev24 = hpd.EvaporationObs.from_knmi(
    stn=260, start="2022-1-1"
)  # et_type='EV24' by default
makk = hpd.EvaporationObs.from_knmi(meteo_var="makkink", stn=260, start="2022-1-1")
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable EV24from 2022-
↪01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi EV24 data from station 260-DE-BILT between 2022-
↪01-01 00:00:00 and None
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable makkinkfrom␣
↪2022-01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi TG data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi Q data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
```

[22]:
```
f, ax = plt.subplots(2, figsize=(11, 4))
ax[0].plot(ev24, label=ev24.name)
ax[0].plot(makk.round(4), label=makk.name)
ax[0].set_ylabel("E [m/d]")
ax[0].set_title("Makkink evaporation")
ax[0].legend()
ax[1].plot(ev24["EV24"] - makk["makkink"].round(4))
ax[1].set_title("Difference Makkink KNMI and Hydropandas")
f.tight_layout()
```



Comparison Penman, Makkink en Hargreaves

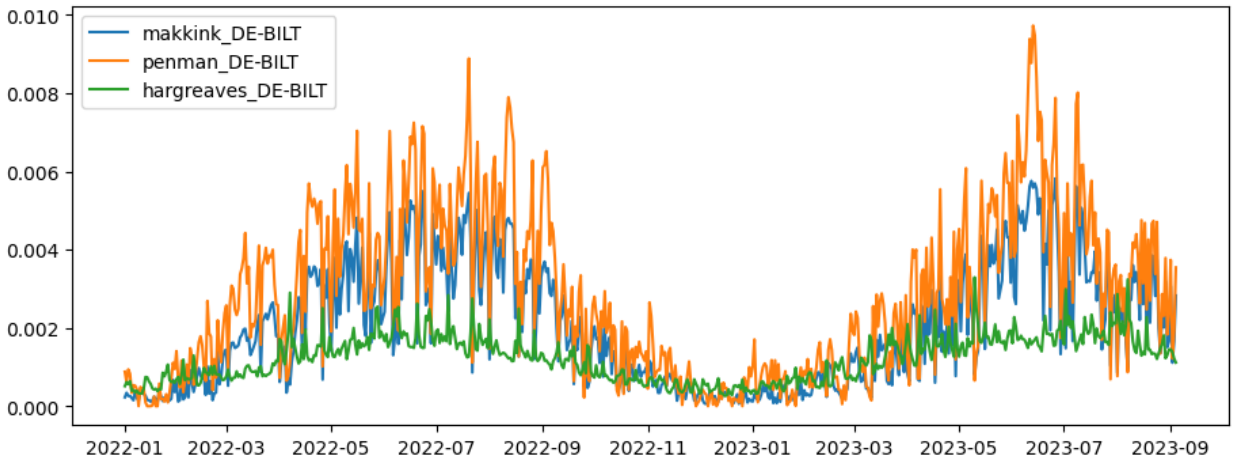
On average Penman gives a higher estimate for reference evaporation than Makkink (~0.55mm). This can be explained by the fact that Penman takes into account windspeed and Makkink ignores this proces. Hargreaves is a very simple way of estimation the evaporation, only taking into account temperature and extraterrestial radiation. Therefore it gives a lower estimate for the reference evporatoin compared to the two other methods (~-0.35mm wrt Makkink).

[23]:
```
penm = hpd.EvaporationObs.from_knmi(
    meteo_var="penman", stn=260, start="2022-1-1"
).squeeze()
harg = hpd.EvaporationObs.from_knmi(
    meteo_var="hargreaves", stn=260, start="2022-1-1"
).squeeze()
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable penmanfrom␣
↪2022-01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi TG data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi TN data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi TX data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi Q data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi FG data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi UG data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable␣
↪hargreavesfrom 2022-01-01 00:00:00 to None
INFO:hydropandas.io.knmi:download knmi TG data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi TN data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
INFO:hydropandas.io.knmi:download knmi TX data from station 260-DE-BILT between 2022-01-
↪01 00:00:00 and None
```

[24]:
```
f, ax = plt.subplots(figsize=(11, 4))
ax.plot(makk, label=makk.name)
ax.plot(penm, label=penm.name)
ax.plot(harg, label=harg.name)
ax.legend()
```

[24]:
```
<matplotlib.legend.Legend at 0x1ba3bf49030>
```



Spatial interpolate (Makkink) Evaporation?

Does interpolation matter? There are ways to interpolate evaporation datasets. However currently the nearest station is always used in HydroPandas' methods. Does this give different results? First lets look spatially.

Get all stations where EV24 is measured

[25]:
```
stns = knmi.get_stations(meteo_var="EV24").sort_index()
```

Collect all EV24 data ever measured by KNMI

[26]:
```
tmin = "1950-01-01"
tmax = "2022-04-11"

# empty dataframe
df = pd.DataFrame(
    columns=stns.index
)  # index=pd.date_range(start=tmin, end=tmax, freq='H')

for stn in tqdm(stns.index):
    df_stn = hpd.MeteoObs.from_knmi(
        meteo_var="EV24", stn=stn, fill_missing_obs=False, start=tmin, end=tmax
    )
    df[stn] = df_stn

df
```

```
  0%|          | 0/36 [00:00<?, ?it/s]
INFO:hydropandas.io.knmi:get KNMI data from station 210 and meteo variable EV24from 1950-01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 210-VALKENBURG between 1950-01-01 00:00:00 and 2022-04-11 00:00:00
  3%|         | 1/36 [00:02<01:19,  2.27s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 215 and meteo variable EV24from 1950-01-01 00:00:00 to 2022-04-11 00:00:00
```

(continues on next page)

(continued from previous page)

```
INFO:hydropandas.io.knmi:download knmi EV24 data from station 215-VOORSCHOTEN between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
  6%|         | 2/36 [00:03<00:57,  1.70s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 235 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 235-DE-KOOY between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
  8%|         | 3/36 [00:05<01:04,  1.96s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 240 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 240-SCHIPHOL between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 11%|         | 4/36 [00:07<01:01,  1.91s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 242 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 242-VLIELAND between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 242-VLIELAND between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
 14%|        | 5/36 [00:08<00:48,  1.55s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 249 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 249-BERKHOUT between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 17%|        | 6/36 [00:09<00:40,  1.36s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 251 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 251-HOORN-TERSCHELLING␣
↪between 1950-01-01 00:00:00 and 2022-04-11 00:00:00
 19%|        | 7/36 [00:10<00:36,  1.24s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 257 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 257-WIJK-AAN-ZEE between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 22%|       | 8/36 [00:11<00:31,  1.12s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 260-DE-BILT between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 25%|       | 9/36 [00:13<00:40,  1.51s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 265 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 265-SOESTERBERG between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
```

```
 28%|       | 10/36 [00:15<00:36,  1.42s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 267 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 267-STAVOREN between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 31%|       | 11/36 [00:16<00:34,  1.39s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 269 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 269-LELYSTAD between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 33%|      | 12/36 [00:17<00:32,  1.36s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 270 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 270-LEEUWARDEN between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 36%|      | 13/36 [00:19<00:37,  1.62s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 273 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 273-MARKNESSE between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 39%|      | 14/36 [00:21<00:36,  1.64s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 275 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 275-DEELEN between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
 42%|     | 15/36 [00:23<00:36,  1.74s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 277 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 277-LAUWERSOOG between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 44%|     | 16/36 [00:25<00:35,  1.78s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 278 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 278-HEINO between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
 47%|     | 17/36 [00:27<00:34,  1.83s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 279 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 279-HOOGEVEEN between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 50%|     | 18/36 [00:29<00:34,  1.93s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 280 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 280-EELDE between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
```

```
 53%|    | 19/36 [00:31<00:34,  2.00s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 283 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 283-HUPSEL between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
 56%|    | 20/36 [00:33<00:33,  2.09s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 286 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 286-NIEUW-BEERTA between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 58%|    | 21/36 [00:36<00:31,  2.08s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 290 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 290-TWENTHE between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 61%|    | 22/36 [00:38<00:28,  2.06s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 310 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 310-VLISSINGEN between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 64%|   | 23/36 [00:40<00:27,  2.11s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 319 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 319-WESTDORPE between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 67%|   | 24/36 [00:42<00:23,  2.00s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 323 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 323-WILHELMINADORP between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 69%|   | 25/36 [00:43<00:20,  1.90s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 330 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 330-HOEK-VAN-HOLLAND␣
↪between 1950-01-01 00:00:00 and 2022-04-11 00:00:00
 72%|  | 26/36 [00:45<00:19,  1.91s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 340 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 340-WOENSDRECHT between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 340-WOENSDRECHT between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 75%|  | 27/36 [00:47<00:16,  1.78s/it]
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 344-ROTTERDAM between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 78%|  | 28/36 [00:49<00:16,  2.08s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 348 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 348-CABAUW-MAST between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 81%|  | 29/36 [00:51<00:14,  2.08s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 350 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 350-GILZE-RIJEN between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 83%| | 30/36 [00:54<00:13,  2.22s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 356 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 356-HERWIJNEN between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 86%| | 31/36 [00:56<00:10,  2.01s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 370 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 370-EINDHOVEN between 1950-
↪01-01 00:00:00 and 2022-04-11 00:00:00
 89%| | 32/36 [00:58<00:08,  2.10s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 375 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 375-VOLKEL between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
 92%|| 33/36 [01:00<00:06,  2.06s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 377 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 377-ELL between 1950-01-01␣
↪00:00:00 and 2022-04-11 00:00:00
 94%|| 34/36 [01:01<00:03,  1.89s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 380 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 380-MAASTRICHT between␣
↪1950-01-01 00:00:00 and 2022-04-11 00:00:00
 97%|| 35/36 [01:04<00:02,  2.11s/it]
INFO:hydropandas.io.knmi:get KNMI data from station 391 and meteo variable EV24from 1950-
↪01-01 00:00:00 to 2022-04-11 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 391-ARCEN between 1950-01-
↪01 00:00:00 and 2022-04-11 00:00:00
```

```
100%|| 36/36 [01:06<00:00,  1.86s/it]
```

[26]:

```
                         210     215     235     240  242     249     251  \
YYYYMMDD
1987-03-26 01:00:00  0.0006     NaN  0.0006     NaN  NaN     NaN     NaN
1987-03-27 01:00:00  0.0016     NaN  0.0015     NaN  NaN     NaN     NaN
1987-03-28 01:00:00  0.0008     NaN  0.0007     NaN  NaN     NaN     NaN
1987-03-29 01:00:00  0.0013     NaN  0.0007     NaN  NaN     NaN     NaN
1987-03-30 01:00:00  0.0019     NaN  0.0021     NaN  NaN     NaN     NaN
...                     ...     ...     ...     ...  ...     ...     ...
2016-04-29 01:00:00  0.0023  0.0022  0.0020  0.0020  NaN  0.0015  0.0018
2016-04-30 01:00:00  0.0023  0.0021  0.0025  0.0020  NaN  0.0022  0.0027
2016-05-01 01:00:00  0.0023  0.0022  0.0029  0.0022  NaN  0.0021  0.0028
2016-05-02 01:00:00  0.0036  0.0035  0.0036  0.0034  NaN  0.0034  0.0035
2016-05-03 01:00:00  0.0029  0.0028  0.0022  0.0029  NaN  0.0028  0.0026

                         257     260  265  ...  340     344     348     350  \
YYYYMMDD                                  ...
1987-03-26 01:00:00      NaN  0.0005  NaN  ...  NaN     NaN     NaN     NaN
1987-03-27 01:00:00      NaN  0.0016  NaN  ...  NaN     NaN  0.0016     NaN
1987-03-28 01:00:00      NaN  0.0007  NaN  ...  NaN     NaN  0.0007     NaN
1987-03-29 01:00:00      NaN  0.0010  NaN  ...  NaN     NaN     NaN     NaN
1987-03-30 01:00:00      NaN  0.0016  NaN  ...  NaN     NaN  0.0017     NaN
...                      ...     ...  ...  ...  ...     ...     ...     ...
2016-04-29 01:00:00  0.0014  0.0022  NaN  ...  NaN  0.0026  0.0023  0.0018
2016-04-30 01:00:00  0.0022  0.0014  NaN  ...  NaN  0.0021  0.0016  0.0014
2016-05-01 01:00:00  0.0026  0.0023  NaN  ...  NaN  0.0025  0.0022  0.0021
2016-05-02 01:00:00  0.0035  0.0035  NaN  ...  NaN  0.0036  0.0036  0.0035
2016-05-03 01:00:00  0.0027  0.0034  NaN  ...  NaN  0.0031  0.0033  0.0034

                         356     370     375     377     380     391
YYYYMMDD
1987-03-26 01:00:00      NaN  0.0008     NaN     NaN  0.0010     NaN
1987-03-27 01:00:00      NaN  0.0017     NaN     NaN  0.0024     NaN
1987-03-28 01:00:00      NaN  0.0005     NaN     NaN  0.0008     NaN
1987-03-29 01:00:00      NaN  0.0013     NaN     NaN  0.0014     NaN
1987-03-30 01:00:00      NaN  0.0015     NaN     NaN  0.0018     NaN
...                      ...     ...     ...     ...     ...     ...
2016-04-29 01:00:00  0.0022  0.0024  0.0019  0.0025  0.0025  0.0023
2016-04-30 01:00:00  0.0012  0.0015  0.0014  0.0010  0.0010  0.0009
2016-05-01 01:00:00  0.0021  0.0022  0.0021  0.0009  0.0011  0.0015
2016-05-02 01:00:00  0.0034  0.0031  0.0030  0.0029  0.0031  0.0031
2016-05-03 01:00:00  0.0034  0.0035  0.0034  0.0034  0.0036  0.0035

[10632 rows x 36 columns]
```

According to the KNMI, thin plate spline is the best way to interpolate Makkink evaporation. Thats also how they provide the gridded Makkink evaporation :

- Evaporation Dataset - gridded daily Makkink evaporation for the Netherlands
- Interpolation of Makkink evaporation in the Netherlands - Paul Hiemstra and Raymond Sluiter (2011)

[28]:

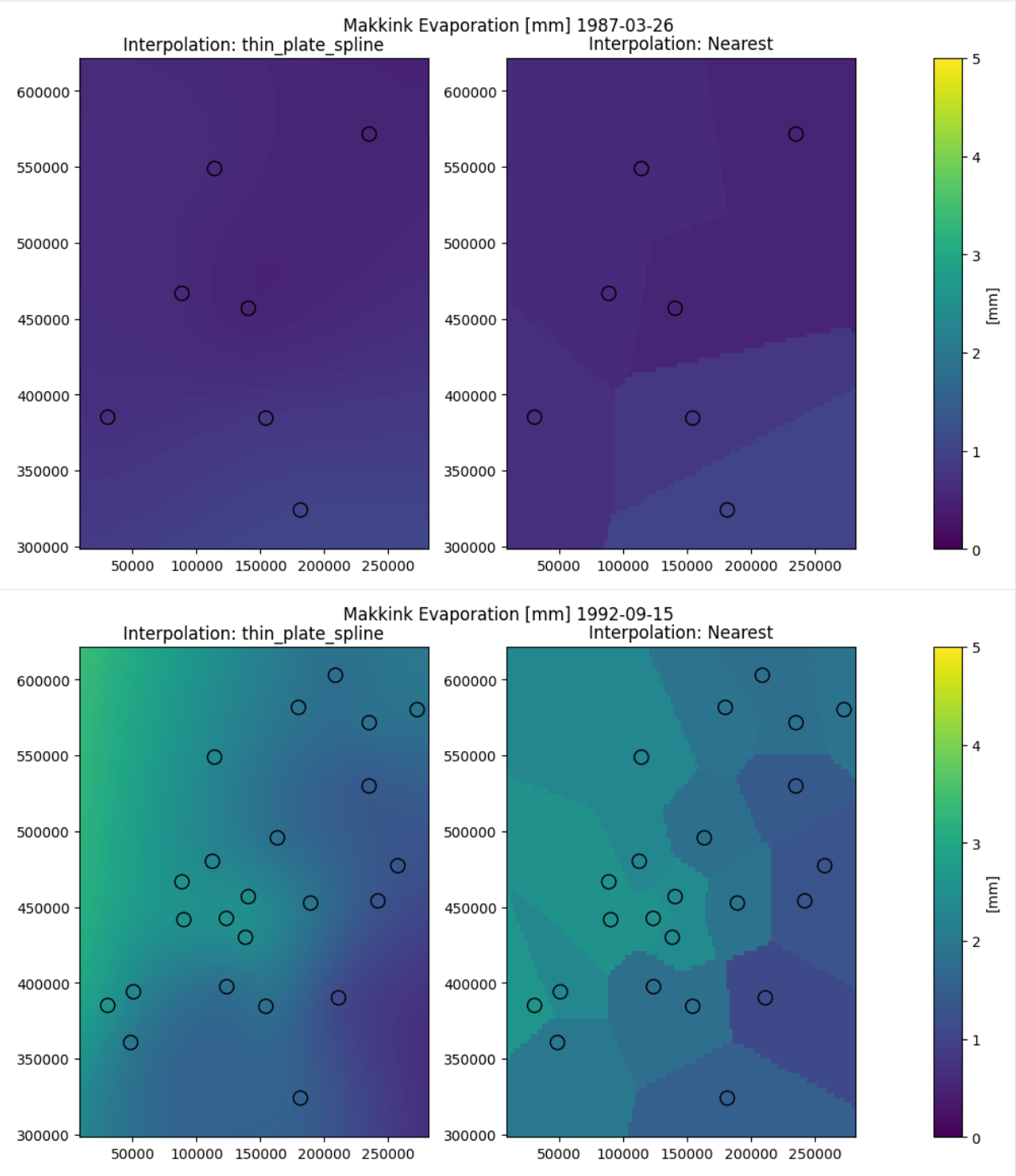
```
xy = stns.loc[df.columns, ["x", "y"]]
```

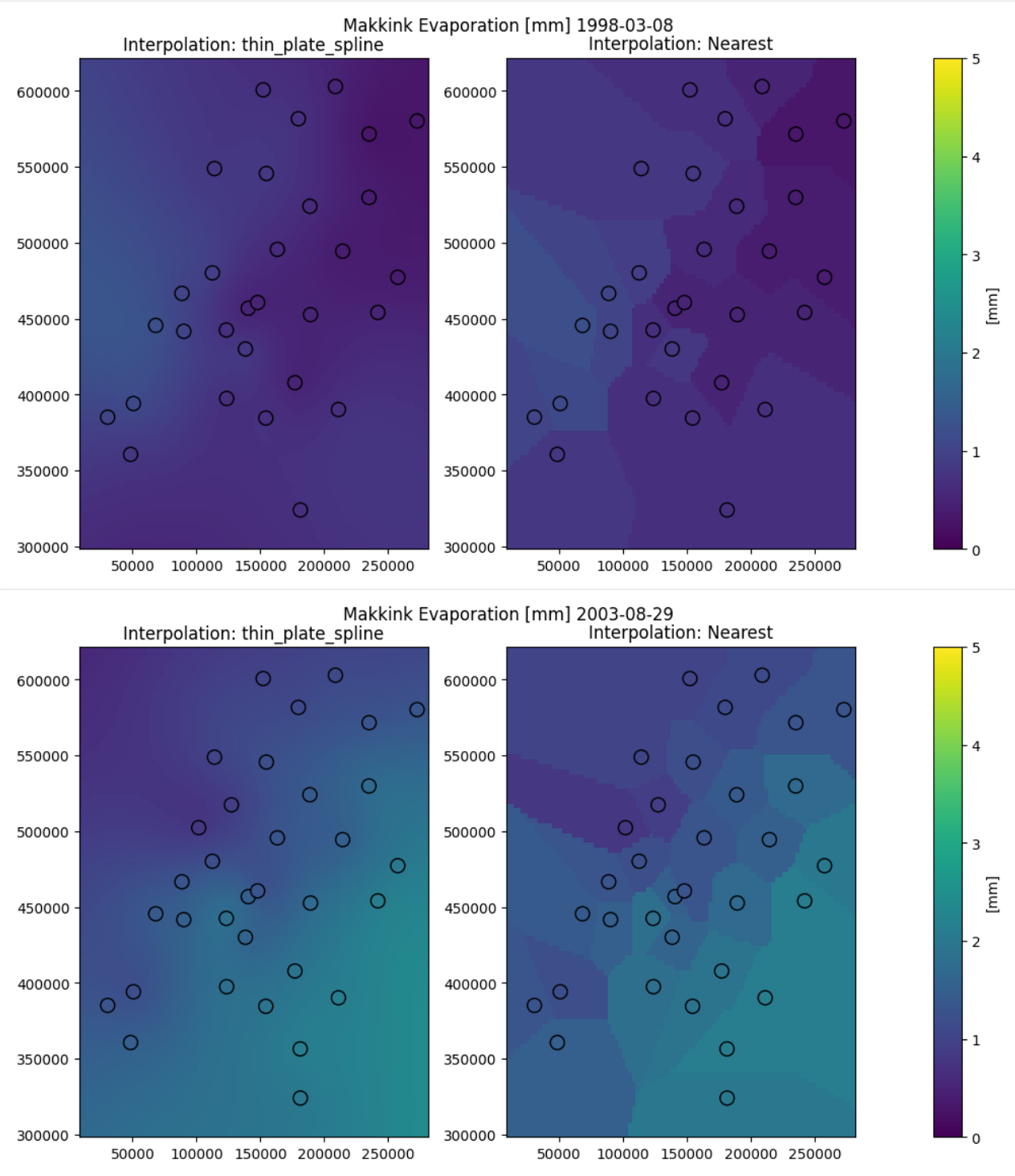
(continues on next page)

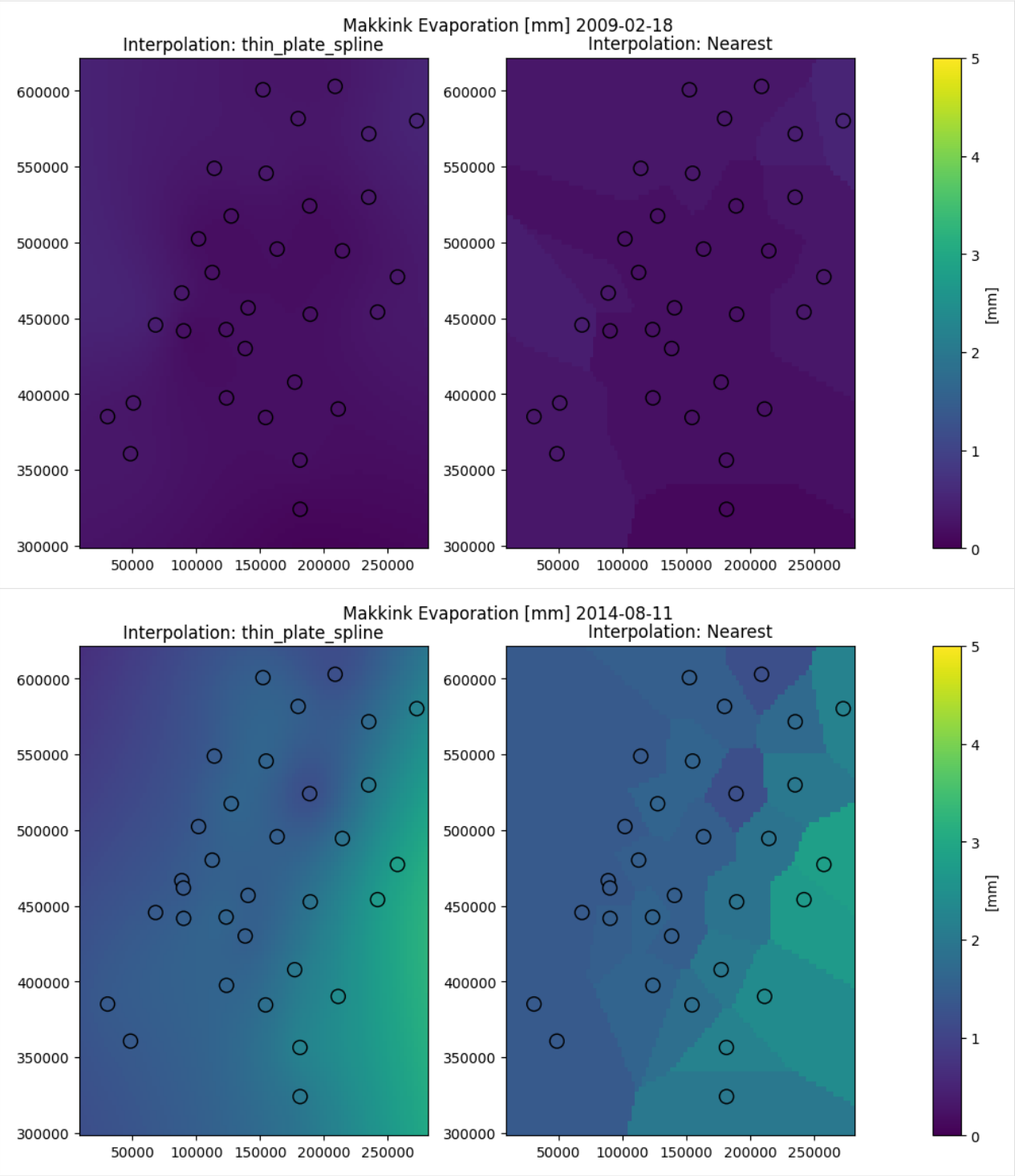
(continued from previous page)

```
for idx in tqdm(df.index[0 : len(df) : 2000]):
    # get all stations with values for this date
    val = df.loc[idx].dropna() * 1000  # mm
    # get stations for this date
    coor = xy.loc[val.index].to_numpy()
    if (
        len(val) < 3
    ):  # if there are less than 3 stations, thin plate spline does not work
        # options: linear, multiquadric, gaussian,
        kernel = "linear"

    else:
        kernel = "thin_plate_spline"
        # options:
        # 'inverse_quadratic', 'linear', 'multiquadric', 'gaussian',
        # 'inverse_multiquadric', 'cubic', 'quintic', 'thin_plate_spline'

    # create an scipy interpolator
    rbf = RBFInterpolator(coor, val.to_numpy(), epsilon=1, kernel=kernel)

    nea = NearestNDInterpolator(coor, val.to_numpy())

    # interpolate on grid of the Netherlands
    grid = np.mgrid[10000:280000:100j, 300000:620000:100j]
    grid2 = grid.reshape(2, -1).T  # interpolator only takes array [[x0, y0],
    #  [x1, y1]]
    val_rbf = rbf.__call__(grid2).reshape(100, 100)
    val_nea = nea.__call__(grid2).reshape(100, 100)

    # create figure
    fig, ax = plt.subplot_mosaic("AAAABBBBC", figsize=(10, 5.925))
    fig.suptitle(f"Makkink Evaporation [mm] {idx.date()}", y=0.95)
    vmin = 0
    vmax = 5

    ax["A"].set_title(f"Interpolation: {kernel}")
    ax["A"].pcolormesh(grid[0], grid[1], val_rbf, vmin=vmin, vmax=vmax)
    ax["B"].set_title(f"Interpolation: Nearest")
    ax["B"].pcolormesh(grid[0], grid[1], val_nea, vmin=vmin, vmax=vmax)
    ax["A"].scatter(*coor.T, c=val, s=100, ec="k", vmin=vmin, vmax=vmax)
    p = ax["B"].scatter(*coor.T, c=val, s=100, ec="k", vmin=vmin, vmax=vmax)
    cb = fig.colorbar(p, cax=ax["C"])
    cb.set_label("[mm]")
    fig.tight_layout()
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
100%|| 6/6 [00:01<00:00,  4.51it/s]
```


Makkink Evaporation [mm] 1987-03-26
Interpolation: thin_plate_spline
Interpolation: Nearest
600000
550000
500000
450000
400000
350000
300000
50000
100000
150000
200000
250000
[mm]
0
1
2
3
4
5
Makkink Evaporation [mm] 1992-09-15
Interpolation: thin_plate_spline
Interpolation: Nearest


Makkink Evaporation [mm] 1998-03-08
Interpolation: thin_plate_spline
Interpolation: Nearest
Makkink Evaporation [mm] 2003-08-29
Interpolation: thin_plate_spline
Interpolation: Nearest
600000
550000
500000
450000
400000
350000
300000
50000
100000
150000
200000
250000
[mm]
0
1
2
3
4
5



The same method is implemented in Hydropandas for an ObsCollection.

[29]:
```
sd = "2022-01-01"
ed = "2022-12-31"
oc = hpd.read_knmi(stns=stns.index, starts=sd, ends=ed, meteo_vars=["EV24"])
oc_et = oc.interpolate(xy=[(100000, 330000)])
```

(continues on next page)

(continued from previous page)

```
eti = oc_et.iloc[0].obs
eti
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 210 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 210-VALKENBURG between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 210-VALKENBURG between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 215 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 215-VOORSCHOTEN between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 235 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 235-DE-KOOY between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 240 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 240-SCHIPHOL between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 242 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 242-VLIELAND between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 242-VLIELAND between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 249 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 249-BERKHOUT between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 251 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 251-HOORN-TERSCHELLING␣
↪between 2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 257 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 257-WIJK-AAN-ZEE between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 260 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 260-DE-BILT between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 265 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 265-SOESTERBERG between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 265-SOESTERBERG between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 267 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 267-STAVOREN between 2022-
```

(continues on next page)

(continued from previous page)

```
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 269 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 269-LELYSTAD between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 270 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 270-LEEUWARDEN between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 273 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 273-MARKNESSE between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 275 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 275-DEELEN between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 277 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 277-LAUWERSOOG between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 278 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 278-HEINO between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 279 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 279-HOOGEVEEN between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 280 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 280-EELDE between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 283 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 283-HUPSEL between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 286 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 286-NIEUW-BEERTA between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 290 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 290-TWENTHE between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 310 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 310-VLISSINGEN between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 319 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 319-WESTDORPE between 2022-
```

(continues on next page)

(continued from previous page)

```
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 323 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 323-WILHELMINADORP between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 330 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 330-HOEK-VAN-HOLLAND␣
↪between 2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 340 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 340-WOENSDRECHT between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 340-WOENSDRECHT between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 344 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 344-ROTTERDAM between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 348 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 348-CABAUW-MAST between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 350 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 350-GILZE-RIJEN between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 356 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 356-HERWIJNEN between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 370 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 370-EINDHOVEN between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 375 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 375-VOLKEL between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 377 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 377-ELL between 2022-01-01␣
↪00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 380 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 380-MAASTRICHT between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 391 and meteo variable EV24from 2022-
↪01-01 00:00:00 to 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 391-ARCEN between 2022-01-
↪01 00:00:00 and 2022-12-31 00:00:00
```

[29]:
```
EvaporationObs 100000_330000
-----metadata------
name : 100000_330000
x : 100000
y : 330000
filename :
source : interpolation
unit :
station : nan
meteo_var : EV24

-----time series------
                     100000_330000
YYYYMMDD
2022-01-01 01:00:00       0.000254
2022-01-02 01:00:00       0.000570
2022-01-03 01:00:00       0.000330
2022-01-04 01:00:00       0.000142
2022-01-05 01:00:00       0.000157
...                            ...
2022-12-27 01:00:00       0.000151
2022-12-28 01:00:00       0.000260
2022-12-29 01:00:00       0.000101
2022-12-30 01:00:00       0.000141
2022-12-31 01:00:00       0.000077

[365 rows x 1 columns]
```

[30]:
```
etn = hpd.MeteoObs.from_knmi(
    xy=(100000, 330000), start=sd, end=ed, meteo_var="EV24", fill_missing_obs=True
)
etn
```

```
INFO:hydropandas.io.knmi:get KNMI data from station nearest to coordinates (100000,␣
↪330000) and meteovariable EV24
INFO:hydropandas.io.knmi:download knmi EV24 data from station 340-WOENSDRECHT between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 340-WOENSDRECHT between 2022-
↪01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:station 340 has no measurements between 2022-01-01 00:00:00 and␣
↪2022-12-31 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi EV24 data from station 323-WILHELMINADORP between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
INFO:hydropandas.io.knmi:station 323 has 4 missing measurements
INFO:hydropandas.io.knmi:trying to fill 4 measurements with station [310]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 310-VLISSINGEN between␣
↪2022-01-01 00:00:00 and 2022-12-31 00:00:00
```

[30]:
```
MeteoObs EV24_WILHELMINADORP
-----metadata------
name : EV24_WILHELMINADORP
x : 50663.8174359615
```

(continues on next page)

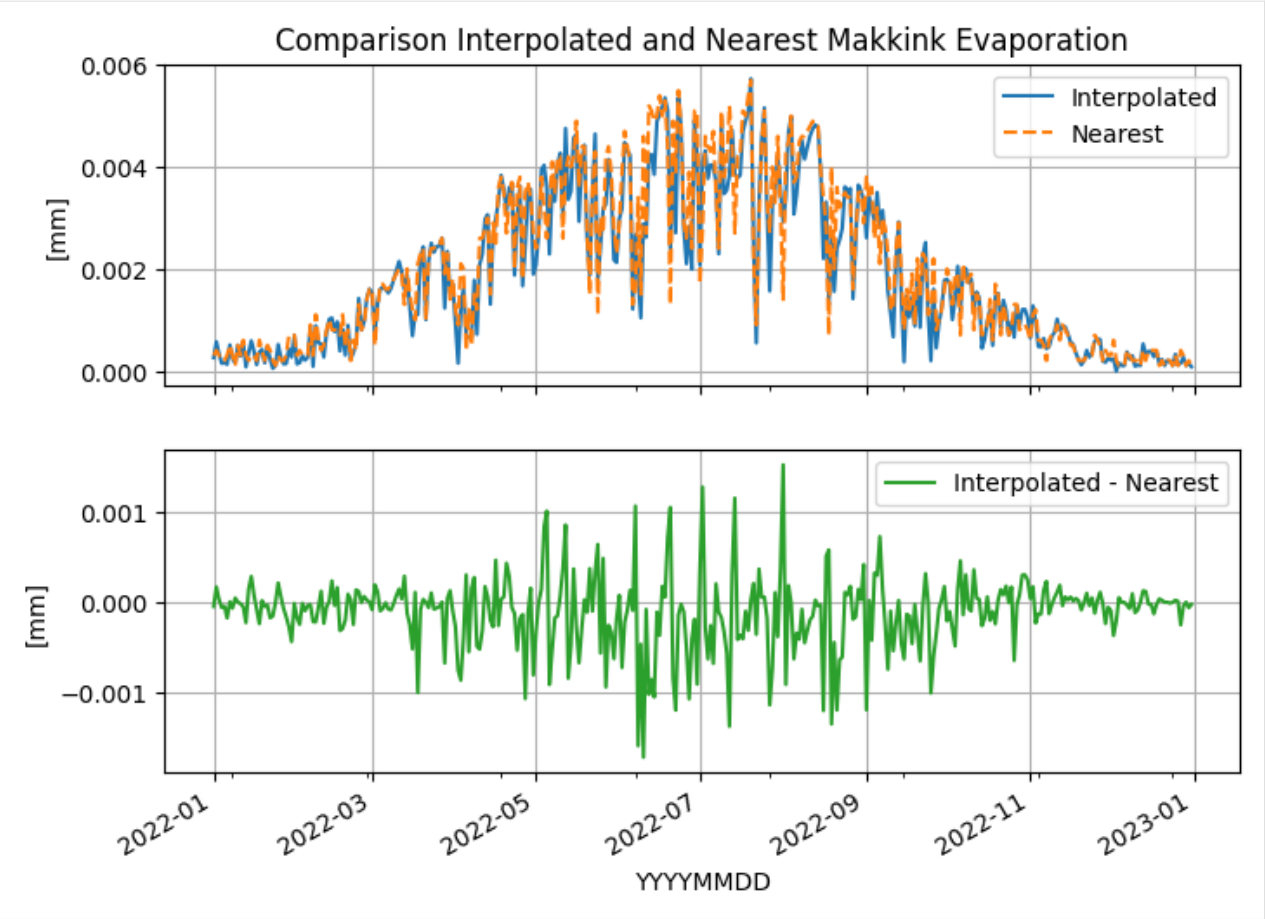
(continued from previous page)

```
y : 394074.1764431248
filename :
source : KNMI
unit :
station : 323
meteo_var : EV24

-----time series------
                      EV24 station
2022-01-01 01:00:00  0.0003     323
2022-01-02 01:00:00  0.0004     323
2022-01-03 01:00:00  0.0003     323
2022-01-04 01:00:00  0.0002     323
2022-01-05 01:00:00  0.0002     323
...                     ...     ...
2022-12-27 01:00:00  0.0004     323
2022-12-28 01:00:00  0.0003     323
2022-12-29 01:00:00  0.0001     323
2022-12-30 01:00:00  0.0002     323
2022-12-31 01:00:00  0.0001     323

[365 rows x 2 columns]
```

As can be seen, for one location the interpolation method is significantly slower. Lets see how the values compare for a time series.

[31]:
```
fig, ax = plt.subplots(2, 1, figsize=(8, 6), sharex=True)
eti.plot(ax=ax[0])
etn.plot(ax=ax[0], linestyle="--", color="C1")
ax[0].set_title("Comparison Interpolated and Nearest Makkink Evaporation")
ax[0].set_ylabel("[mm]")
ax[0].grid()
ax[0].legend(["Interpolated", "Nearest"])

(eti.squeeze() - etn["EV24"].squeeze()).plot(ax=ax[1], color="C2")
ax[1].set_ylabel("[mm]")
ax[1].grid()
ax[1].legend(["Interpolated - Nearest"])
```

[31]:
```
<matplotlib.legend.Legend at 0x1ba3ca4fa00>
```



The interpolated evaporation can also be collected for multiple points (using x and y in a list of DataFrame) in an ObsCollection

[32]:
```
oc_et = oc.interpolate(xy=[(100000, 320000), (100000, 330000), (100000, 340000)])
oc_et
```

[32]:
```
                    x       y filename          source unit  station  \
name
100000_320000  100000  320000           interpolation            NaN
100000_330000  100000  330000           interpolation            NaN
100000_340000  100000  340000           interpolation            NaN

              meteo_var                                                obs
name
100000_320000      EV24  EvaporationObs 100000_320000
-----metadata----...
100000_330000      EV24  EvaporationObs 100000_330000
-----metadata----...
100000_340000      EV24  EvaporationObs 100000_340000
-----metadata----...
```

The interpolation method is slow at first, but if collected for many different locations the time penalty is not that significant anymore.

Hydropandas and Pastas

This notebook demonstrates how hydropandas can be used to build Pastas Models. Pastas is a Python package that can be used to simulate and analyze groundwater time series. One of the biggest challenges when making a Pastas Model is to find a valid dataset. The Hydropandas packages can be used to obtain such a dataset. This notebook shows how to build a simple pastas model from Dutch data.

1. *Groundwater observations*
2. *Meteo observations*
3. *Pastas model*
4. *Surface water observations*
5. *Pastastore*

[1]:
```
import hydropandas as hpd
import pastas as ps
import pastastore as pst

import logging

ps.set_log_level("ERROR")
hpd.util.get_color_logger("INFO")

print(hpd.__version__)
print(ps.__version__)
```

```
0.8.1b
1.1.0
```

Groundwater observations

First we read the groundwater level observation using the `from_dino` function of a `GroundwaterObs` object. The code below reads the groundwater level timeseries from a csv file in the data directory.

[2]:
```
gw_levels = hpd.GroundwaterObs.from_dino(r"data/B49F0555001_1.csv")
```

```
INFO:hydropandas.io.dino:reading -> B49F0555001_1
```

Note that `gw_levels` is a `GroundwaterObs` object. This is basically a pandas DataFrame with additional methods and attributes related to groundwater level observations. We can print `gw_levels` to see all of its properties.

[3]:
```
print(gw_levels)
```

```
GroundwaterObs B49F0555-001
-----metadata------
name : B49F0555-001
x : 94532.0
y : 399958.0
filename : B49F0555001_1
source : dino
unit : m NAP
monitoring_well : B49F0555
tube_nr : 1.0
```

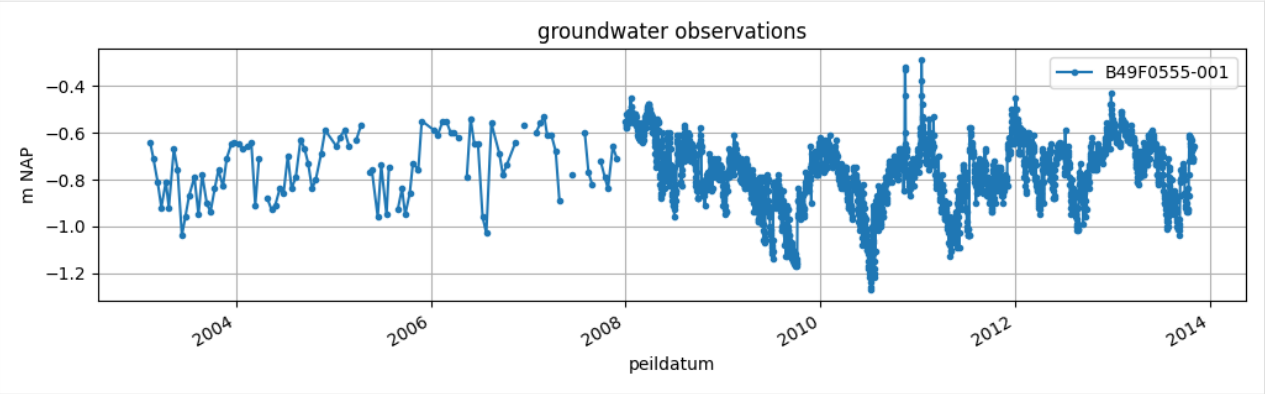
(continues on next page)

(continued from previous page)

```
screen_top : -0.75
screen_bottom : -1.75
ground_level : -0.61
tube_top : -0.15
metadata_available : True

-----time series------
            stand_m_tov_nap   locatie  filternummer  stand_cm_tov_mp  \
peildatum
2003-02-14            -0.64  B49F0555             1             49.0
2003-02-26            -0.71  B49F0555             1             56.0
2003-03-14            -0.81  B49F0555             1             66.0
2003-03-28            -0.92  B49F0555             1             77.0
2003-04-14            -0.81  B49F0555             1             66.0
...                     ...       ...           ...              ...
2013-10-26            -0.71  B49F0555             1             56.0
2013-10-27            -0.72  B49F0555             1             57.0
2013-10-28            -0.71  B49F0555             1             56.0
2013-10-29            -0.67  B49F0555             1             52.0
2013-10-30            -0.66  B49F0555             1             51.0

            stand_cm_tov_mv  stand_cm_tov_nap bijzonderheid  opmerking
peildatum
2003-02-14              3.0             -64.0           NaN        NaN
2003-02-26             10.0             -71.0           NaN        NaN
2003-03-14             20.0             -81.0           NaN        NaN
2003-03-28             31.0             -92.0           NaN        NaN
2003-04-14             20.0             -81.0           NaN        NaN
...                     ...               ...           ...        ...
2013-10-26             10.0             -71.0           NaN        NaN
2013-10-27             11.0             -72.0           NaN        NaN
2013-10-28             10.0             -71.0           NaN        NaN
2013-10-29              6.0             -67.0           NaN        NaN
2013-10-30              5.0             -66.0           NaN        NaN

[2241 rows x 8 columns]
```

The `GroundwaterObs` object comes with its own plot methods, to quickly visualize the data:

[4]:
```
ax = gw_levels["stand_m_tov_nap"].plot(
    figsize=(12, 3),
    marker=".",
    grid=True,
    label=gw_levels.name,
    legend=True,
    ylabel="m NAP",
    title="groundwater observations",
)
```



Meteo observations

We can obtain evaporation and precipitation data from the knmi using the `from_knmi_nearest_xy` method in hydropandas. For a given set of (x,y) coordinates it will find the closest KNMI meteostation and download the data. If there are missing measurements they are filled automatically using the value of a nearby station.

[5]:
```
evaporation = hpd.EvaporationObs.from_knmi(
    xy=(gw_levels.x, gw_levels.y),
    meteo_var="EV24",
    start=gw_levels.index[0],
    end=gw_levels.index[-1],
    fill_missing_obs=True,
)
precipitation = hpd.PrecipitationObs.from_knmi(
    xy=(gw_levels.x, gw_levels.y),
    start=gw_levels.index[0],
    end=gw_levels.index[-1],
    fill_missing_obs=True,
)
```

```
INFO:hydropandas.io.knmi:get KNMI data from station nearest to coordinates (94532.0,␣
↪399958.0) and meteovariable EV24
INFO:hydropandas.io.knmi:download knmi EV24 data from station 340-WOENSDRECHT between␣
↪2003-02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 340-WOENSDRECHT between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 340 has no measurements between 2003-02-14 00:00:00 and␣
↪2013-10-30 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi EV24 data from station 331-THOLEN between 2003-02-
↪14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 331-THOLEN between 2003-02-14␣
↪00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 331 has no measurements between 2003-02-14 00:00:00 and␣
↪2013-10-30 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi EV24 data from station 315-HANSWEERT between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 315-HANSWEERT between 2003-02-
```

(continues on next page)

(continued from previous page)

```
↪14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 315 has no measurements between 2003-02-14 00:00:00 and␣
↪2013-10-30 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi EV24 data from station 323-WILHELMINADORP between␣
↪2003-02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 323 has 3 missing measurements
INFO:hydropandas.io.knmi:trying to fill 3 measurements with station [324]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 324-STAVENISSE between␣
↪2003-02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 324-STAVENISSE between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
WARNING:hydropandas.io.knmi:station 324 cannot be downloaded
INFO:hydropandas.io.knmi:trying to fill 3 measurements with station [316]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 316-SCHAAR between 2003-02-
↪14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 316-SCHAAR between 2003-02-14␣
↪00:00:00 and 2013-10-30 00:00:00
WARNING:hydropandas.io.knmi:station 316 cannot be downloaded
INFO:hydropandas.io.knmi:trying to fill 3 measurements with station [310]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 310-VLISSINGEN between␣
↪2003-02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station nearest to coordinates (94532.0,␣
↪399958.0) and meteovariable RH
INFO:hydropandas.io.knmi:download knmi RH data from station 340-WOENSDRECHT between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 340-WOENSDRECHT between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 340 has no measurements between 2003-02-14 00:00:00 and␣
↪2013-10-30 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi RH data from station 331-THOLEN between 2003-02-
↪14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 331-THOLEN between 2003-02-14␣
↪00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 331 has no measurements between 2003-02-14 00:00:00 and␣
↪2013-10-30 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi RH data from station 315-HANSWEERT between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 315-HANSWEERT between 2003-02-
↪14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 315 has no measurements between 2003-02-14 00:00:00 and␣
↪2013-10-30 00:00:00
INFO:hydropandas.io.knmi:trying to get measurements from nearest station
INFO:hydropandas.io.knmi:download knmi RH data from station 323-WILHELMINADORP between␣
↪2003-02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:station 323 has 3 missing measurements
INFO:hydropandas.io.knmi:trying to fill 3 measurements with station [324]
INFO:hydropandas.io.knmi:download knmi RH data from station 324-STAVENISSE between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 324-STAVENISSE between 2003-
```

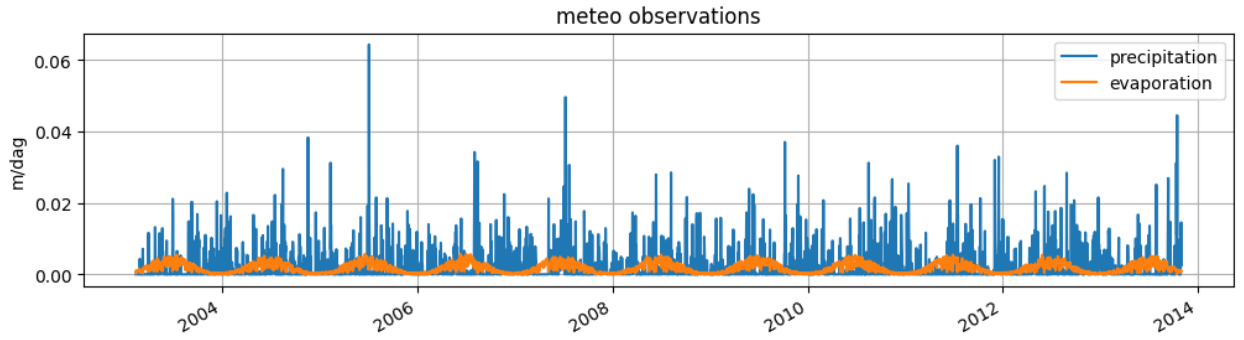
(continues on next page)

(continued from previous page)

```
↪02-14 00:00:00 and 2013-10-30 00:00:00
WARNING:hydropandas.io.knmi:station 324 cannot be downloaded
INFO:hydropandas.io.knmi:trying to fill 3 measurements with station [316]
INFO:hydropandas.io.knmi:download knmi RH data from station 316-SCHAAR between 2003-02-
↪14 00:00:00 and 2013-10-30 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 316-SCHAAR between 2003-02-14␣
↪00:00:00 and 2013-10-30 00:00:00
WARNING:hydropandas.io.knmi:station 316 cannot be downloaded
INFO:hydropandas.io.knmi:trying to fill 3 measurements with station [310]
INFO:hydropandas.io.knmi:download knmi RH data from station 310-VLISSINGEN between 2003-
↪02-14 00:00:00 and 2013-10-30 00:00:00
```

[6]:
```
ax = precipitation["RH"].plot(label="precipitation", legend=True, figsize=(12, 3))
evaporation["EV24"].plot(
    ax=ax,
    label="evaporation",
    legend=True,
    grid=True,
    ylabel="m/dag",
    title="meteo observations",
)
```

[6]:
```
<Axes: title={'center': 'meteo observations'}, ylabel='m/dag'>
```



Pastas model

Now that we have groundwater observations and meteo data we can create a Pastas model.

[7]:
```
ml = ps.Model(gw_levels["stand_m_tov_nap"], name=gw_levels.name)

# Add the recharge data as explanatory variable
ts1 = ps.RechargeModel(
    precipitation["RH"].resample("D").first(),
    evaporation["EV24"].resample("D").first(),
    ps.Gamma(),
    name="rainevap",
    settings=("prec", "evap"),
)
```

(continues on next page)

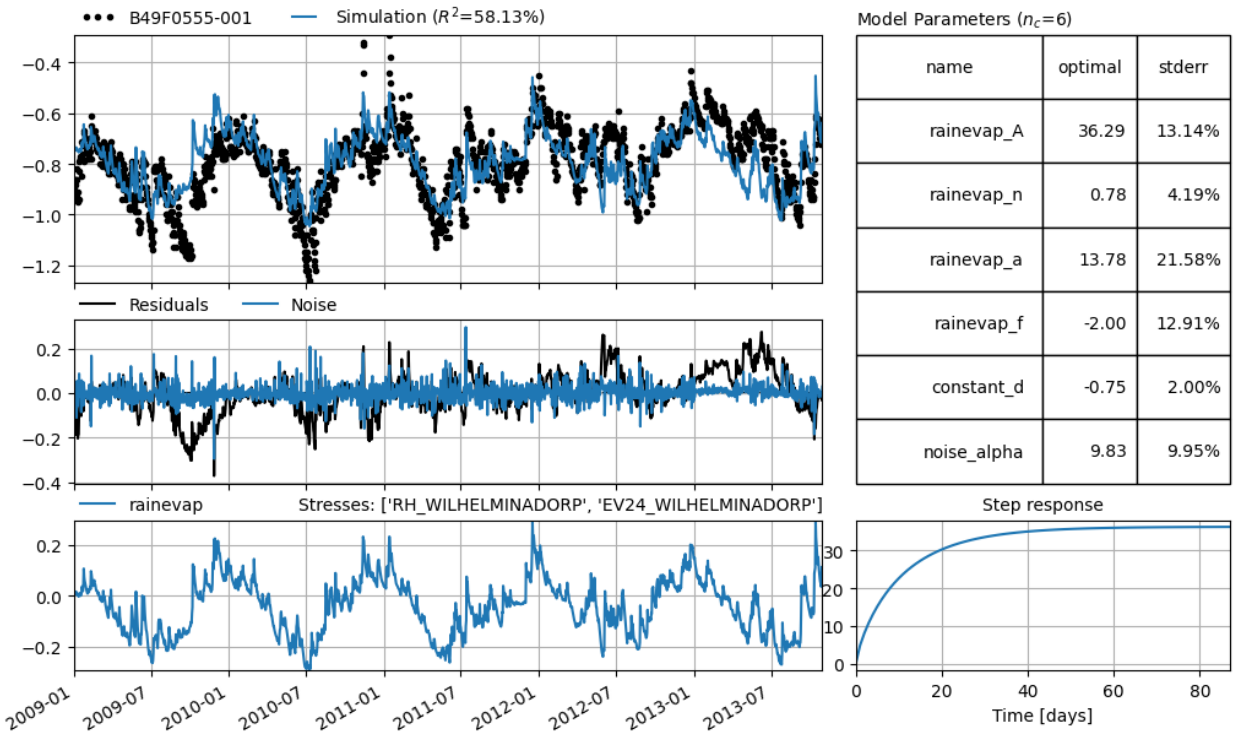
(continued from previous page)

```
ml.add_stressmodel(ts1)

ml.solve(tmin="2009")
```

```
Fit report B49F0555-001            Fit Statistics
================================================
nfev    35                     EVP          58.13
nobs    1764                   R2            0.58
noise   True                   RMSE          0.09
tmin    2009-01-01 00:00:00    AIC      -11449.32
tmax    2013-10-30 00:00:00    BIC      -11416.47
freq    D                      Obj           1.33
warmup  3650 days 00:00:00     ___
solver  LeastSquares           Interp.         No

Parameters (6 optimized)
================================================
               optimal   stderr     initial  vary
rainevap_A   36.288161  ±13.14%  199.486999  True
rainevap_n    0.783781   ±4.19%    1.000000  True
rainevap_a   13.784595  ±21.58%   10.000000  True
rainevap_f   -2.000000  ±12.91%   -1.000000  True
constant_d   -0.750944   ±2.00%   -0.774436  True
noise_alpha   9.830169   ±9.95%    1.000000  True

Warnings! (1)
================================================
Parameter 'rainevap_f' on lower bound: -2.00e+00
```

[8]:
```
ml.plots.results(figsize=(10, 6))
```

[8]:
```
[<Axes: xlabel='peildatum'>,
 <Axes: xlabel='peildatum'>,
 <Axes: title={'right': "Stresses: ['RH_WILHELMINADORP', 'EV24_WILHELMINADORP']"}>,
 <Axes: title={'center': 'Step response'}, xlabel='Time [days]'>,
 <Axes: title={'left': 'Model Parameters ($n_c$=6)'}>]
```



Surface water observations

Our model is not always able to simulate the groundwater level accurately. Maybe it will improve if we add surface water observations. The code below is used to read the surface water level timeseries from a csv file in the data directory.

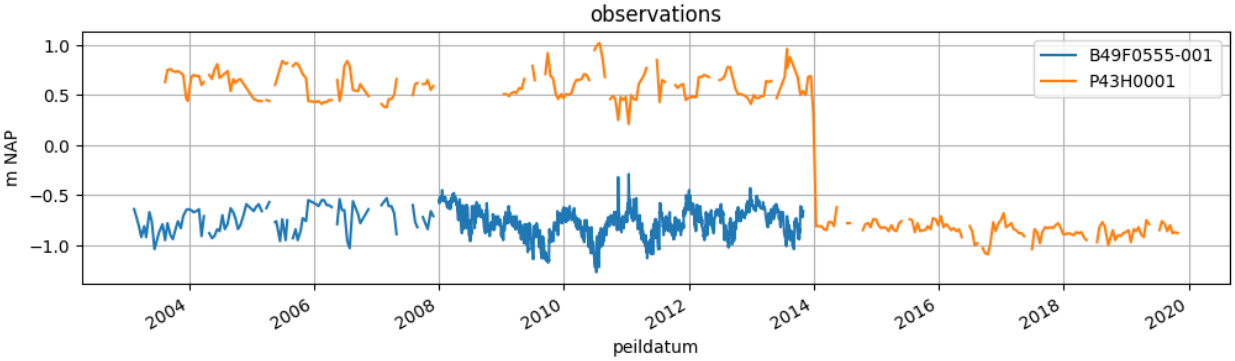
[9]:
```
river_levels = hpd.WaterlvlObs.from_dino(r"data/P43H0001.csv")
```

```
INFO:hydropandas.io.dino:reading -> P43H0001
```

We can plot the data together with the groundwater levels.

[10]:
```
ax = gw_levels["stand_m_tov_nap"].plot(
    figsize=(12, 3),
    label=gw_levels.name,
    ylabel="m NAP",
    title="observations",
    legend=True,
)
river_levels["stand_m_tov_nap"].plot(
    ax=ax, grid=True, label=river_levels.name, legend=True
)
```

[10]:
```
<Axes: title={'center': 'observations'}, xlabel='peildatum', ylabel='m NAP'>
```



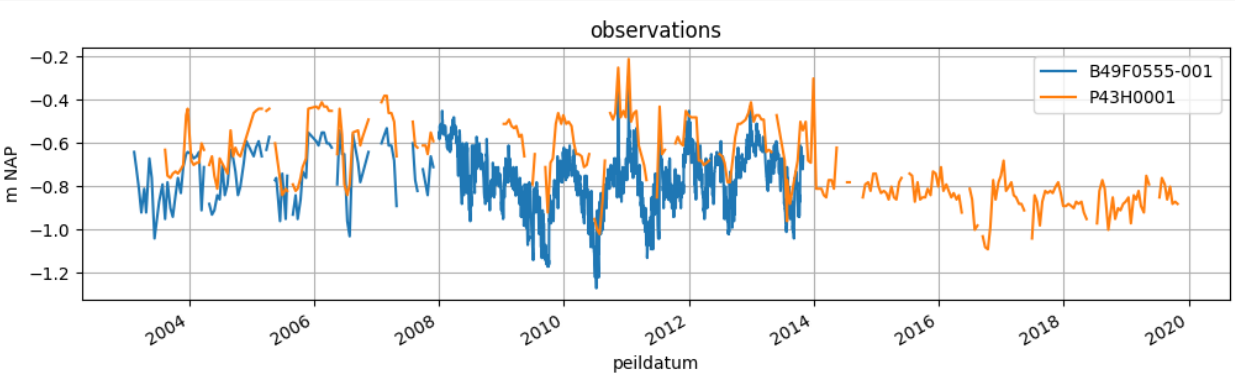
As can be observed in the plot above, there is a downward shift in the surface water levels at the end of 2014. Clearly something went wrong with the registration of the river levels. We assume that the negative values from the end of 2014 onwards are correct. The positive values were registered incorrectly (missing a minus sign). We fix the timeseries by updating the 'stand_m_tov_nap' column of the `WaterlvlObs` object named `river_levels`.

[11]:
```
river_levels["stand_m_tov_nap"] = river_levels["stand_m_tov_nap"].abs() * -1
```

Now we plot the timeseries again, to see if the applied fix looks reasonable.

[12]:
```
ax = gw_levels["stand_m_tov_nap"].plot(
    figsize=(12, 3),
    label=gw_levels.name,
    ylabel="m NAP",
    title="observations",
    legend=True,
)
river_levels["stand_m_tov_nap"].plot(
    ax=ax, grid=True, label=river_levels.name, legend=True
)
```

[12]:
```
<Axes: title={'center': 'observations'}, xlabel='peildatum', ylabel='m NAP'>
```



Now we add the river levels as an external stress in the pastas model.

[13]:
```
w = ps.StressModel(
    river_levels["stand_m_tov_nap"].resample("D").first(),
    rfunc=ps.One(),
    name="waterlevel",
    settings="waterlevel",
```

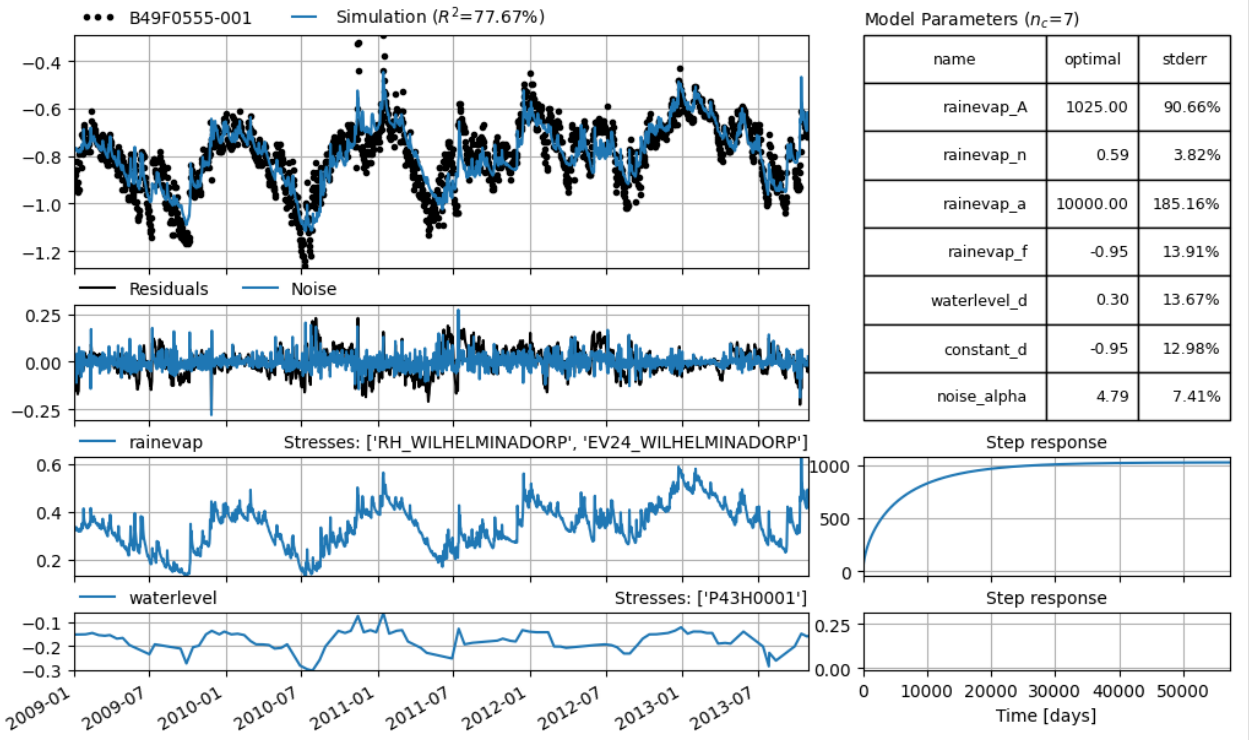
(continues on next page)

(continued from previous page)

```
)
ml.add_stressmodel(w)
ml.solve(tmin="2009")

ml.plots.results(figsize=(10, 6))
```

```
Fit report B49F0555-001                Fit Statistics
=======================================================
nfev    35                     EVP               77.67
nobs    1764                   R2                 0.78
noise   True                   RMSE               0.07
tmin    2009-01-01 00:00:00    AIC           -11466.58
tmax    2013-10-30 00:00:00    BIC           -11428.25
freq    D                      Obj                1.32
warmup  3650 days 00:00:00     ___
solver  LeastSquares           Interp.              No

Parameters (7 optimized)
=======================================================
                   optimal    stderr     initial  vary
rainevap_A     1024.998645   ±90.66%  199.486999  True
rainevap_n        0.594123    ±3.82%    1.000000  True
rainevap_a    10000.000000  ±185.16%   10.000000  True
rainevap_f       -0.951316   ±13.91%   -1.000000  True
waterlevel_d      0.298044   ±13.67%    0.161028  True
constant_d       -0.948496   ±12.98%   -0.774436  True
noise_alpha       4.789231    ±7.41%    1.000000  True

Warnings! (2)
=======================================================
Parameter 'rainevap_a' on upper bound: 1.00e+04
Response tmax for 'rainevap' > than calibration period.
```

[13]:
```
[<Axes: xlabel='peildatum'>,
 <Axes: xlabel='peildatum'>,
 <Axes: title={'right': "Stresses: ['RH_WILHELMINADORP', 'EV24_WILHELMINADORP']"}>,
 <Axes: title={'center': 'Step response'}, xlabel='Time [days]'>,
 <Axes: title={'right': "Stresses: ['P43H0001']"}>,
 <Axes: title={'center': 'Step response'}, xlabel='Time [days]'>,
 <Axes: title={'left': 'Model Parameters ($n_c$=7)'}>]
```



Model Parameters (n_c=7)

name	optimal	stderr
rainevap_A	1025.00	90.66%
rainevap_n	0.59	3.82%
rainevap_a	10000.00	185.16%
rainevap_f	-0.95	13.91%
waterlevel_d	0.30	13.67%
constant_d	-0.95	12.98%
noise_alpha	4.79	7.41%

We can see that the evp has increased when we've added the water level stress, does this mean that the model improved? Please have a look at the pastas documentation for more information on this subject.

Pastastore

We can also use the hydropandas package to create a PastaStore with multiple observations. A Pastastore is a combination of observations, stresses and pastas time series models. More information on the Pastastore can be found here.

Groundwater observations

First we read multiple observations from a directory with dino groundwater measurements. We store them in an ObsCollection object named `oc_dino`.

[14]:
```
extent = [117850, 117980, 439550, 439700]  # Schoonhoven zuid-west
dinozip = "data/dino.zip"
oc_dino = hpd.read_dino(dirname=dinozip, keep_all_obs=False)
oc_dino = oc_dino.loc[
    ["B58A0092-004", "B58A0092-005", "B58A0102-001", "B58A0167-001", "B58A0212-001"]
]
oc_dino
```

```
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B02H0092001_1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B02H0092001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B02H1007001_1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
```

(continues on next page)

(continued from previous page)

```
↪B02H1007001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B04D0032002_1.csv
WARNING:hydropandas.io.dino:could not read metadata -> Grondwaterstanden_Put/B04D0032002_
↪1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B04D0032002_1.csv
INFO:root:not added to collection -> Grondwaterstanden_Put/B04D0032002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B27D0260001_1.csv
WARNING:hydropandas.io.dino:could not read metadata -> Grondwaterstanden_Put/B27D0260001_
↪1.csv
WARNING:hydropandas.io.dino:no NAP measurements available -> Grondwaterstanden_Put/
↪B27D0260001_1.csv
INFO:root:not added to collection -> Grondwaterstanden_Put/B27D0260001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0080001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0080002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0133001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B33F0133002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B37A0112001_1.csv
WARNING:hydropandas.io.dino:could not read metadata -> Grondwaterstanden_Put/B37A0112001_
↪1.csv
WARNING:hydropandas.io.dino:could not read measurements -> Grondwaterstanden_Put/
↪B37A0112001_1.csv
INFO:root:not added to collection -> Grondwaterstanden_Put/B37A0112001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003002_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003003_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B42B0003004_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0092004_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0092005_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0102001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0167001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B58A0212001_1.csv
INFO:hydropandas.io.dino:reading -> Grondwaterstanden_Put/B22D0155001_1.csv
```

[14]:

```
                    x         y                                filename  \
name
B58A0092-004  186924.0  372026.0  Grondwaterstanden_Put/B58A0092004_1.csv
B58A0092-005  186924.0  372026.0  Grondwaterstanden_Put/B58A0092005_1.csv
B58A0102-001  187900.0  373025.0  Grondwaterstanden_Put/B58A0102001_1.csv
B58A0167-001  185745.0  371095.0  Grondwaterstanden_Put/B58A0167001_1.csv
B58A0212-001  183600.0  373020.0  Grondwaterstanden_Put/B58A0212001_1.csv

             source   unit monitoring_well  tube_nr  screen_top  \
name
B58A0092-004   dino  m NAP        B58A0092      4.0     -115.23
B58A0092-005   dino  m NAP        B58A0092      5.0     -134.23
B58A0102-001   dino  m NAP        B58A0102      1.0       -3.35
B58A0167-001   dino  m NAP        B58A0167      1.0       23.33
B58A0212-001   dino  m NAP        B58A0212      1.0       26.03

              screen_bottom  ground_level  tube_top  metadata_available  \
name
```

(continues on next page)

(continued from previous page)

```
B58A0092-004        -117.23         29.85     29.61                True
B58A0092-005        -137.23         29.84     29.62                True
B58A0102-001          -8.35         29.65     29.73                True
B58A0167-001          22.33         30.50     30.21                True
B58A0212-001          25.53         28.49     28.53                True

                                                                    obs
name
B58A0092-004  GroundwaterObs B58A0092-004
-----metadata-----...
B58A0092-005  GroundwaterObs B58A0092-005
-----metadata-----...
B58A0102-001  GroundwaterObs B58A0102-001
-----metadata-----...
B58A0167-001  GroundwaterObs B58A0167-001
-----metadata-----...
B58A0212-001  GroundwaterObs B58A0212-001
-----metadata-----...
```

Using the `to_pastastore()` method we can export the Dino measurements to a pastastore.

[15]:
```
# add observations to pastastore
pstore = oc_dino.to_pastastore()
```

```
INFO:hydropandas.io.pastas:add to pastastore -> B58A0092-004
INFO:hydropandas.io.pastas:add to pastastore -> B58A0092-005
INFO:hydropandas.io.pastas:add to pastastore -> B58A0102-001
INFO:hydropandas.io.pastas:add to pastastore -> B58A0167-001
INFO:hydropandas.io.pastas:add to pastastore -> B58A0212-001
```

Meteo

Besides the groundwater level observations we also want to add meteo data as external stresses in the pastastore. We use the `read_knmi` function to obtain the meteo data. For `locations` we use the ObsCollection with Dino data to get the KNMI station data nearest to our Dino observations. We specify `tmin` and `tmax` to get meteo data for the same period as our observations. Finally, we use the `meteo_vars=('RH', 'EV24')` to specify that we want precipitation (RH) and evaporation (EV24) observations respectively.

[16]:
```
# get tmin and tmax
tmintmax = pstore.get_tmin_tmax("oseries")
tmin = tmintmax.tmin.min()
tmax = tmintmax.tmax.max()

# get precipitation and evaporation
meteo_vars = ("RH", "EV24")

# get knmi ObsCollection
knmi_oc = hpd.read_knmi(
    locations=oc_dino,
    meteo_vars=meteo_vars,
    starts=tmin,
```

(continues on next page)

(continued from previous page)

```
    ends=tmax,
    fill_missing_obs=True,
)
knmi_oc
```

```
INFO:hydropandas.io.knmi:get KNMI data from station 377 and meteo variable RHfrom 1963-
↪05-28 00:00:00 to 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:download knmi RH data from station 377-ELL between 1963-05-28␣
↪00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:station 377 has no measurements before 1999-07-16 01:00:00
INFO:hydropandas.io.knmi:station 377 has 13198 missing measurements
INFO:hydropandas.io.knmi:trying to fill 13198 measurements with station [380]
INFO:hydropandas.io.knmi:download knmi RH data from station 380-MAASTRICHT between 1963-
↪05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:get KNMI data from station 377 and meteo variable EV24from 1963-
↪05-28 00:00:00 to 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:download knmi EV24 data from station 377-ELL between 1963-05-28␣
↪00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:station 377 has no measurements before 1999-07-16 01:00:00
INFO:hydropandas.io.knmi:station 377 has 13198 missing measurements
INFO:hydropandas.io.knmi:trying to fill 13198 measurements with station [380]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 380-MAASTRICHT between␣
↪1963-05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [370]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 370-EINDHOVEN between 1963-
↪05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [391]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 391-ARCEN between 1963-05-
↪28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [375]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 375-VOLKEL between 1963-05-
↪28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [350]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 350-GILZE-RIJEN between␣
↪1963-05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [356]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 356-HERWIJNEN between 1963-
↪05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [275]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 275-DEELEN between 1963-05-
↪28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [340]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 340-WOENSDRECHT between␣
↪1963-05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:no measurements found for station 340-WOENSDRECHT between 1963-
↪05-28 00:00:00 and 2018-07-24 00:00:00
WARNING:hydropandas.io.knmi:station 340 cannot be downloaded
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [348]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 348-CABAUW-MAST between␣
↪1963-05-28 00:00:00 and 2018-07-24 00:00:00
INFO:hydropandas.io.knmi:trying to fill 371 measurements with station [260]
INFO:hydropandas.io.knmi:download knmi EV24 data from station 260-DE-BILT between 1963-
```

(continues on next page)

(continued from previous page)

```
↪05-28 00:00:00 and 2018-07-24 00:00:00
```

[16]:
```
                      x              y filename source unit  station  \
name
RH_ELL    181242.447083  356427.781413            KNMI           377
EV24_ELL  181242.447083  356427.781413            KNMI           377

         meteo_var                                                 obs
name
RH_ELL          RH  PrecipitationObs RH_ELL
-----metadata------
na...
EV24_ELL      EV24  EvaporationObs EV24_ELL
-----metadata------
na...
```

Using the `to_pastastore` method we export the meteo observations as external stresses to the pastastore. Note that we add the stresses to the existing pastastore (`pstore`) with our groundwater observations.

[17]:
```
# add stresses to pastastore
kinds = ("prec", "evap")

for i, meteo_var in enumerate(meteo_vars):
    knmi_oc[knmi_oc.meteo_var == meteo_var].to_pastastore(
        pstore, col=None, kind=kinds[i]
    )
pstore
```

```
INFO:hydropandas.io.pastas:add to pastastore -> RH_ELL
INFO:hydropandas.io.pastas:add to pastastore -> EV24_ELL
```

[17]:
```
<PastaStore> :
 - <DictConnector> 'my_db': 5 oseries, 2 stresses, 0 models
```

Creating and solving models

The Pastastore contains time series with observation and stresses. This means we can build and solve the pastas models using the `create_models_bulk` and `solve_models` methods.

[18]:
```
pstore.create_models_bulk(store=True, add_recharge=True, ignore_errors=False)
pstore.solve_models()
pstore
```

```
Bulk creation models:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
Bulk creation models: 100%|| 5/5 [00:00<00:00, 16.86it/s]
Solving models: 100%|| 5/5 [00:03<00:00,  1.43it/s]
```

[18]:
```
<PastaStore> :
 - <DictConnector> 'my_db': 5 oseries, 2 stresses, 5 models
```

Model results, such as explained variance percentage (evp), can be obtained using the `get_statistics` method.

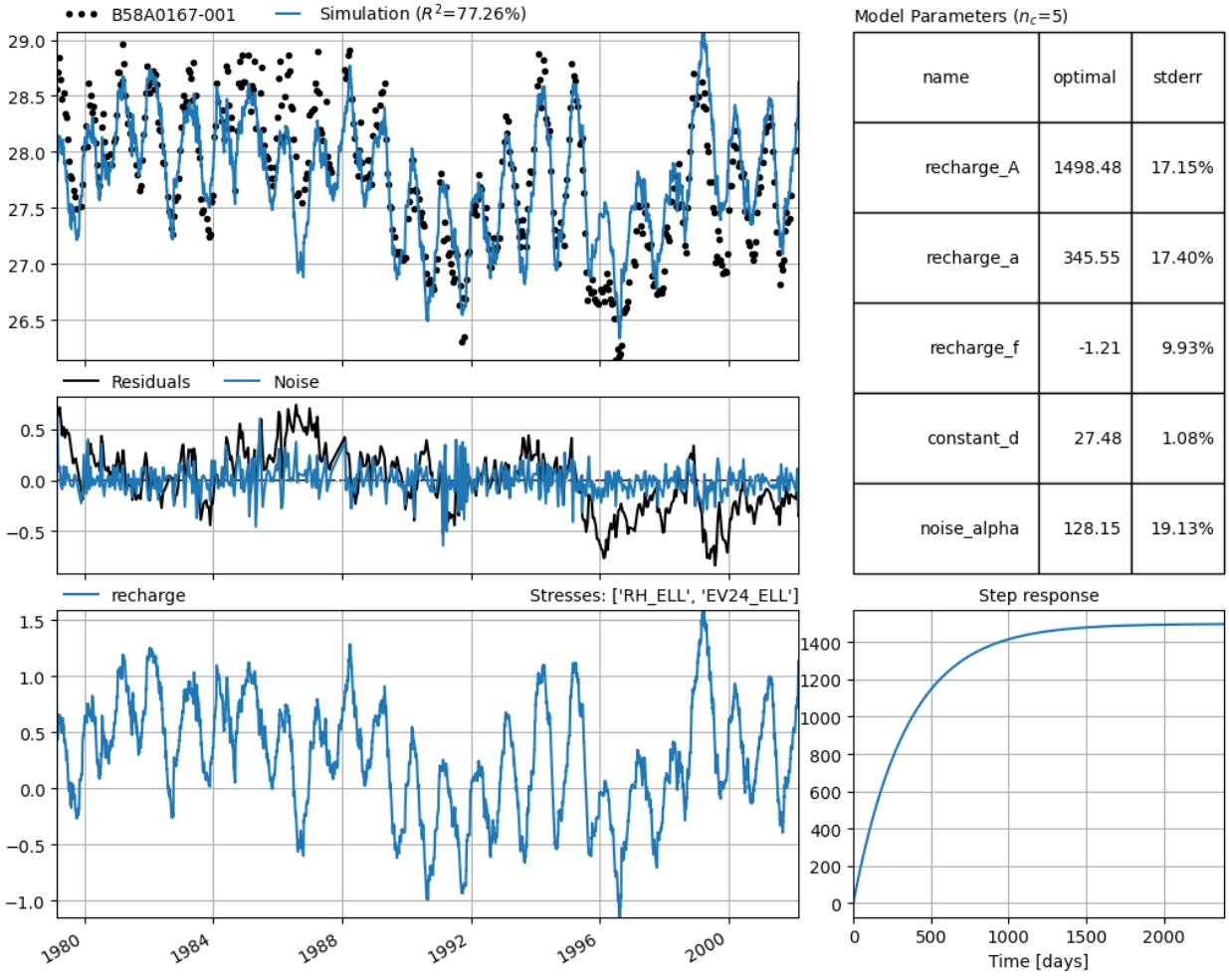
[19]:
```
pstore.get_statistics(["evp"])
```

[19]:
```
B58A0092-004     0.000000
B58A0092-005    48.821134
B58A0102-001    84.360621
B58A0167-001    77.290365
B58A0212-001    65.252635
Name: evp, dtype: float64
```

Finally we can extract a single model from the pastastore to visualise its results.

[20]:
```
# results from a single model
ml1 = pstore.get_models("B58A0167-001")
ml1.plots.results()
```

[20]:
```
[<Axes: >,
 <Axes: >,
 <Axes: title={'right': "Stresses: ['RH_ELL', 'EV24_ELL']"}>,
 <Axes: title={'center': 'Step response'}, xlabel='Time [days]'>,
 <Axes: title={'left': 'Model Parameters ($n_c$=5)'}>]
```



Model Parameters (n_c=5)

name	optimal	stderr
recharge_A	1498.48	17.15%
recharge_a	345.55	17.40%
recharge_f	-1.21	9.93%
constant_d	27.48	1.08%
noise_alpha	128.15	19.13%

Merging observations

This notebook shows how observations and observation collections can be merged. Merging observations can be useful if: - you have data from multiple sources measuring at the same location - you get new measurements that you want to add to the old measurements.

Notebook contents

1. Simple merge
2. Merge options
3. Merging observation collections

[1]:
```
import numpy as np
import pandas as pd
import hydropandas as hpd
from IPython.display import display

import logging

hpd.util.get_color_logger("INFO");
```

Simple merge

[2]:
```
# observation 1
df = pd.DataFrame(
    {"measurements": np.random.randint(0, 10, 5)},
    index=pd.date_range("2020-1-1", "2020-1-5"),
)
o1 = hpd.Obs(df, name="obs", x=0, y=0)
print(o1)
```

```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-01             4
2020-01-02             4
2020-01-03             9
2020-01-04             3
2020-01-05             8
```

[3]:
```
# observation 2
df = pd.DataFrame(
```

(continues on next page)

(continued from previous page)

```
    {"measurements": np.random.randint(0, 10, 5)},
    index=pd.date_range("2020-1-6", "2020-1-10"),
)
o2 = hpd.Obs(df, name="obs", x=0, y=0)
print(o2)
```

```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-06             7
2020-01-07             3
2020-01-08             8
2020-01-09             6
2020-01-10             3
```

[4]:
```
o1.merge_observation(o2)
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
INFO:hydropandas.observation:new and existing observation have the same metadata
```

[4]:
```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-01             4
2020-01-02             4
2020-01-03             9
2020-01-04             3
2020-01-05             8
2020-01-06             7
2020-01-07             3
2020-01-08             8
2020-01-09             6
2020-01-10             3
```

Merge options

overlapping timeseries

[5]:
```
# create a parly overlapping dataframe
df = pd.DataFrame(
    {
        "measurements": np.concatenate(
            [o1["measurements"].values[-2:], np.random.randint(0, 10, 3)]
        )
    },
    index=pd.date_range("2020-1-4", "2020-1-8"),
)
o3 = hpd.Obs(df, name="obs", x=0, y=0)
print(o3)
```

```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-04             3
2020-01-05             8
2020-01-06             4
2020-01-07             5
2020-01-08             1
```

[6]:
```
o1.merge_observation(o3)
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
INFO:hydropandas.observation:new and existing observation have the same metadata
```

[6]:
```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-01             4
2020-01-02             4
```

(continues on next page)

(continued from previous page)

```
2020-01-03             9
2020-01-04             3
2020-01-05             8
2020-01-06             4
2020-01-07             5
2020-01-08             1
```

[7]:
```
# create a parly overlapping dataframe with different values
df = pd.DataFrame(
    {"measurements": np.random.randint(0, 10, 5)},
    index=pd.date_range("2020-1-4", "2020-1-8"),
)
o4 = hpd.Obs(df, name="obs", x=0, y=0)
print(o4)
```

```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-04             3
2020-01-05             4
2020-01-06             0
2020-01-07             4
2020-01-08             0
```

by default an error is raised if the overlapping time series have different values

[8]:
```
o1.merge_observation(o4)
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
WARNING:hydropandas.observation:timeseries of observation obs overlap withdifferent␣
↪values
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-fb7c0e48ad44> in <module>
----> 1 o1.merge_observation(o4)

c:\users\oebbe\02_python\hydropandas\hydropandas\observation.py in merge_
↪observation(self, right, overlap, merge_metadata)
    427
    428         # merge timeseries
--> 429         o = self._merge_timeseries(right, overlap=overlap)
```

(continues on next page)

(continued from previous page)

```
    430
    431         # merge metadata

c:\users\oebbe\02_python\hydropandas\hydropandas\observation.py in _merge_
↪timeseries(self, right, overlap)
    336                 )
--> 338                     raise ValueError(
    337                 if overlap == "error":
    339                         "observations have different values for same time steps"
    340                     )

ValueError: observations have different values for same time steps
```

With the 'overlap' argument you can specify to use the left or the right observation when merging. See example below.

[9]:
```
print("use left")
display(o1.merge_observation(o4, overlap="use_left"))  # use the existing observation
print("use right")
display(o1.merge_observation(o4, overlap="use_right"))  # use the existing observation
```

```
use left
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
WARNING:hydropandas.observation:timeseries of observation obs overlap withdifferent␣
↪values
INFO:hydropandas.observation:new and existing observation have the same metadata
```

```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-01             4
2020-01-02             4
2020-01-03             9
2020-01-04             3
2020-01-05             8
2020-01-06             0
2020-01-07             4
2020-01-08             0
```

```
use right
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
```

(continues on next page)

(continued from previous page)

```
WARNING:hydropandas.observation:timeseries of observation obs overlap withdifferent␣
↪values
INFO:hydropandas.observation:new and existing observation have the same metadata
```

```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-01             4
2020-01-02             4
2020-01-03             9
2020-01-04             3
2020-01-05             4
2020-01-06             0
2020-01-07             4
2020-01-08             0
```

metadata

The `merge_observation` method checks by default if the metadata of the two observations is the same.

[10]:
```
# observation 2
df = pd.DataFrame(
    {"measurements": np.random.randint(0, 10, 5)},
    index=pd.date_range("2020-1-6", "2020-1-10"),
)
o5 = hpd.Obs(df, name="obs5", x=0, y=0)
o5
```

[10]:
```
Obs obs5
-----metadata------
name : obs5
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-06             0
2020-01-07             6
2020-01-08             8
2020-01-09             6
2020-01-10             7
```

When the metadata differs a ValueError is raised.

[11]:
```
o1.merge_observation(o5)
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-204327616616> in <module>
----> 1 o1.merge_observation(o5)

c:\users\oebbe\02_python\hydropandas\hydropandas\observation.py in merge_
↪observation(self, right, overlap, merge_metadata)
    432         if merge_metadata:
    433             metadata = {key: getattr(right, key) for key in right._metadata}
--> 434             new_metadata = self.merge_metadata(metadata, overlap=overlap)
    435         else:
    436             new_metadata = {key: getattr(self, key) for key in self._metadata}

c:\users\oebbe\02_python\hydropandas\hydropandas\observation.py in merge_metadata(self,␣
↪right, overlap)
    245                     same_metadata = False
--> 247                     raise ValueError(
    246                 if overlap == "error":
    248                         f"existing observation {key} differs from new␣
↪observation"
    249                     )

ValueError: existing observation name differs from new observation
```

If you set the `merge_metadata` argument to `False` the metadata is not merged and only the timeseries of the observations is merged.

[12]:
```
o1.merge_observation(o5, merge_metadata=False)
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
```

[12]:
```
Obs obs
-----metadata------
name : obs
x : 0
y : 0
filename :
source :
unit :

-----time series------
            measurements
2020-01-01             4
2020-01-02             4
```

(continues on next page)

(continued from previous page)

```
2020-01-03              9
2020-01-04              3
2020-01-05              8
2020-01-06              0
2020-01-07              6
2020-01-08              8
2020-01-09              6
2020-01-10              7
```

Just as with overlapping timeseries, the 'overlap' argument can also be used for overlapping metadata values

[13]:
```
o_merged = o1.merge_observation(o5, overlap="use_left", merge_metadata=True)
print('observation name when overlap="use_left":', o_merged.name)
o_merged = o1.merge_observation(o5, overlap="use_right", merge_metadata=True)
print('observation name when overlap="use_right":', o_merged.name)
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
INFO:hydropandas.observation:existing observation name differs from newobservation, use␣
↪existing
observation name when overlap="use_left": obs
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
INFO:hydropandas.observation:existing observation name differs from newobservation, use␣
↪new
observation name when overlap="use_right": obs5
```

all combinations

[14]:
```
# observation 6
df = pd.DataFrame(
    {"measurements": np.random.randint(0, 10, 5), "filter": np.ones(5)},
    index=pd.date_range("2020-1-1", "2020-1-5"),
)
o6 = hpd.Obs(df, name="obs6", x=100, y=0)
o6
```

[14]:
```
Obs obs6
-----metadata------
name : obs6
x : 100
y : 0
filename :
source :
unit :

-----time series------
```

(continues on next page)

(continued from previous page)

```
            measurements  filter
2020-01-01             8     1.0
2020-01-02             7     1.0
2020-01-03             5     1.0
2020-01-04             3     1.0
2020-01-05             6     1.0
```

[15]:
```
# observation 7
df = pd.DataFrame(
    {
        "measurements": np.concatenate(
            [o5["measurements"].values[-1:], np.random.randint(0, 10, 4)]
        ),
        "remarks": ["", "", "", "unreliable", ""],
    },
    index=pd.date_range("2020-1-4", "2020-1-8"),
)
o7 = hpd.Obs(df, name="obs7", x=0, y=100)
o7
```

[15]:
```
Obs obs7
-----metadata------
name : obs7
x : 0
y : 100
filename :
source :
unit :

-----time series------
            measurements     remarks
2020-01-04             7
2020-01-05             5
2020-01-06             0
2020-01-07             9  unreliable
2020-01-08             6
```

[16]:
```
o6.merge_observation(o7, overlap="use_right")
```

```
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
WARNING:hydropandas.observation:timeseries of observation obs7 overlap withdifferent␣
↪values
INFO:hydropandas.observation:existing observation name differs from newobservation, use␣
↪new
INFO:hydropandas.observation:existing observation x differs from newobservation, use new
INFO:hydropandas.observation:existing observation y differs from newobservation, use new
```

[16]:
```
Obs obs7
-----metadata------
name : obs7
```

(continues on next page)

(continued from previous page)

```
x : 0
y : 100
filename :
source :
unit :

-----time series------
            measurements     remarks  filter
2020-01-01             8         NaN     1.0
2020-01-02             7         NaN     1.0
2020-01-03             5         NaN     1.0
2020-01-04             7                 1.0
2020-01-05             5                 1.0
2020-01-06             0                 NaN
2020-01-07             9  unreliable     NaN
2020-01-08             6                 NaN
```

Merge observation collections

[17]:
```
# create an observation collection from a single observation
oc1 = hpd.ObsCollection(o1)
```

We can add a single observation to this collection using the `add_observation` method.

[18]:
```
oc1.add_observation(o2)
oc1
```

```
INFO:hydropandas.obs_collection:observation name obs already in collection, merging␣
↪observations
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
INFO:hydropandas.observation:new and existing observation have the same metadata
```

[18]:
```
      x  y filename source unit  \
name
obs   0  0

                                                    obs
name
obs   Obs obs
-----metadata------
name : obs
x : 0 ...
```

We can also combine two observation collections.

[19]:
```
# create another observation collection from a list of observations
oc2 = hpd.ObsCollection([o5, o6])
oc2
```

(continues on next page)

(continued from previous page)

```
# add the collection to the previous one
oc1.add_obs_collection(oc2, inplace=True)
oc1
```

```
INFO:hydropandas.obs_collection:adding obs5 to collection
INFO:hydropandas.obs_collection:adding obs6 to collection
```

[19]:
```
        x  y filename source unit  \
name
obs     0  0
obs5    0  0
obs6  100  0

                                                    obs
name
obs   Obs obs
-----metadata------
name : obs
x : 0 ...
obs5  Obs obs5
-----metadata------
name : obs5
x : ...
obs6  Obs obs6
-----metadata------
name : obs6
x : ...
```

There is an automatic check for overlap based on the name of the observations. If the observations in both collections are exactly the same they are merged.

[20]:
```
# add o2 to the observation collection 1
oc1.add_observation(o2)
```

```
INFO:hydropandas.obs_collection:observation name obs already in collection, merging␣
↪observations
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
INFO:hydropandas.observation:new and existing observation have the same metadata
```

If the observation you want to add has the same name but not the same timeseries an error is raised.

[21]:
```
o1_mod = o1.copy()
o1_mod.loc["2020-01-02", "measurements"] = 100
oc1.add_observation(o1_mod)
```

```
INFO:hydropandas.obs_collection:observation name obs already in collection, merging␣
↪observations
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
```

(continues on next page)

(continued from previous page)

```
WARNING:hydropandas.observation:timeseries of observation obs overlap withdifferent␣
↪values
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-ffb5717f4bf3> in <module>
      1 o1_mod = o1.copy()
      2 o1_mod.loc["2020-01-02", "measurements"] = 100
----> 3 oc1.add_observation(o1_mod)

c:\users\oebbe\02_python\hydropandas\hydropandas\obs_collection.py in add_
↪observation(self, o, check_consistency, **kwargs)
    876
    877             o1 = self.loc[o.name, "obs"]
--> 878             omerged = o1.merge_observation(o, **kwargs)
    879
    880             # overwrite observation in collection

c:\users\oebbe\02_python\hydropandas\hydropandas\observation.py in merge_
↪observation(self, right, overlap, merge_metadata)
    427
    428         # merge timeseries
--> 429         o = self._merge_timeseries(right, overlap=overlap)
    430
    431         # merge metadata

c:\users\oebbe\02_python\hydropandas\hydropandas\observation.py in _merge_
↪timeseries(self, right, overlap)
    336                 )
--> 338                     raise ValueError(
    337                 if overlap == "error":
    339                         "observations have different values for same time steps"
    340                     )

ValueError: observations have different values for same time steps
```

To avoid errors we can use the `overlap` arguments to specify which observation we want to use.

[22]:
```
oc1.add_observation(o1_mod, overlap="use_left")
oc1
```

```
INFO:hydropandas.obs_collection:observation name obs already in collection, merging␣
↪observations
WARNING:hydropandas.observation:function 'merge_observation' not thoroughly tested,␣
↪please be carefull!
INFO:hydropandas.observation:new observation has a different time series
INFO:hydropandas.observation:merge time series
WARNING:hydropandas.observation:timeseries of observation obs overlap withdifferent␣
↪values
INFO:hydropandas.observation:new and existing observation have the same metadata
```

[22]:
```
        x  y filename source unit  \
name
obs     0  0
```

(continues on next page)

(continued from previous page)

```
obs5    0  0
obs6  100  0


                                                    obs
name
obs   Obs obs
-----metadata------
name : obs
x : 0 ...
obs5  Obs obs5
-----metadata------
name : obs5
x : ...
obs6  Obs obs6
-----metadata------
name : obs6
x : ...
```

Reading Bronhouderportaal BRO data

This notebook introduces how to use the `hydropandas` package to read, visualise and analyse meta data of newly installed groundwater wells. These meta data is to be submitted to Bronhouderportaal BRO afterwards.

Notebook contents

1. Read ObsCollection
2. Visualise ObsCollection
3. Analyse ObsCollection

[1]:
```
import hydropandas as hpd

import logging
from IPython.display import HTML

import pandas as pd
```

[2]:
```
hpd.util.get_color_logger("INFO")
```

[2]:
```
<RootLogger root (INFO)>
```

Read ObsCollection

An ObsCollection is created for the multiple monitoring wells.

[3]:
```
dirname = "data/bronhouderportaal-bro"
oc = hpd.read_bronhouderportaal_bro(dirname, full_meta=True, add_to_df=True)
oc
```

[3]:
```
                          x           y         filename  \
name
GROND5_B1-1#000-1  56525.207  386749.698  GROND5_B1-1.xml
GROND5_B1-1#000-2  56525.207  386749.698  GROND5_B1-1.xml
GROND5_B1-2#000-1  56336.522  386967.299  GROND5_B1-2.xml
GROND5_B1-2#000-2  56336.522  386967.299  GROND5_B1-2.xml
GROND5_B1-3#000-1  56349.209  387156.262  GROND5_B1-3.xml
GROND5_B1-3#000-2  56349.209  387156.262  GROND5_B1-3.xml

                                  source   unit  monitoring_well  tube_nr  \
name
GROND5_B1-1#000-1  bronhouderportaal-bro  m NAP  GROND5_B1-1#000        1
GROND5_B1-1#000-2  bronhouderportaal-bro  m NAP  GROND5_B1-1#000        2
GROND5_B1-2#000-1  bronhouderportaal-bro  m NAP  GROND5_B1-2#000        1
GROND5_B1-2#000-2  bronhouderportaal-bro  m NAP  GROND5_B1-2#000        2
GROND5_B1-3#000-1  bronhouderportaal-bro  m NAP  GROND5_B1-3#000        1
GROND5_B1-3#000-2  bronhouderportaal-bro  m NAP  GROND5_B1-3#000        2

                   screen_top  screen_bottom  ground_level  ...  \
name                                                        ...
GROND5_B1-1#000-1      -10.64         -11.64          0.36  ...
GROND5_B1-1#000-2       -1.14          -2.14          0.36  ...
GROND5_B1-2#000-1       -9.33         -10.33          1.67  ...
GROND5_B1-2#000-2        0.17          -0.83          1.67  ...
GROND5_B1-3#000-1       -9.70         -10.70          1.30  ...
GROND5_B1-3#000-2       -0.20          -1.20          1.30  ...

                   constructionStandard  tubeGlue artesianWellCapPresent  \
name
GROND5_B1-1#000-1               NEN5766      geen                     ja
GROND5_B1-1#000-2               NEN5766      geen                     ja
GROND5_B1-2#000-1               NEN5766      geen                     ja
GROND5_B1-2#000-2               NEN5766      geen                     ja
GROND5_B1-3#000-1               NEN5766      geen                     ja
GROND5_B1-3#000-2               NEN5766      geen                     ja

                  plainTubePartLength  wellHeadProtector  \
name
GROND5_B1-1#000-1               11.50    kokerNietMetaal
GROND5_B1-1#000-2                2.21    kokerNietMetaal
GROND5_B1-2#000-1               11.49    kokerNietMetaal
GROND5_B1-2#000-2                2.17    kokerNietMetaal
GROND5_B1-3#000-1               11.44    kokerNietMetaal
GROND5_B1-3#000-2                2.20    kokerNietMetaal

                  objectIdAccountableParty  owner  tube_bottom tubeMaterial  \
```

(continues on next page)

(continued from previous page)

```
name
GROND5_B1-1#000-1                    B1-1    111       -12.64          pvc
GROND5_B1-1#000-2                    B1-1    111        -3.14          pvc
GROND5_B1-2#000-1                    B1-2    111       -11.33          pvc
GROND5_B1-2#000-2                    B1-2    111        -1.83          pvc
GROND5_B1-3#000-1                    B1-3    111       -11.70          pvc
GROND5_B1-3#000-2                    B1-3    111        -2.20          pvc

                  variableDiameter
name
GROND5_B1-1#000-1               nee
GROND5_B1-1#000-2               nee
GROND5_B1-2#000-1               nee
GROND5_B1-2#000-2               nee
GROND5_B1-3#000-1               nee
GROND5_B1-3#000-2               nee

[6 rows x 44 columns]
```

Visualize ObsCollection

Visualize the `ObsCollection`.

[4]:
```
oc.plots.interactive_map()
```

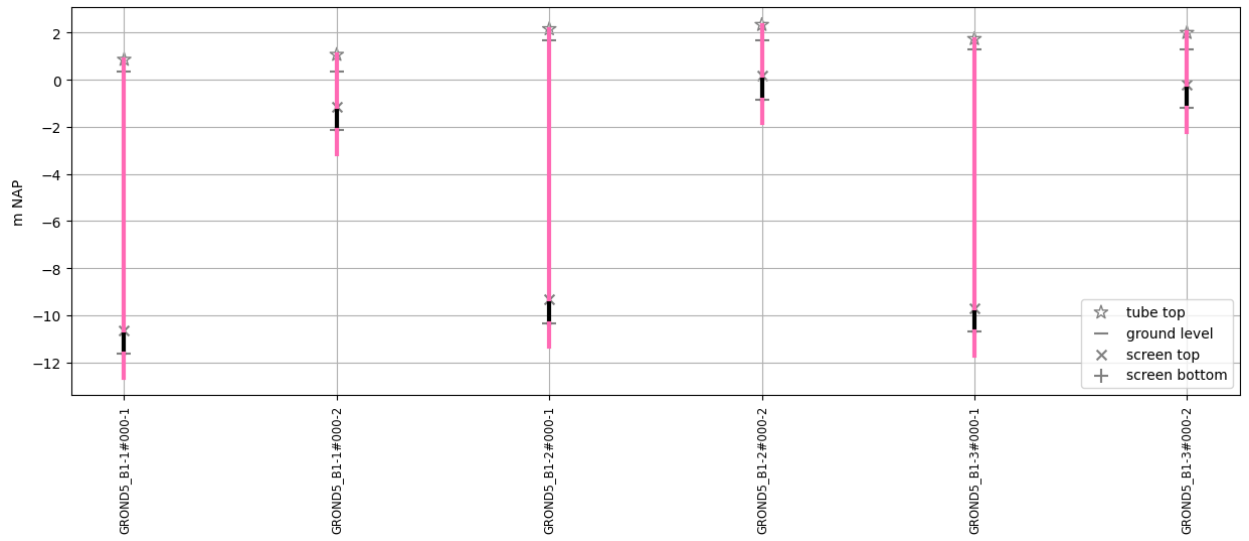
```
WARNING:hydropandas.extensions.plots:all observations in the collection are empty
INFO:hydropandas.extensions.plots:no iplot available for GROND5_B1-1#000-2
INFO:hydropandas.extensions.plots:no iplot available for GROND5_B1-2#000-2
INFO:hydropandas.extensions.plots:no iplot available for GROND5_B1-3#000-2
```

[4]:
```
<folium.folium.Map at 0x22c88d24760>
```

[5]:
```
oc.plots.section_plot(plot_obs=False)
```

```
INFO:hydropandas.extensions.plots:created sectionplot -> GROND5_B1-3#000-2
```

[5]:
```
(<Figure size 1500x500 with 1 Axes>, [<Axes: ylabel='m NAP'>])
```



Analyse ObsCollection

Visualize the `ObsCollection`.

First step is to check which columns have unique values for all wells. E.g. the owner should be the same for all wells. That requires that we drop the `obs` column, beceause `pd.nunique` cannot deal with that specific HydroPandas column-type.

[6]:
```
oc_temp = oc.copy().drop(["obs"], axis=1)
oc_unique = oc_temp.iloc[0][oc_temp.columns[oc_temp.nunique() <= 1]]
oc_unique
```

[6]:
```
source                                bronhouderportaal-bro
unit                                                  m NAP
metadata_available                                     True
initialFunction                                       stand
sedimentSumpLength                                      1.0
deliveryAccountableParty                                111
screenLength                                            1.0
deliveredVerticalPosition_offset                        0.0
horizontalPositioningMethod                 RTKGPS5tot10cm
localVerticalReferencePoint                           NAP
tubeStatus                                  gebruiksklaar
qualityRegime                                       IMBRO
tubeTopDiameter_unit                                   mm
groundLevelStable                                      ja
sockMaterial                                         geen
sedimentSumpPresent                                    ja
tubeTopPositioningMethod                    RTKGPS4tot10cm
tubeType                                     standaardbuis
tubePackingMaterial                            filtergrind
numberOfGeoOhmCables                                    0
deliveryContext                                        WW
groundLevelPositioningMethod          tachymetrie0tot10cm
```

(continues on next page)

(continued from previous page)

```
tubeTopDiameter                                  25.0
constructionStandard                          NEN5766
tubeGlue                                         geen
artesianWellCapPresent                             ja
wellHeadProtector                    kokerNietMetaal
owner                                              111
tubeMaterial                                       pvc
variableDiameter                                   nee
iplot_fname                                        NaN
Name: , dtype: object
```

[7]:
```
oc_non_unique = oc[oc.columns.drop(oc_temp.columns[oc_temp.nunique() <= 1])]
oc_non_unique
```

[7]:
```
                           x           y         filename  monitoring_well  \
name
GROND5_B1-1#000-1  56525.207  386749.698  GROND5_B1-1.xml  GROND5_B1-1#000
GROND5_B1-1#000-2  56525.207  386749.698  GROND5_B1-1.xml  GROND5_B1-1#000
GROND5_B1-2#000-1  56336.522  386967.299  GROND5_B1-2.xml  GROND5_B1-2#000
GROND5_B1-2#000-2  56336.522  386967.299  GROND5_B1-2.xml  GROND5_B1-2#000
GROND5_B1-3#000-1  56349.209  387156.262  GROND5_B1-3.xml  GROND5_B1-3#000
GROND5_B1-3#000-2  56349.209  387156.262  GROND5_B1-3.xml  GROND5_B1-3#000

                   tube_nr  screen_top  screen_bottom  ground_level  tube_top  \
name
GROND5_B1-1#000-1        1      -10.64         -11.64          0.36      0.86
GROND5_B1-1#000-2        2       -1.14          -2.14          0.36      1.07
GROND5_B1-2#000-1        1       -9.33         -10.33          1.67      2.16
GROND5_B1-2#000-2        2        0.17          -0.83          1.67      2.34
GROND5_B1-3#000-1        1       -9.70         -10.70          1.30      1.74
GROND5_B1-3#000-2        2       -0.20          -1.20          1.30      2.00

                                                                  obs  \
name
GROND5_B1-1#000-1  GroundwaterObs GROND5_B1-1#000-1
-----metadata...
GROND5_B1-1#000-2  GroundwaterObs GROND5_B1-1#000-2
-----metadata...
GROND5_B1-2#000-1  GroundwaterObs GROND5_B1-2#000-1
-----metadata...
GROND5_B1-2#000-2  GroundwaterObs GROND5_B1-2#000-2
-----metadata...
GROND5_B1-3#000-1  GroundwaterObs GROND5_B1-3#000-1
-----metadata...
GROND5_B1-3#000-2  GroundwaterObs GROND5_B1-3#000-2
-----metadata...

                  wellConstructionDate  plainTubePartLength  \
name
GROND5_B1-1#000-1           2020-06-23                11.50
GROND5_B1-1#000-2           2020-06-23                 2.21
GROND5_B1-2#000-1           2020-06-18                11.49
```

(continues on next page)

(continued from previous page)

```
GROND5_B1-2#000-2           2020-06-18                2.17
GROND5_B1-3#000-1           2020-06-22               11.44
GROND5_B1-3#000-2           2020-06-22                2.20

                 objectIdAccountableParty  tube_bottom        lat       lon
name
GROND5_B1-1#000-1                    B1-1       -12.64  51.461207  3.970150
GROND5_B1-1#000-2                    B1-1        -3.14  51.461207  3.970150
GROND5_B1-2#000-1                    B1-2       -11.33  51.463129  3.967375
GROND5_B1-2#000-2                    B1-2        -1.83  51.463129  3.967375
GROND5_B1-3#000-1                    B1-3       -11.70  51.464829  3.967504
GROND5_B1-3#000-2                    B1-3        -2.20  51.464829  3.967504
```

[8]:
```
# get statistics
oc_non_unique.describe()
```

[8]:
```
                 x              y   tube_nr  screen_top  screen_bottom  \
count      6.000000       6.000000  6.000000    6.000000       6.000000
mean   56403.646000  386957.753000  1.500000   -5.140000      -6.140000
std       94.331533     181.971242  0.547723    5.238309       5.238309
min    56336.522000  386749.698000  1.000000  -10.640000     -11.640000
25%    56339.693750  386804.098250  1.000000   -9.607500     -10.607500
50%    56349.209000  386967.299000  1.500000   -5.235000      -6.235000
75%    56481.207500  387109.021250  2.000000   -0.435000      -1.435000
max    56525.207000  387156.262000  2.000000    0.170000      -0.830000

       ground_level  tube_top  plainTubePartLength  tube_bottom        lat  \
count      6.000000   6.00000             6.000000     6.000000   6.000000
mean       1.110000   1.69500             6.835000    -7.140000  51.463055
std        0.604053   0.60252             5.084749     5.238309   0.001621
min        0.360000   0.86000             2.170000   -12.640000  51.461207
25%        0.595000   1.23750             2.202500   -11.607500  51.461687
50%        1.300000   1.87000             6.825000    -7.235000  51.463129
75%        1.577500   2.12000            11.477500    -2.435000  51.464404
max        1.670000   2.34000            11.500000    -1.830000  51.464829

            lon
count  6.000000
mean   3.968343
std    0.001401
min    3.967375
25%    3.967407
50%    3.967504
75%    3.969489
max    3.970150
```

Check the usage of `tube_nr`. Has tube number one the lowest `screen_bottom` and lowest `screen_top`?

[9]:
```
lst_lowest_tube = []
for monitoring_well in oc.monitoring_well.unique():
    oc_mw = oc.loc[oc.monitoring_well == monitoring_well]
```

(continues on next page)

(continued from previous page)

```
    lowest_screen_bottom_tube_nr = oc_mw.loc[
        oc_mw.screen_bottom == oc_mw.screen_bottom.min(), "tube_nr"
    ].values[0]

    lowest_screen_top_tube_nr = oc_mw.loc[
        oc_mw.screen_top == oc_mw.screen_top.min(), "tube_nr"
    ].values[0]

    lst_lowest_tube.append(
        [monitoring_well, lowest_screen_bottom_tube_nr, lowest_screen_top_tube_nr]
    )

df_lowest_tube = pd.DataFrame(
    lst_lowest_tube,
    columns=[
        "monitoring_well",
        "lowest_screen_bottom_tube_nr",
        "lowest_screen_top_tube_nr",
    ],
).set_index("monitoring_well")

df_lowest_tube
```

[9]:

monitoring_well	lowest_screen_bottom_tube_nr	lowest_screen_top_tube_nr
GROND5_B1-1#000	1	1
GROND5_B1-2#000	1	1
GROND5_B1-3#000	1	1

Upload to Bronhouderportaal BRO

Upload the XML-files to Bronhouderportaal BRO via the website.

[ ]:

Exploring groundwater data from Lizard

In this notebook, you will experiment how to use the `hydropandas` package to access, visualize and explore meta data from Lizard, a cloud datawarehouse that stores all groundwater observations from Vitens. Vitens is the largest drinking water company in the Netherlands, and it has more than 10.000 groundwater wells and more than 50.000 timeseries in its datawarehouse. The data spans from the 1930's to the present, and it is constantly updated with new observations. Vitens also validates the data using ArtDiver and provides quality flags and comments for each observation. The data is open to the public and you can find more information at https://vitens.lizard.net.

Feel free to customize and expand upon this introduction as needed. Happy coding!

Notebook contents

1. Find groundwater wells on a map
2. Analyse groundwater observations
3. Build a Pastas model

[1]:
```
import logging
from IPython.display import HTML

import pastas as ps
import pandas as pd
import hydropandas as hpd
```

[2]:
```
hpd.util.get_color_logger("INFO")
```

[2]:
```
<RootLogger root (INFO)>
```

Get observations from extent

Use `ObsCollection` to find monitoring wells by specifying a geographical extent in Rijksdriehoeks coordinates.

[3]:
```
my_extent = (137000, 138000, 458000, 459000)
oc = hpd.read_lizard(extent=my_extent)
```

```
Number of monitoring wells: 1
Number of pages: 1
```

```
Page: 100%|| 1/1 [00:00<00:00,  2.06it/s]
monitoring well: 100%|| 1/1 [00:08<00:00,  8.59s/it]
```

Visualize all groundwater wells inside the extent on a map (visualize the `ObsCollection`). The markers are clickable to show a preview of the availables observations.

[4]:
```
oc.plots.interactive_map(color="red", zoom_start=15, tiles="Esri.WorldImagery")
```

[4]:
```
<folium.folium.Map at 0x2200ae72fa0>
```

Print all the retrieved groundwater wells and tubes, and make a plot of the observations.

[5]:
```
oc
```

[5]:
```
                        x              y filename  source   unit  \
name
UPWP016001  137401.64297  458893.683528           lizard  m NAP
UPWP016003  137401.64297  458893.683528           lizard  m NAP
UPWP016002  137401.64297  458893.683528           lizard  m NAP

            monitoring_well  tube_nr  screen_top  screen_bottom  ground_level  \
name
UPWP016001        B31H0580        1      -22.43         -24.43          1.58
UPWP016003        B31H0580        3      -65.43         -67.43          1.58
UPWP016002        B31H0580        2      -53.93         -55.93          1.58
```

(continues on next page)

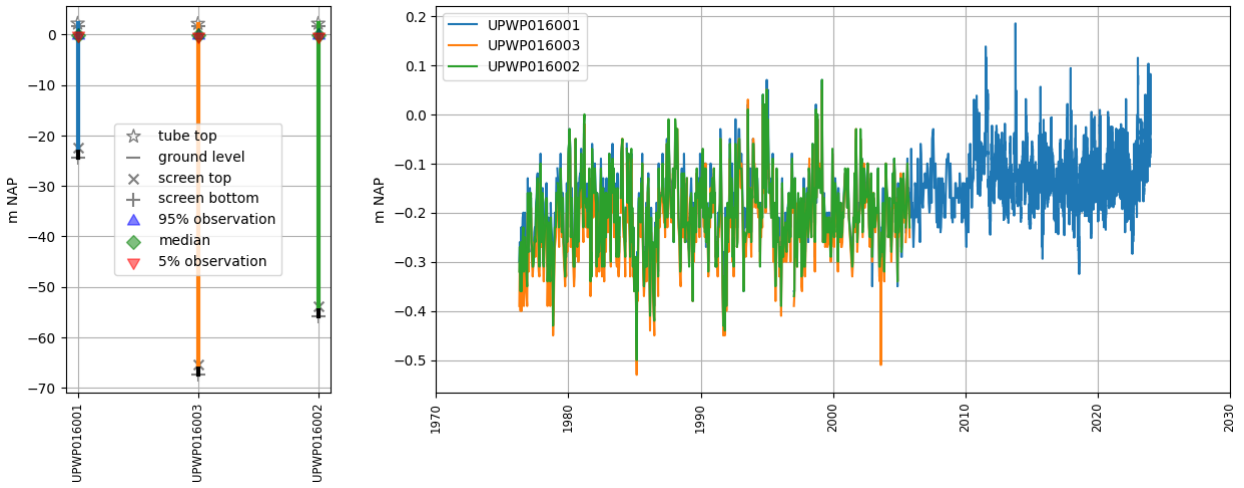
(continued from previous page)

```
            tube_top  metadata_available  \
name
UPWP016001     2.198                True
UPWP016003     2.141                True
UPWP016002     2.178                True

                                                          obs        lat  \
name
UPWP016001  GroundwaterObs UPWP016001
-----metadata------
...  52.117985
UPWP016003  GroundwaterObs UPWP016003
-----metadata------
...  52.117985
UPWP016002  GroundwaterObs UPWP016002
-----metadata------
...  52.117985

                lon
name
UPWP016001  5.13026
UPWP016003  5.13026
UPWP016002  5.13026
```

[6]:
```
oc.plots.section_plot(plot_obs=True)
```

```
INFO:hydropandas.extensions.plots:created sectionplot -> UPWP016001
INFO:hydropandas.extensions.plots:created sectionplot -> UPWP016003
INFO:hydropandas.extensions.plots:created sectionplot -> UPWP016002
```

[6]:
```
(<Figure size 1500x500 with 2 Axes>,
 [<Axes: ylabel='m NAP'>, <Axes: ylabel='m NAP'>])
```

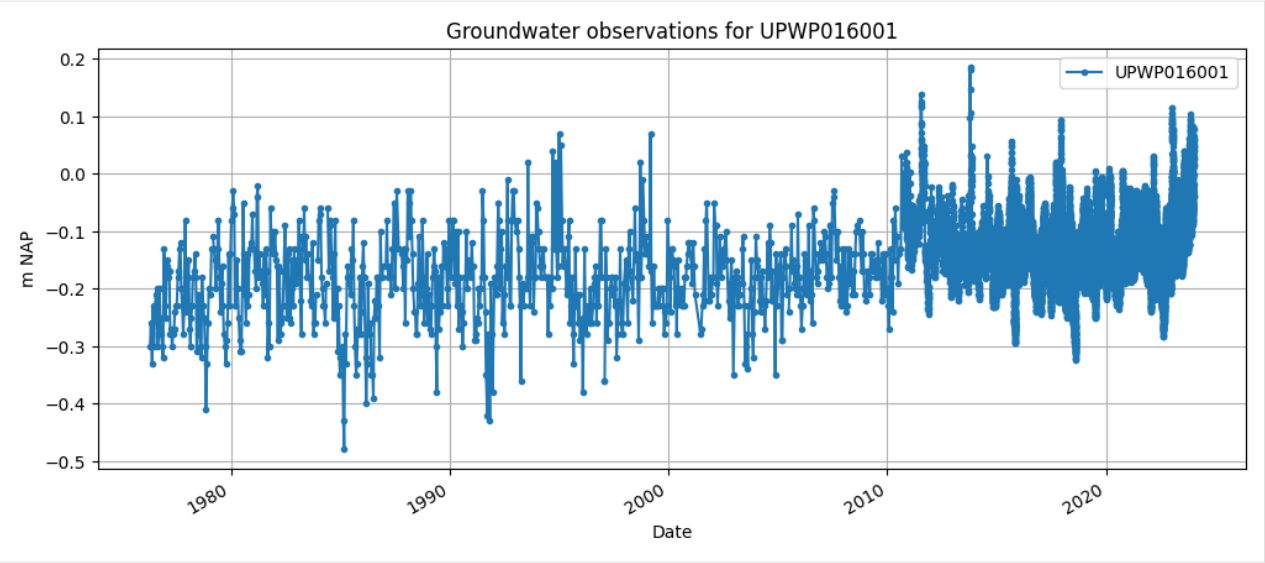


Groundwater observations

Now lets download the groundwater level observation using the `from_lizard` function of a `GroundwaterObs` object. The code below reads the groundwater level timeseries for the well UPWP016 from Lizard and makes a plot.

[7]:
```
gw_lizard = hpd.GroundwaterObs.from_lizard(
    "UPWP016", tmin="1900-01-01", tmax="2030-01-01"
)
print(gw_lizard)

ax = gw_lizard["value"].plot(
    figsize=(12, 5),
    marker=".",
    grid=True,
    label=gw_lizard.name,
    legend=True,
    xlabel="Date",
    ylabel="m NAP",
    title="Groundwater observations for " + gw_lizard.name,
)
```

```
GroundwaterObs UPWP016001
-----metadata------
name : UPWP016001
x : 137401.64297031244
y : 458893.6835282785
filename :
source : lizard
unit : m NAP
monitoring_well : B31H0580
tube_nr : 1
screen_top : -22.43
screen_bottom : -24.43
ground_level : 1.58
tube_top : 2.198
metadata_available : True

-----time series------
                     value         flag comment
peil_datum_tijd
1976-04-15 12:00:00 -0.300  betrouwbaar
1976-04-28 12:00:00 -0.260  betrouwbaar
1976-05-18 12:00:00 -0.330  betrouwbaar
1976-06-02 12:00:00 -0.230  betrouwbaar
1976-06-17 12:00:00 -0.300  betrouwbaar
...                    ...          ...     ...
2024-01-03 21:00:00  0.059  betrouwbaar
2024-01-04 00:00:00  0.059  betrouwbaar
2024-01-04 03:00:00  0.062  betrouwbaar
2024-01-04 06:00:00  0.066  betrouwbaar
2024-01-04 09:00:00  0.078  betrouwbaar

[29259 rows x 3 columns]
```



The groundwater observations contain a validation flag per timestamp. These can 'betrouwbaar' (reliable), 'onbetrouwbaar' (unreliable) en 'onbeslist' (unvalidated). Below flags of the timeseries are shown as a percentage, and the unreliable timestamps are printed.

[8]:
```
print(gw_lizard["flag"].value_counts(normalize=True) * 100)
gw_lizard[gw_lizard["flag"] == "onbetrouwbaar"]
```

```
UPWP016001
betrouwbaar      99.979493
onbetrouwbaar     0.017089
onbeslist         0.003418
Name: proportion, dtype: float64
```

[8]:
```
GroundwaterObs UPWP016001
-----metadata------
name : UPWP016001
x : 137401.64297031244
y : 458893.6835282785
filename :
source : lizard
unit : m NAP
monitoring_well : B31H0580
tube_nr : 1
screen_top : -22.43
screen_bottom : -24.43
ground_level : 1.58
tube_top : 2.198
metadata_available : True

-----time series------
                     value           flag                 comment
peil_datum_tijd
1979-02-16 12:00:00    NaN  onbetrouwbaar  Geen waarneming (-90)
1983-11-29 12:00:00    NaN  onbetrouwbaar  Geen waarneming (-90)
1984-05-14 12:00:00    NaN  onbetrouwbaar  Geen waarneming (-90)
1996-12-28 12:00:00    NaN  onbetrouwbaar  Geen waarneming (-90)
```

(continues on next page)

(continued from previous page)

```
2005-06-14 12:00:00    NaN  onbetrouwbaar  Geen waarneming (-90)
```

Pastas model

Lets make a Pastas model for this groundwater well (starting from 2015) and use the nearest KNMI station for meteorological data

[9]:
```
# Get the precipitation and evaporation data from the KNMI
precipitation = hpd.PrecipitationObs.from_knmi(
    xy=(gw_lizard.x, gw_lizard.y),
    start=gw_lizard.index[0],
    end=gw_lizard.index[-1],
    fill_missing_obs=True,
)
evaporation = hpd.EvaporationObs.from_knmi(
    xy=(gw_lizard.x, gw_lizard.y),
    meteo_var="EV24",
    start=gw_lizard.index[0],
    end=gw_lizard.index[-1],
    fill_missing_obs=True,
)

# Create a Pastas Model
ml = ps.Model(gw_lizard["value"], name=gw_lizard.name)

# Add the recharge data as explanatory variable
ts1 = ps.RechargeModel(
    precipitation["RH"].resample("D").first(),
    evaporation["EV24"].resample("D").first(),
    ps.Gamma(),
    name="rainevap",
    settings=("prec", "evap"),
)

# Add the stressmodel to the model and solve for period after 2015
ml.add_stressmodel(ts1)
ml.solve(tmin="2015")
ml.plots.results(figsize=(10, 6))
```

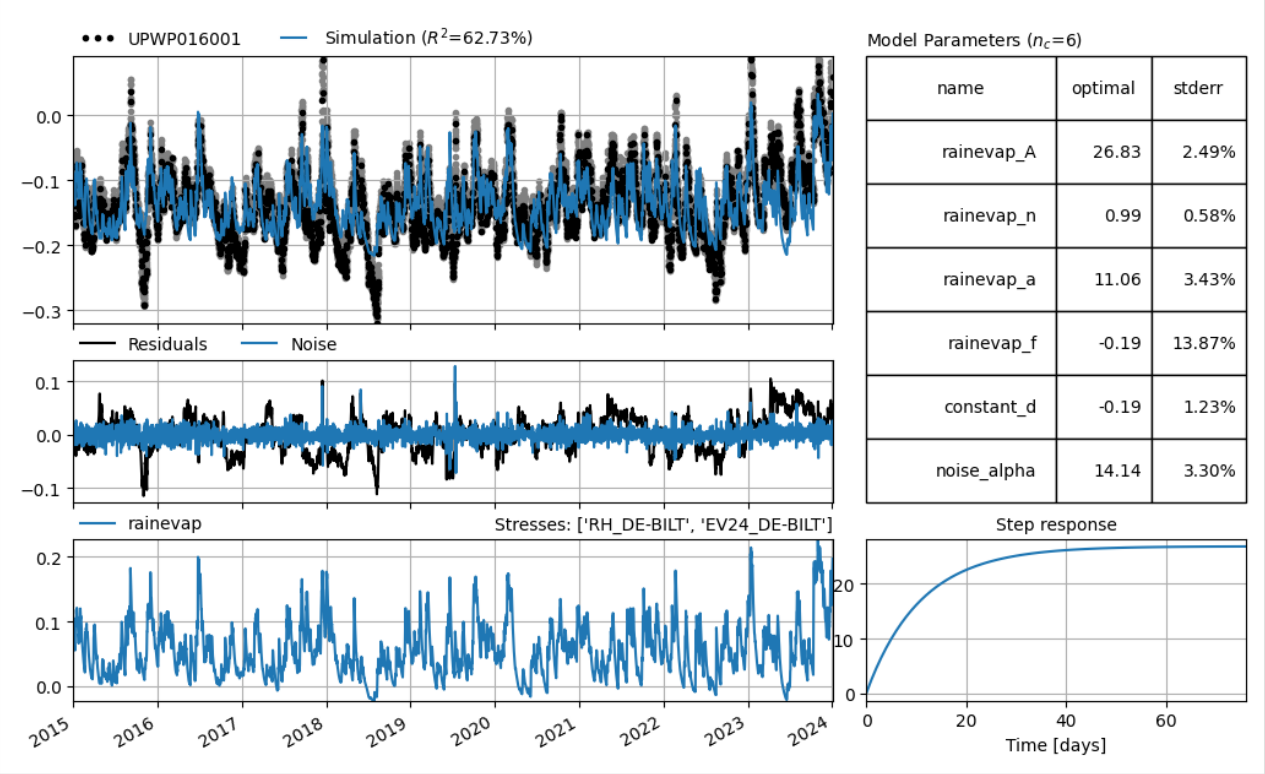
```
INFO:hydropandas.io.knmi:get KNMI data from station nearest to coordinates (137401.
↪64297031244, 458893.6835282785) and meteovariable RH
WARNING:hydropandas.io.knmi:changing end_date to 2024-01-04
INFO:hydropandas.io.knmi:get KNMI data from station nearest to coordinates (137401.
↪64297031244, 458893.6835282785) and meteovariable EV24
WARNING:hydropandas.io.knmi:changing end_date to 2024-01-04
```

```
WARNING: The Time Series 'UPWP016001' has nan-values. Pastas will use the fill_nan␣
↪settings to fill up the nan-values.
```

```
WARNING:pastas.timeseries:The Time Series 'UPWP016001' has nan-values. Pastas will use␣
↪the fill_nan settings to fill up the nan-values.
```

```
INFO: Time Series 'UPWP016001': 11 nan-value(s) was/were found and filled with: drop.
INFO:pastas.timeseries:Time Series 'UPWP016001': 11 nan-value(s) was/were found and␣
↪filled with: drop.
INFO: Time Series 'RH_DE-BILT' was extended in the future to 2024-01-04 00:00:00 with␣
↪the mean value (0.0024) of the time series.
INFO:pastas.timeseries:Time Series 'RH_DE-BILT' was extended in the future to 2024-01-04␣
↪00:00:00 with the mean value (0.0024) of the time series.
INFO: Time Series 'EV24_DE-BILT' was extended in the future to 2024-01-04 00:00:00 with␣
↪the mean value (0.0017) of the time series.
INFO:pastas.timeseries:Time Series 'EV24_DE-BILT' was extended in the future to 2024-01-
↪04 00:00:00 with the mean value (0.0017) of the time series.
Fit report UPWP016001              Fit Statistics
================================================
nfev    33                     EVP          62.73
nobs    3291                   R2            0.63
noise   True                   RMSE          0.03
tmin    2015-01-01 00:00:00    AIC      -29158.55
tmax    2024-01-04 09:00:00    BIC      -29121.96
freq    D                      Obj           0.23
warmup  3650 days 00:00:00     ___
solver  LeastSquares           Interp.         No

Parameters (6 optimized)
================================================
               optimal   stderr     initial  vary
rainevap_A   26.825256   ±2.49%  198.612463  True
rainevap_n    0.986379   ±0.58%    1.000000  True
rainevap_a   11.059931   ±3.43%   10.000000  True
rainevap_f   -0.192514  ±13.87%   -1.000000  True
constant_d   -0.193830   ±1.23%   -0.134993  True
noise_alpha  14.137844   ±3.30%    1.000000  True
```

[9]:
```
[<Axes: xlabel='peil_datum_tijd'>,
 <Axes: xlabel='peil_datum_tijd'>,
 <Axes: title={'right': "Stresses: ['RH_DE-BILT', 'EV24_DE-BILT']"}>,
 <Axes: title={'center': 'Step response'}, xlabel='Time [days]'>,
 <Axes: title={'left': 'Model Parameters ($n_c$=6)'}>]
```



Model Parameters (n_c=6)

name	optimal	stderr
rainevap_A	26.83	2.49%
rainevap_n	0.99	0.58%
rainevap_a	11.06	3.43%
rainevap_f	-0.19	13.87%
constant_d	-0.19	1.23%
noise_alpha	14.14	3.30%

1.3 User guide

1.3.1 The *Obs* class

The Obs class holds the measurements and metadata for one timeseries. There are currently 5 specific Obs classes for different types of measurements:

- GroundwaterObs: for groundwater measurements
- WaterQualityObs: for (ground)water quality measurements
- WaterlvlObs: for surface water level measurements
- ModelObs: for hydropandas from a MODFLOW model
- MeteoObs: for hydropandas from KNMI or other meteorological services
- PrecipitationObs: for precipitation observations, subclass of MeteoObs
- EvaporationObs: for evaporation observations, subclass of MeteoObs

Each of these Obs classes is essentially a pandas DataFrame with additional methods and attributes related to the type of measurement that it holds. The classes also contain specific methods to read data from specific sources.

1.3.2 The *ObsCollection* class

The ObsCollection class, as the name implies, represents a collection of Obs classes, e.g. 10 timeseries of the groundwater level in a certain area. The ObsCollection is also a pandas DataFrame in which each timeseries is stored in a different row. Each row contains metadata (e.g. latitude and longitude of the observation point) and the Obs object (DataFrame) that holds the measurements. It is recommended to let an ObsCollection contain only one Obs type, e.g. to create an ObsCollection for 10 GroundwaterObs, and a separate ObsCollection for 5 WaterlvlObs.

Like the Obs class, the ObsCollection class contains a bunch of methods for reading data from different sources.

1.4 hydropandas

1.4.1 Subpackages

hydropandas.extensions package

hydropandas.extensions.accessor module

Copied from ./pandas/core/accessor.py.

class hydropandas.extensions.accessor.**CachedAccessor**(*name*, *accessor*)

Bases: object

Custom property-like object (descriptor) for caching accessors.

Parameters

- **name** (*str*) – The namespace this will be accessed under, e.g. df.foo
- **accessor** (*cls*) – The class with the extension methods. The class' __init__ method should expect one of a Series, DataFrame or Index as the single argument data

hydropandas.extensions.accessor.**register_obs_accessor**(*name*)

hydropandas.extensions.accessor.**register_obscollection_accessor**(*name*)

hydropandas.extensions.geo module

class hydropandas.extensions.geo.**GeoAccessor**(*oc_obj*)

Bases: object

get_bounding_box(*xcol='x'*, *ycol='y'*, *buffer=0*)

returns the bounding box of all observations.

Parameters

- **xcol** (*str, optional*) – column name with x values
- **ycol** (*str, optional*) – column name with y values
- **buffer** (*int or float, optional*) – add a buffer around the bouding box from the observations

Returns

coordinates of bouding box

Return type
xmin, ymin, xmax, ymax

`get_distance_to_point`(*point*, *xcol='x'*, *ycol='y'*)

get distance of every observation to a point.

Parameters

- **point** (*shapely.geometry.point.Point*) – point geometry
- **xcol** (*str, optional*) – x column in self._obj used to get x coordinates
- **ycol** (*str, optional*) – y column in self._obj used to get y coordinates

Returns
distance to the point for every observation in self._obj

Return type
pd.Series

`get_extent`(*xcol='x'*, *ycol='y'*, *buffer=0*)

returns the extent of all observations.

Parameters

- **xcol** (*str, optional*) – column name with x values
- **ycol** (*str, optional*) – column name with y values
- **buffer** (*int or float, optional*) – add a buffer around the bouding box from the observations

Returns
coordinates of bouding box

Return type
xmin, xmax, ymin, ymax

`get_lat_lon`(*in_epsg='epsg:28992'*, *out_epsg='epsg:4326'*)

get lattitude and longitude from x and y attributes.

Parameters

- **in_epsg** (*str, optional*) – epsg code of current x and y attributes, default (RD new)
- **out_epsg** (*str, optional*) – epsg code of desired output, default lat/lon

Returns
with columns 'lat' and 'lon'

Return type
pandas.DataFrame

`get_nearest_line`(*gdf=None*, *xcol_obs='x'*, *ycol_obs='y'*, *multiple_lines='error'*)

get nearest line for each point in the obs collection. Function calls the nearest_polygon function.

Parameters

- **gdf** (*GeoDataFrame*) – dataframe with line features
- **xcol_obs** (*str, optional*) – x column in self._obj used to get geometry
- **ycol_obs** (*str, optional*) – y column in self._obj used to get geometry
- **multiple_lines** (*str, optional*) – keyword on how to deal with multiple lines being nearest. Options are:

'error' -> raise a ValueError 'keep_all' -> return the indices of multiple lines as a string seperated by a comma 'keep_first' -> return the index of the first line

Returns
with columns 'nearest polygon' and 'distance nearest polygon'

Return type
pandas.DataFrame

get_nearest_point(*obs_collection2=None*, *gdf2=None*, *xcol_obs1='x'*, *ycol_obs1='y'*, *xcol_obs2='x'*, *ycol_obs2='y'*)

get nearest point of another obs collection for each point in the current obs collection.

Parameters

- **obs_collection2** (*ObsCollection, optional*) – collection of observations of which the nearest point is found
- **gdf2** (*GeoDataFrame, optional*) – dataframe to look for nearest observation point
- **xcol_obs1** (*str, optional*) – x column in self._obj used to get geometry
- **ycol_obs1** (*str, optional*) – y column in self._obj used to get geometry
- **xcol_obs2** (*str, optional*) – x column in obs_collection2 used to get geometry
- **ycol_obs2** (*str, optional*) – y column in self._obj used to get geometry

Returns
with columns 'nearest point' and 'distance nearest point'

Return type
pandas.DataFrame

get_nearest_polygon(*gdf=None*, *xcol_obs='x'*, *ycol_obs='y'*, *multiple_polygons='error'*)

get nearest polygon for each point in the obs collection. Function also works for lines instead of polygons.

Parameters

- **gdf** (*GeoDataFrame*) – dataframe with polygon features
- **xcol_obs** (*str, optional*) – x column in self._obj used to get geometry
- **ycol_obs** (*str, optional*) – y column in self._obj used to get geometry
- **multiple_polygons** (*str, optional*) – keyword on how to deal with multiple polygons being nearest. Options are:

  'error' -> raise a ValueError 'keep_all' -> return the indices of multiple polygons as a string seperated by a comma 'keep_first' -> return the index of the first polygon

Returns
with columns 'nearest polygon' and 'distance nearest polygon'

Return type
pandas.DataFrame

set_lat_lon(*in_epsg='epsg:28992'*, *out_epsg='epsg:4326'*, *add_to_meta=True*)

create columns with lat and lon values of the observation points.

Parameters

- **in_epsg** (*str, optional*) – epsg code of current x and y attributes, default (RD new)
- **out_epsg** (*str, optional*) – epsg code of desired output, default lat/lon

- **add_to_meta** (*bool, optional*) – if True the lat and lon values are added to the observation meta dictionary. The default is True.

Return type
None.

within_extent(*extent*, *inplace=False*)

Slice ObsCollection by extent.

Parameters
extent (*tuple*) – format (xmin, xmax, ymin, ymax), default dis.sr.get_extent() format

Returns
new_oc – ObsCollection with observations within extent

Return type
obs_collection.ObsCollection

within_polygon(*gdf=None*, *shapefile=None*, *inplace=False*, ***kwargs*)

Slice ObsCollection by checking if points are within a shapefile.

Parameters

- **gdf** (*GeoDataFrame, optional*) – geodataframe containing a single polygon
- **shapefile** (*str, optional*) – Not yet implemented
- **inplace** (*bool, default False*) – Modify the ObsCollection in place (do not create a new object).
- ****kwargs** – kwargs will be passed to the self._obj.to_gdf() method

Returns
new_oc – ObsCollection with observations within polygon

Return type
obs_collection.ObsCollection

class hydropandas.extensions.geo.**GeoAccessorObs**(*obs*)

Bases: object

get_lat_lon(*in_epsg='epsg:28992'*, *out_epsg='epsg:4326'*)

get lattitude and longitude from x and y attributes.

Parameters

- **in_epsg** (*str, optional*) – epsg code of current x and y attributes, default (RD new)
- **out_epsg** (*str, optional*) – epsg code of desired output, default lat/lon

Returns
lon, lat

Return type
longitude and lattitude of x, y coordinates

hydropandas.extensions.gwobs module

class hydropandas.extensions.gwobs.**GeoAccessorObs**(*obs*)

Bases: object

get_modellayer_modflow(*gwf=None*, *ds=None*, *left=-999*, *right=999*)

Add modellayer to meta dictionary.

Parameters

- **gwf** (*flopy.mf6.modflow.mfgwf.ModflowGwf*) – modflow model
- **ds** (*xarray.Dataset*) – xarray Dataset with with top and bottoms, must have dimensions 'x' and 'y' and variables 'top' and 'bot'

Returns
modellayer

Return type
int

get_regis_layer()

find the name of the REGIS layer based on the tube screen depth.

Returns
name of REGIS layer

Return type
str

class hydropandas.extensions.gwobs.**GwObsAccessor**(*oc_obj*)

Bases: object

get_modellayers(*gwf=None*, *ds=None*)

Get the modellayer per observation. The layers can be obtained from the modflow model or can be defined in zgr.

Parameters

- **gwf** (*flopy.mf6.modflow.mfgwf.ModflowGwf*) – modflow model
- **ds** (*xarray.Dataset*) – xarray Dataset with with top and bottoms, must have dimensions 'x' and 'y' and variables 'top' and 'bot'

Return type
pd.Series with the modellayers of each observation

get_regis_layers()

Get the regis layer per observation.

Return type
pd.Series with the names of the regis layer of each observation

set_tube_nr(*radius=1*, *xcol='x'*, *ycol='y'*, *if_exists='error'*, *add_to_meta=False*)

This method computes the tube numbers based on the location of the observations.

Then it sets the value of the tube number:

- in the ObsCollection dataframe
- as the attribute of an Obs object
- in the meta dictionary of the Obs object (only if add_to_meta is True)

This method is useful for groundwater observations. If two or more observation points are close to each other they will be seen as one monitoring_well with multiple tubes. The tube_nr is based on the 'screen_bottom' attribute of the observations in such a way that the deepest tube has the highest tube number.

Parameters

- **radius** (*int, optional*) – max distance between two observations to be seen as one location, by default 1
- **xcol** (*str, optional*) – column name with x coordinates, by default 'x'
- **ycol** (*str, optional*) – column name with y coordinates, by default 'y'
- **if_exists** (*str, optional*) – what to do if an observation point already has a tube_nr, options: 'error', 'replace' or 'keep', by default 'error'
- **add_to_meta** (*bool, optional*) – if True the tube_nr is added to the meta dictionary of an observation. The default is False.

Raises

RuntimeError – if the column tube_nr exists and if_exists='error' an error is raised

set_tube_nr_monitoring_well(*loc_col*, *radius=1*, *xcol='x'*, *ycol='y'*, *if_exists='error'*, *add_to_meta=False*)

This method sets the tube_nr and monitoring_well name of an observation point based on the location of the observations.

When two or more tubes are close to another, as defined by radius, they are set to the same *monitoring_well* and an increasing *tube_nr* based on depth.

The value of the tube_nr and the monitoring_well are set:

- in the ObsCollection dataframe
- as the attribute of an Obs object
- in the meta dictionary of the Obs object (only if add_to_meta is True)

This method is useful for groundwater observations. If two or more observation points are close to each other they will be seen as one monitoring_well with multiple tubes. The tube_nr is based on the 'screen_bottom' attribute of the observations in such a way that the deepest tube has the highest tube number. The monitoring_well is based on the named of the loc_col of the screen with the lowest tube_nr.

Parameters

- **loc_col** (*str*) – the column name with the names to use for the monitoring_well
- **radius** (*int, optional*) – max distance between two observations to be seen as one location, by default 1
- **xcol** (*str, optional*) – column name with x coordinates, by default 'x'
- **ycol** (*str, optional*) – column name with y coordinates, by default 'y'
- **if_exists** (*str, optional*) – what to do if an observation point already has a tube_nr, options: 'error', 'replace' or 'keep', by default 'error'
- **add_to_meta** (*bool, optional*) – if True the tube_nr and location are added to the meta dictionary of an observation. The default is False.

Raises

RuntimeError – if the column tube_nr exists and if_exists='error' an error is raised

hydropandas.extensions.gwobs.**check_if_var_is_invalid**(*var*)

hydropandas.extensions.gwobs.**get_model_layer_z**(*z*, *zvec*, *left=-999*, *right=999*)

Get index of model layer based on elevation.

Assumptions:

- the highest value in zvec is the top of model layer 0.
- if z is equal to the bottom of a layer, the model layer above that layer is assigned.

Parameters

- **z** (*int, float*) – elevation.
- **zvec** (*list, np.array*) – elevations of model layers. shape is nlay + 1
- **left** (*int, optional*) – if z is below the lowest value in zvec, this value is returned. The default is -999.
- **right** (*TYPE, optional*) – if z is above the highest value in zvec, this value is returned. The default is 999.

Returns

int – model layer

Return type

int

Examples

```
>>> zvec = [0, -10, -20, -30]
>>> get_model_layer_z(-5, zvec)
0
```

```
>>> get_model_layer_z(-25, zvec)
2
```

```
>>> get_model_layer_z(-50, zvec)
-999
```

```
>>> get_model_layer_z(100, zvec)
999
```

```
>>> get_model_layer_z(-20, zvec)
1
```

hydropandas.extensions.gwobs.**get_modellayer_from_screen_depth**(*ftop*, *fbot*, *zvec*, *left=-999*, *right=999*)

Parameters

- **ftop** (*int or float*) – top of screen.
- **fbot** (*int or float*) – bottom of screen, has to be lower than ftop.
- **zvec** (*list or np.array*) – elevations of the modellayers at the location of the tube.

- **left** (*int, optional*) – value to return if tube screen is below the modellayers. The default is -999.
- **right** (*int, optional*) – value to return if tube screen is above the modellayers. The default is-999.

Raises
ValueError – raised if something unexpected happens.

Returns
modellayer.

Return type
int or np.nan

Examples

```
>>> zvec = [0, -10, -20, -30, -40]
>>> get_modellayer_from_screen_depth(-5, -7, zvec)
0
```

```
>>> get_modellayer_from_screen_depth(-25, -27, zvec)
2
```

```
>>> get_modellayer_from_screen_depth(-15, -27, zvec)
2
```

```
>>> get_modellayer_from_screen_depth(-5, -27, zvec)
1
```

```
>>> get_modellayer_from_screen_depth(-5, -37, zvec)
1
```

```
>>> get_modellayer_from_screen_depth(15, -5, zvec)
0
```

```
>>> get_modellayer_from_screen_depth(15, 5, zvec)
999
```

```
>>> get_modellayer_from_screen_depth(-55, -65, zvec)
-999
```

```
>>> get_modellayer_from_screen_depth(15, -65, zvec)
nan
```

```
>>> get_modellayer_from_screen_depth(None, -7, zvec)
0
```

```
>>> get_modellayer_from_screen_depth(None, None, zvec)
nan
```

hydropandas.extensions.gwobs.**get_zvec**(*x*, *y*, *gwf=None*, *ds=None*)

get a list with the vertical layer boundaries at a point in the model.

Parameters

- **x** (*int or float*) – x coordinate.
- **y** (*int or float*) – y coordinate.
- **gwf** (*flopy.mf6.modflow.mfgwf.ModflowGwf*) – modflow model with top and bottoms
- **ds** (*xarray.Dataset*) – xarray Dataset typically created in nlmod. Must have variables 'top' and 'botm'.

Raises

NotImplementedError – not all grid types are supported yet.

Returns

zvec – list of vertical layer boundaries. length is nlay + 1.

Return type

list

hydropandas.extensions.plots module

class hydropandas.extensions.plots.**CollectionPlots**(*oc_obj*)

Bases: object

interactive_map(*plot_dir='figures'*, *m=None*, *tiles='OpenStreetMap'*, *fname=None*, *per_monitoring_well=True*, *color='blue'*, *legend_name=None*, *add_legend=True*, *map_label=''*, *map_label_size=20*, *col_name_lat='lat'*, *col_name_lon='lon'*, *zoom_start=13*, *create_interactive_plots=True*, ***kwargs*)

Create an interactive map with interactive plots using folium and bokeh.

Notes

Some notes on this method:

- if you want to have multiple obs collections on one folium map, only the last one should have add_legend = True to create a correct legend
- the color of the observation point on the map is now the same color as the line of the observation measurements. Also a built-in color cycle is used for different measurements at the same monitoring_well.

Parameters

- **plot_dir** (*str*) – directory used for the folium map and bokeh plots
- **m** (*folium.Map, str, optional*) – current map to add observations too, if None a new map is created
- **tiles** (*str, optional*) – background tiles, default is openstreetmap
- **fname** (*str, optional*) – name of the folium map
- **per_monitoring_well** (*bool, optional*) – if True plot multiple tubes at the same monitoring well in one figure
- **color** (*str, optional*) – color of the observation points on the map

- **legend_name** (*str, optional*) – the name of the observation points shown in the map legend
- **add_legend** (*boolean, optional*) – add a legend to a plot
- **map_label** (*str, optional*) – add a label to the monitoring wells on the map, the label should be 1. the attribute of an observation 2. the key in the meta attribute of the observation 3. a generic label for each observation in this collection. A label is only added if map_label is not ''. The default is ''.
- **map_label_size** (*int, optional*) – label size of the map_label in pt.
- **col_name_lat** (*str, optional*) – name of the column in the obs_collection dic with the lat values of the observation points
- **col_name_lon** (*str, optional*) – see col_name_lat
- **zoom_start** (*int, optional*) – start zoom level of the folium ma
- **create_interactive_plots** (*boolean, optional*) – if True interactive plots will be created, if False the iplot_fname in the meta ditctionary of the observations is used.
- ****kwargs** – will be passed to the interactive_plots method options are:
  - cols : list of str or None
  - hoover_names : list of str
  - plot_legend_names : list of str
  - plot_freq : list of str
  - markers : list of str
  - hoover_names : list of str
  - plot_colors : list of str
  - ylabel : str
  - add_screen_to_legend : boolean
  - tmin : dt.datetime
  - tmax : dt.datetime

Returns
m – the folium map

Return type
folium.Map

interactive_plots(*savedir='figures'*, *tmin=None*, *tmax=None*, *per_monitoring_well=True*, ***kwargs*)

Create interactive plots of the observations using bokeh.

Parameters

- **savedir** (*str*) – directory used for the folium map and bokeh plots
- **tmin** (*dt.datetime, optional*) – start date for timeseries plot
- **tmax** (*dt.datetime, optional*) – end date for timeseries plot
- **per_monitoring_well** (*bool, optional*) – if True plot multiple tubes at the same monitoring_well in one figure
- ****kwargs** – will be passed to the Obs.interactive_plot method, options include:

- cols : list of str or None
- hoover_names : list of str
- plot_freq : list of str
- plot_legend_names : list of str
- markers : list of str
- hoover_names : list of str
- plot_colors : list of str
- ylabel : str
- add_screen_to_legend : boolean

section_plot(*tmin=None*, *tmax=None*, *cols=(None,)*, *section_colname_x=None*, *section_label_x=None*, *section_well_layout_color='gray'*, *section_markersize=100*, *ylabel='auto'*, *fn_save=None*, *check_obs_close_to_screen_bottom=True*, *plot_well_layout_markers=True*, *plot_obs=True*)

Create plot with well layout (left) en observations (right).

Parameters

- **tmin** (*dt.datetime, optional*) – start date for timeseries plot
- **tmax** (*dt.datetime, optional*) – end date for timeseries plot
- **cols** (*tuple of str or None, optional*) – the columns of the observation to plot. The first numeric column is used if cols is None, by default None.
- **section_colname_x** (*str, optional*) – column used for x position on section plot, when None order collection is used
- **section_label_x** (*str, optional*) – label applied to x-axis in section plot
- **section_well_layout_color** (*str, optional*) – color of well layout, default is gray
- **section_markersize** (*int, optional*) – size of makers in sectionplot
- **ylabel** (*str or list, optional*) – when 'auto' column unit in collection is ylabel, otherwise first element of list is label of section plot, second element of observation plot
- **fn_save** (*str, optional*) – filename to save plot
- **check_obs_close_to_screen_bottom** (*bool, optional*) – plots a horizontal line when minimum observation is close to screen_bottom
- **plot_well_layout_markers** (*bool, optional*) – plots ground level, top tube, screen levels and sandtrap via makers. Default is True
- **plot_obs** (*bool, optional*) – Plots observation. Default is True
- **TODO** –
  - speficy colors via extra column in ObsCollection
  - **addtional visual checks:**
    maximum observation is close to or above ground level, maximum observation is close to or above tube top minimum observation is close to or below tube bottom (sand trap)
  - include some interactive Bokeh fancy
  - apply an offset when two wells are at same location
  - limit y-axis of section plot to observations only

- remove the checking (if obs are near bottom) from this function
- moving the legend outside the plot
- set xlim of observation plot more tight when tmin is not specified

series_per_group(*plot_column*, *by=None*, *savefig=True*, *outputdir='.'*)

Plot time series per group.

The default groupby is based on identical x, y coordinates, so plots unique time series per location.

Parameters

- **plot_column** (*str*) – name of column containing time series data
- **by** (*(list of) str or (list of) array-like*) – groupby parameters, default is None which sets groupby to columns ["x", "y"].
- **savefig** (*bool, optional*) – save figures, by default True
- **outputdir** (*str, optional*) – path to output directory, by default the current directory (".")

class hydropandas.extensions.plots.**ObsPlots**(*obs*)

Bases: object

interactive_plot(*savedir=None*, *cols=(None,)*, *markers=('line',)*, *p=None*, *plot_legend_names=('',)*, *plot_freq=(None,)*, *tmin=None*, *tmax=None*, *hoover_names=('Peil',)*, *hoover_date_format='%Y-%m-%d'*, *ylabel=None*, *plot_colors=('blue',)*, *add_screen_to_legend=False*, *return_filename=False*)

Create an interactive plot of the observations using bokeh.

Todo:

- add options for hoovers, markers, linestyle

Parameters

- **savedir** (*str, optional*) – directory used for the folium map and bokeh plots
- **cols** (*tuple of str or None, optional*) – the columns of the observation to plot. The first numeric column is used if cols is None, by default None.
- **markers** (*list of str, optional*) – type of markers that can be used for plot, 'line' and 'circle' are supported
- **p** (*bokeh.plotting.figure, optional*) – reference to existing figure, if p is None a new figure is created
- **plot_legend_names** (*list of str, optional*) – legend in bokeh plot
- **plot_freq** (*list of str, optional*) – bokeh plot is resampled with this frequency to reduce the size
- **tmin** (*dt.datetime, optional*) – start date for timeseries plot
- **tmax** (*dt.datetime, optional*) – end date for timeseries plot
- **hoover_names** (*list of str, optional*) – names will be displayed together with the cols values when hoovering over plot
- **hoover_date_format** (*str, optional*) – date format to use when hoovering over a plot

- **ylabel** (*str or None, optional*) – label on the y-axis. If None the unit attribute of the observation is used.
- **plot_colors** (*list of str, optional*) – plot_colors used for the plots
- **add_screen_to_legend** (*boolean, optional*) – if True the attributes screen_top and screen_bottom are added to the legend name
- **return_filename** (*boolean, optional*) – if True filename will be returned

Returns
fname_plot – filename of the bokeh plot or reference to bokeh plot

Return type
str or bokeh plot

hydropandas.extensions.stats module

class hydropandas.extensions.stats.**StatsAccessor**(*oc_obj*)

Bases: object

consecutive_obs_years(*min_obs=12*, *col=None*)

get the number of consecutive years with more than a minimum of observations.

Parameters

- **min_obs** (*int or str, optional*) – if min_obs is an integer it is the minimum number of observations per year. If min_obs is a string it is the column name of the obs_collection with minimum number of observation per year per observation.
- **col** (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.

Returns
df – dataframe with the observations as column, the years as rows, and the values are the number of consecutive years.

Return type
pd.DataFrame

property dates_first_obs

property dates_last_obs

get_first_last_obs_date()

get the date of the first and the last measurement.

Returns

- *DataFrame with 2 columns with the dates of the first and the last*
- *measurement*

get_max(*tmin=None*, *tmax=None*, *col=None*)

get the maximum value of every obs object.

Parameters

- **tmin** (*dt.datetime, optional*) – get the maximum value after this date. If None all observations are used.

- **tmax** (*dt.datetime, optional*) – get the maximum value before this date. If None all observations are used.
- **col** (*str or None, optional*) – the column of the obs dataframe to get maximum from. The first numeric column is used if col is None, by default None.

Returns

- *pandas series with the maximum of each observation in the obs*
- *collection.*

get_min(*tmin=None*, *tmax=None*, *col=None*)

get the minimum value of every obs object.

Parameters

- **tmin** (*dt.datetime, optional*) – get the minimum value after this date. If None all observations are used.
- **tmax** (*dt.datetime, optional*) – get the minimum value before this date. If None all observations are used.
- **col** (*str or None, optional*) – the column of the obs dataframe to get minimum from. The first numeric column is used if col is None, by default None.

Returns

- *pandas series with the minimum of each observation in the obs*
- *collection.*

get_no_of_observations(*tmin=None*, *tmax=None*, *col=None*)

get number of non-nan values of a column in the observation df.

Parameters

- **tmin** (*dt.datetime, optional*) – get the number of observations after this date. If None all observations are used.
- **tmax** (*dt.datetime, optional*) – get the number of observations before this date. If None all observations are used.
- **col** (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.

Returns

- *pandas series with the number of observations for each row in the obs*
- *collection.*

get_seasonal_stat(*col=None*, *stat='mean'*, *winter_months=(1, 2, 3, 4, 11, 12)*, *summer_months=(5, 6, 7, 8, 9, 10)*)

get statistics per season.

Parameters

- **col** (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.
- **stat** (*str, optional*) – type of statistics, all statisics from df.describe() are available
- **winter_months** (*tuple of int, optional*) – month number of winter months

- **summer_months** (*tuple of int, optional*) – month number of summer months

Return type
DataFrame with stats for summer and winter

mean_in_period(*tmin=None*, *tmax=None*, *col=None*)

get the mean value of one column (col) in all observations within a period defined by tmin and tmax. If both tmin and tmax are None the whole period in which there are observations is used.

Parameters

- **tmin** (*datetime, optional*) – start of averaging period. The default is None.
- **tmax** (*datetime, optional*) – end of averaging period. The default is None.
- **col** (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.

Returns
mean values for each observation.

Return type
pd.Series

property n_observations

obs_per_year(*col=None*)

property obs_periods

class hydropandas.extensions.stats.**StatsAccessorObs**(*obs*)

Bases: object

consecutive_obs_years(*col=None*, *min_obs=12*)

get_seasonal_stat(*col=None*, *stat='mean'*, *winter_months=(1, 2, 3, 4, 11, 12)*, *summer_months=(5, 6, 7, 8, 9, 10)*)

get statistics per season.

Parameters

- **col** (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.
- **stat** (*str, optional*) – type of statistics, all statisics from df.describe() are available
- **winter_months** (*tuple of int, optional*) – month number of winter months
- **summer_months** (*tuple of int, optional*) – month number of summer months

Returns
two lists with the statistics for the summer and the winter.

Return type
winter_stats, summer_stats

obs_per_year(*col=None*)

get number of observations per year

Parameters
col (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.

Returns
DESCRIPTION.

Return type
TYPE

hydropandas.extensions.stats.**consecutive_obs_years**(*obs_per_year*, *min_obs=12*)

hydropandas.io package

hydropandas.io.arctic module

hydropandas.io.dino module

hydropandas.io.dino.**read_artdino_dir**(*dirname*, *ObsClass=None*, *subdir='csv'*, *suffix='1.csv'*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, *keep_all_obs=True*, ***kwargs*)

Read Dino directory with point observations.

TODO: - Evt. nog verbeteren door meteen Dataframe te vullen op het moment dat een observatie wordt ingelezen. Nu wordt eerst alles ingelezen in een lijst en daar een dataframe van gemaakt.

Parameters

- **dirname** (*str*) – directory name, can be a .zip file or the parent directory of subdir
- **ObsClass** (*type*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **subdir** (*str*) – subdirectory of dirname with data files
- **suffix** (*str*) – suffix of files in subdir that will be read
- **unpackdir** (*str*) – destination directory of the unzipped file
- **force_unpack** (*boolean, optional*) – force unpack if dst already exists
- **preserve_datetime** (*boolean, optional*) – use date of the zipfile for the destination file
- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even without data or metadata
- ****kwargs** (*dict, optional*) – Extra arguments are passed to ObsClass.from_artdino_file()

Returns
obs_df – collection of multiple point observations

Return type
pd.DataFrame

hydropandas.io.dino.**read_artdino_groundwater_csv**(*path*, *to_mnap=True*, *read_series=True*)

Read dino groundwater quantity data from a CSV file as exported by ArtDiver.

Parameters

- **path** (*str*) – path to csv file
- **to_mnap** (*boolean, optional*) – if True a column with 'stand_m_tov_nap' is added to the dataframe
- **read_series** (*boolean, optional*) – if False only metadata is read, default is True

Returns

- **measurements** (*pd.DataFrame*)
- **meta** (*dict*) – dictionary with metadata

hydropandas.io.dino.**read_dino_dir**(*path: Union[str, Path]*, *ObsClass*, *subdir: str = 'Grondwaterstanden_Put'*, *suffix: str = '1.csv'*, *keep_all_obs: bool = True*, ***kwargs: dict*)

Read Dino directory with point observations.

TODO: - Evt. nog verbeteren door meteen Dataframe te vullen op het moment dat een observatie wordt ingelezen. Nu wordt eerst alles ingelezen in een lijst en daar een dataframe van gemaakt. - aparte unzip functie maken en toch de juiste tijdelijke directory krijgen.

Parameters

- **path** (*str | Path*) – directory name, can be a .zip file or the parent directory of subdir
- **ObsClass** (*type*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **subdir** (*str*) – subdirectory of dirname with data files
- **suffix** (*str*) – suffix of files in subdir that will be read
- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even without data or metadata
- ****kwargs** (*dict, optional*) – Extra arguments are passed to ObsClass.from_dino_file()

Returns
obs_df – collection of multiple point observations

Return type
pd.DataFrame

hydropandas.io.dino.**read_dino_groundwater_csv**(*f: Union[str, Path, FileIO]*, *to_mnap: bool = True*, *read_series: bool = True*, *remove_duplicates: bool = False*, *keep_dup: str = 'last'*)

Read dino groundwater quantity data from a dinoloket csv file.

Parameters

- **f** (*Union[str, Path, TextIOWrapper]*) – path to csv file
- **to_mnap** (*boolean, optional*) – if True a column with 'stand_m_tov_nap' is added to the dataframe
- **read_series** (*boolean, optional*) – if False only metadata is read, default is True
- **remove_duplicates** (*boolean, optional*) – if True duplicate indices are removed. Default is False.
- **keep_dup** (*str, optional*) – indicate which duplicate indices should be kept, only used when remove_duplicates is True. Default is 'last'

Returns

- **measurements** (*pd.DataFrame*)
- **meta** (*dict*) – dictionary with metadata

hydropandas.io.dino.**read_dino_groundwater_quality_txt**(*f: Union[str, Path, FileIO]*)

Read dino groundwater quality (grondwatersamenstelling) from a dinoloket txt file.

Notes

this function has not been tested thoroughly

Parameters
filepath_or_buffer (*str*) – path to txt file

Returns

- **measurements** (*pd.DataFrame*)
- **meta** (*dict*) – dictionary with metadata

hydropandas.io.dino.**read_dino_waterlvl_csv**(*f: Union[str, Path, FileIO]*, *to_mnap: bool = True*, *read_series: bool = True*)

Read dino waterlevel data from a dinoloket csv file.

Parameters

- **f** (*str, Path, FileIO*) – path to dino water level csv file
- **to_mnap** (*boolean, optional*) – if True a column with 'stand_m_tov_nap' is added to the dataframe
- **read_series** (*boolean, optional*) – if False only metadata is read, default is True

hydropandas.io.fieldlogger module

hydropandas.io.knmi module

Module with functions to read or download time series with observations from knmi.

The main function to download time series are:

- get_knmi_timeseries_xy: obtain a knmi station based on the xy location
- get_knmi_timeseries_stn: obtain a knmi station based on the station number

The main function to read time series txt files is:

- read_knmi_timeseries_file: read a knmi txt file

hydropandas.io.knmi.**download_knmi_data**(*stn*, *meteo_var*, *start*, *end*, *settings*, *stn_name=None*)

download knmi data of a measurements station for certain observation type.

Parameters

- **stn** (*int*) – measurement station.
- **meteo_var** (*str*) – observation type.
- **start** (*pd.TimeStamp or None*) – start date of observations.
- **end** (*pd.TimeStamp or None*) – end date of observations.
- **settings** (*dict*) – settings for obtaining data
- **stn_name** (*str, optional*) – station name. If None the name is obtained form te stn number. The default is None.

Raises
ValueError – if the data from knmi cannot not be read a ValueError is raised. Unless raise_exceptions is False

Returns

- **knmi_df** (*pandas DataFrame*) – data from one station from one type of observation
- **variables** (*dictionary*) – information about the observerd variables
- **stations** (*pandas DataFrame*) – information about the measurement station.

hydropandas.io.knmi.**fill_missing_measurements**(*stn*, *meteo_var*, *start*, *end*, *settings*, *stn_name=None*)

fill missing measurements in knmi data.

Parameters

- **stn** (*int*) – measurement station.
- **meteo_var** (*str*) – observation type.
- **start** (*pd.TimeStamp*) – start date of observations.
- **end** (*pd.TimeStamp*) – end date of observations.
- **settings** (*dict*) – settings for obtaining data.
- **stn_name** (*str, optional*) – station name. If None the name is obtained form te stn number. The default is None.

Returns

- **knmi_df** (*pandas DataFrame*) – data from one station from one type of observation, with additional column to see which station is used to fill the value
- **variables** (*dictionary*) – information about the observerd variables
- **stations** (*pandas DataFrame*) – information about the measurement station.

hydropandas.io.knmi.**get_evaporation**(*meteo_var*, *stn=260*, *start=None*, *end=None*, *settings=None*)

Collect different types of (reference) evaporation from KNMI weather stations

Parameters

- **meteo_var** (*str*) – Choice between 'penman', 'makkink' or 'hargraves'.
- **stn** (*str*) – station number, defaults to 260 De Bilt
- **start** (*pd.TimeStamp*) – start time of observations.
- **end** (*pd.TimeStamp*) – end time of observations.
- **settings** (*dict or None, optional*) – settings for the time series

Return type
pandas.DataFrame

hydropandas.io.knmi.**get_knmi_daily_meteo_api**(*stn*, *meteo_var*, *start=None*, *end=None*)

download and read knmi daily meteo data.

Parameters

- **stn** (*int*) – station number.
- **meteo_var** (*str*) – e.g. 'EV24'.
- **start** (*pd.TimeStamp or None*) – start time of observations.
- **end** (*pd.TimeStamp or None*) – end time of observations.

Returns

- *pandas DataFrame* – measurements.

- **variables** (*dictionary*) – additional information about the variables
- **stations** (*pandas DataFrame*) – additional data about the measurement station

hydropandas.io.knmi.**get_knmi_daily_meteo_url**(*stn*, *meteo_var*, *start=None*, *end=None*)

download and read knmi daily meteo data.

Parameters

- **stn** (*str*) – station number.
- **meteo_var** (*str*) – e.g. 'EV24'.
- **start** (*pd.TimeStamp or None*) – start time of observations.
- **end** (*pd.TimeStamp or None*) – end time of observations.

Returns

- *pandas DataFrame* – measurements.
- **variables** (*dictionary*) – additional information about the variables
- **stations** (*pandas DataFrame*) – additional data about the measurement station

hydropandas.io.knmi.**get_knmi_daily_rainfall_api**(*stn*, *start=None*, *end=None*)

download and read knmi daily rainfall.

Parameters

- **stn** (*int*) – station number.
- **start** (*pd.TimeStamp or None*) – start time of observations.
- **end** (*pd.TimeStamp or None*) – end time of observations.

Raises

ValueError – if there is no data for the provided stn an error is raised.

Returns

- *pandas DataFrame* – measurements.
- **variables** (*dictionary*) – additional information about the variables

hydropandas.io.knmi.**get_knmi_daily_rainfall_url**(*stn*, *stn_name*, *start=None*, *end=None*)

download and read knmi daily rainfall.

Parameters

- **stn** (*int*) – station number.
- **stn_name** (*str*) – the name of the station in capital letters, can be tricky
- **start** (*pd.TimeStamp or None*) – start time of observations.
- **end** (*pd.TimeStamp or None*) – end time of observations.

Raises

ValueError – if there is no data for the provided stn an error is raised.

Returns

- *pandas DataFrame* – measurements.
- **variables** (*dictionary*) – additional information about the variables

hydropandas.io.knmi.**get_knmi_hourly_api**(*stn*, *meteo_var*, *start*, *end*)

Retrieve hourly meteorological data from the KNMI API.

Parameters

- **stn** (*int*) – The station number.
- **meteo_var** (*str*) – The meteorological variable to retrieve.
- **start** (*pd.Timestamp or None*) – The start date and time of the data.
- **end** (*pd.Timestamp or None*) – The end date and time of the data.

Returns

- *pandas.DataFrame* – A DataFrame containing the requested meteorological variable data.
- *dict* – A dictionary containing information about the variables in the DataFrame.

Raises

- **requests.ConnectionError** – If there is a connection error while accessing the KNMI API.
- **RuntimeError** – If the KNMI API is down or returns unexpected data.

hydropandas.io.knmi.**get_knmi_obs**(*stn=None*, *fname=None*, *xy=None*, *meteo_var=None*, *start=None*, *end=None*, ***kwargs*)

get knmi observation from stn, fname or nearest xy coordinates.

Parameters

- **stn** (*int, str or None, optional*) – measurement station e.g. 829. The default is None.
- **fname** (*str, path object, file-like object or None, optional*) – filename of a knmi file. The default is None.
- **xy** (*list, tuple or None, optional*) – RD coördinates of a location in the Netherlands. The station nearest to this location used. The Default is None.
- **meteo_var** (*str or None, optional*) – meteo variable e.g. "RH" or "EV24". See list with all options in the hydropandas documentation.
- **start** (*str, datetime or None, optional*) – start date of observations. The default is None.
- **end** (*str, datetime or None, optional*) – end date of observations. The default is None.
- ****kwargs** –

  fill_missing_obs
  [bool, optional] if True nan values in time series are filled with nearby time series. The default is False.

  interval
  [str, optional] desired time interval for observations. Options are 'daily' and 'hourly'. The default is 'daily'.

  use_api
  [bool, optional]

if True the api is used to obtain the data, API documentation is here:
https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script

if False a text file is downloaded into a temporary folder and the data is read from there. Default is True since the api is back online (July 2021).

raise_exceptions
[bool, optional] if True you get errors when no data is returned. The default is False.

Returns

- *pd.DataFrame, measurements*
- *dict, metadata*

Raises
ValueError – if no meteo_var is given or stn, fname and xy are all None.

hydropandas.io.knmi.**get_knmi_obslist**(*locations=None*, *stns=None*, *xy=None*, *meteo_vars=('RH',)*, *starts=None*, *ends=None*, *ObsClasses=None*, ***kwargs*)

Get a list of observations of knmi stations. Either specify a list of knmi stations (stns) or a dataframe with x, y coordinates (locations).

Parameters

- **locations** (*pandas DataFrame or None*) – dataframe with x and y coordinates. The default is None
- **stns** (*list of str or None*) – list of knmi stations. The default is None
- **xy** (*list or numpy array, optional*) – xy coordinates of the locations. e.g. [[10,25], [5,25]]
- **meteo_vars** (*list or tuple of str*) – meteo variables e.g. ["RH", "EV24"]. The default is ("RH")
- **starts** (*None, str, datetime or list, optional*) – start date of observations per meteo variable. The start date is included in the time series. If start is None the start date will be January 1st of the previous year. if start is str it will be converted to datetime if start is a list it should be the same length as meteo_vars and the start time for each variable. The default is None
- **ends** (*list of str, datetime or None*) – end date of observations per meteo variable. The end date is included in the time series. If end is None the start date will be January 1st of the previous year. if end is a str it will be converted to datetime if end is a list it should be the same length as meteo_vars and the end time for each meteo variable. The default is None
- **ObsClasses** (*list of type or None*) – class of the observations, can be PrecipitationObs or EvaporationObs. The default is None.
- ****kwargs** –

  fill_missing_obs
  [bool, optional] if True nan values in time series are filled with nearby time series. The default is False.

  interval
  [str, optional] desired time interval for observations. Options are 'daily' and 'hourly'. The default is 'daily'.

use_api
[bool, optional]

if True the api is used to obtain the data, API documentation is here:
https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script

if False a text file is downloaded into a temporary folder and the data is read from there. Default is True since the api is back online (July 2021).

raise_exceptions
[bool, optional] if True you get errors when no data is returned. The default is False.

Returns
obs_list – collection of multiple point observations

Return type
list of obsevation objects

hydropandas.io.knmi.**get_knmi_timeseries_fname**(*fname*, *meteo_var*, *settings*, *start*, *end*)

hydropandas.io.knmi.**get_knmi_timeseries_stn**(*stn*, *meteo_var*, *settings*, *start=None*, *end=None*)

Get a knmi time series and metadata.

Parameters

- **stn** (*int*) – measurement station e.g. 829.
- **meteo_var** (*str*) – observation type e.g. "RH" or "EV24". See list with all options in the hpd.read_knmi function.
- **settings** (*dict*) – settings for obtaining the right time series, see _get_default_settings for more information
- **start** (*pd.TimeStamp or None, optional*) – start date of observations. The default is None.
- **end** (*pd.TimeStamp or None, optional*) – end date of observations. The default is None.

Returns

- **knmi_df** (*pandas DataFrame*) – time series with measurements.
- **meta** (*dictionary*) – metadata from the measurement station.

hydropandas.io.knmi.**get_n_nearest_stations_xy**(*xy*, *meteo_var*, *n=1*, *stations=None*, *ignore=None*)

Find the N nearest KNMI stations that measure variable 'meteo_var' to the x, y coordinates.

Parameters

- **xy** (*list, tuple or numpy.array of int or float*) – sinlge pair of xy coordinates. e.g. (150_000., 400_000.)
- **meteo_var** (*str*) – measurement variable e.g. 'RH' or 'EV24'
- **n** (*int, optional*) – number of stations you want to return. The default is 1.
- **stations** (*pandas DataFrame, optional*) – if None stations will be obtained using the get_stations function. The default is None.
- **ignore** (*list, optional*) – list of stations to ignore. The default is None.

Returns
station numbers.

Return type
list

hydropandas.io.knmi.**get_nearest_station_df**(*locations*, *xcol='x'*, *ycol='y'*, *stations=None*, *meteo_var='RH'*, *ignore=None*)

Find the KNMI stations that measure 'meteo_var' closest to the coordinates in 'locations'.

Parameters

- **locations** (*pandas.DataFrame*) – DataFrame containing x and y coordinates
- **xcol** (*str*) – name of the column in the locations dataframe with the x values
- **ycol** (*str*) – name of the column in the locations dataframe with the y values
- **stations** (*pandas DataFrame, optional*) – if None stations will be obtained using the get_stations function. The default is None.
- **meteo_var** (*str*) – measurement variable e.g. 'RH' or 'EV24'
- **ignore** (*list, optional*) – list of stations to ignore. The default is None.

Returns
stns – station numbers.

Return type
list

hydropandas.io.knmi.**get_nearest_station_xy**(*xy*, *stations=None*, *meteo_var='RH'*, *ignore=None*)

find the KNMI stations that measure 'meteo_var' closest to the given x and y coordinates.

Parameters

- **xy** (*list or numpy array, optional*) – xy coordinates of the locations. e.g. [[10,25], [5,25]]
- **stations** (*pandas DataFrame, optional*) – if None stations will be obtained using the get_stations function. The default is None.
- **meteo_var** (*str*) – measurement variable e.g. 'RH' or 'EV24'
- **ignore** (*list, optional*) – list of stations to ignore. The default is None.

Returns
stns – station numbers.

Return type
list

Notes

assumes you have a structured rectangular grid.

hydropandas.io.knmi.**get_station_name**(*stn*, *stations=None*)

Returns the station name from a KNMI station.

Modifies the station name in such a way that a valid url can be obtained.

Parameters

- **stn** (*int*) – station number.
- **stations** (*pandas DataFrame*) – DataFrame with the station metadata.

Returns
Name of the station or None if station is not found.

Return type
str or None

hydropandas.io.knmi.**get_stations**(*meteo_var*)

get knmi stations from json files according to variable.

Parameters
meteo_var (*str, optional*) – type of meteodata, by default 'RH'

Return type
pandas DataFrame with stations, names and coordinates (Lat/Lon & RD)

hydropandas.io.knmi.**hargreaves**(*tmean*, *tmin*, *tmax*, *dates*, *lat=52.1*, *x=None*)

Estimate of Hargraves potential evaporation according to Allen et al. 1990.

Parameters

- **tmean** (*pandas.Series*) – Daily mean temperature
- **tmin** (*pandas.Series*) – Daily minimum temperature
- **tmax** (*pandas.Series*) – Daily maximum temperature
- **dates** (*pandas.Series.index*) – Dates
- **lat** (*float, optional*) – Latitude of station, by default 52.1
- **x** (*_type_, optional*) – Optional parameter to scale evaporation estimate, by default None

Return type
pandas.Series

hydropandas.io.knmi.**makkink**(*tmean*, *K*)

Estimate of Makkink reference evaporation according to KNMI.

Parameters

- **tmean** (*tmean : pandas.Series*) – Daily mean temperature in Celsius
- **K** (*pandas.Series*) – Global radiation estimate in J/cm2

Return type
pandas.Series

hydropandas.io.knmi.**penman**(*tmean*, *tmin*, *tmax*, *K*, *wind*, *rh*, *dates*, *z=1.0*, *lat=52.1*, *G=0.0*, *wh=10.0*, *tdew=None*)

Estimate of Penman reference evaporation according to Allen et al 1990.

Parameters

- **tmean** (*pandas.Series*) – Daily mean temperature in Celsius
- **tmin** (*pandas.Series*) – Daily minimum temperature in Celsius
- **tmax** (*pandas.Series*) – Daily maximum temperature in Celsius
- **K** (*pandas.Series*) – Global radiation estimate in J/cm2
- **wind** (*pandas.Series*) – Daily mean wind speed in m/s
- **rh** (*pandas.Series*) – Relative humidity in %

- **dates** (*pandas.Series*) – Dates
- **z** (*float, optional*) – Elevation of station in m, by default 1.0
- **lat** (*float, optional*) – Latitude of station, by default 52.1
- **G** (*float, optional*) – Ground flux in MJ/m2, by default 0.0
- **wh** (*float, optional*) – Height of wind measurement in m, by default 10.0
- **tdew** (*pandas.Series*) – Dew point temperature in C, by default None

Return type
pandas.Series

hydropandas.io.knmi.**read_knmi_daily_meteo**(*f*, *meteo_var*)

hydropandas.io.knmi.**read_knmi_daily_meteo_file**(*path*, *meteo_var*, *start=None*, *end=None*)

read knmi daily meteo data from a file

Parameters

- **path** (*str*) – file path of .txt file.
- **meteo_var** (*str*) – e.g. 'EV24'.
- **start** (*pd.TimeStamp or None*) – start time of observations.
- **end** (*pd.TimeStamp or None*) – end time of observations.

Raises
ValueError – If the meteo var is not in the file.

Returns

- *pandas DataFrame* – measurements.
- **variables** (*dictionary*) – additional information about the variables

hydropandas.io.knmi.**read_knmi_daily_rainfall**(*f*, *meteo_var*)

Read daily rainfall data from a KNMI file.

Parameters

- **f** (*file-like object*) – The file object containing the KNMI data.
- **meteo_var** (*str*) – The meteorological variable to extract.

Returns

- *pandas.DataFrame* – The DataFrame containing the extracted daily rainfall data.
- *dict* – A dictionary containing information about the variables in the DataFrame.

Notes

This function assumes that the file object *f* is already open and positioned at the start of the data. The file is expected to have a header with variable names and a corresponding data table.

The DataFrame returned by this function has the following modifications: - The index is set to the datetime values derived from the 'YYYYMMDD' column. - The 'YYYYMMDD' column is dropped. - Duplicate indices are removed, keeping the first occurrence. - If the last row has missing values, it is removed. - The 'meteo_var' column is cast to float data type. - Variables are transformed using an internal function *_transform_variables*. - The unit of measurement for the 'meteo_var' variable is set to 'm'.

hydropandas.io.knmi.**read_knmi_daily_rainfall_file**(*fname_txt*, *start=None*, *end=None*)

read a knmi file with daily rainfall data.

Parameters

- **path** (*str*) – file path of a knmi .txt file.
- **start** (*pd.TimeStamp or None*) – start time of observations.
- **end** (*pd.TimeStamp or None*) – end time of observations.

Returns

- *pandas DataFrame* – measurements.
- **variables** (*dictionary*) – additional information about the variables

hydropandas.io.knmi.**read_knmi_hourly**(*f*, *meteo_var*, *start=None*, *end=None*)

Read hourly KNMI file.

Parameters

f (*str or filelike object*) – path to file or filelike object

Returns

- **df** (*pd.DataFrame*) – DataFrame containing data
- **variables** (*dict*) – dictionary containing metadata about the variables

hydropandas.io.menyanthes module

hydropandas.io.menyanthes.**matlab2datetime**(*tindex*)

Transform a MATLAB serial date number to a Python datetime object, rounded to seconds.

Parameters

tindex (*float*) – The MATLAB serial date number to convert.

Returns

datetime – The equivalent datetime object in Python.

Return type

datetime.datetime

Notes

MATLAB serial date numbers represent the number of days elapsed since January 1, 0000 (the proleptic Gregorian calendar), with January 1, 0000 as day 1. Fractions of a day can be represented as a decimal.

The returned datetime object is rounded to the nearest second.

Examples

```
>>> matlab2datetime(719529.496527778)
datetime.datetime(2019, 1, 1, 11, 55, 2)
```

hydropandas.io.menyanthes.**read_file**(*path*, *ObsClass*, *load_oseries=True*, *load_stresses=True*)

Read data from a Menyanthes file and create observation objects.

Parameters

- **path** (*str*) – Full path of the Menyanthes file (.men) to read.
- **ObsClass** (GroundwaterObs *or* WaterlvlObs) – Class of observation object to create.
- **load_oseries** (*bool, optional*) – Flag indicating whether to load observation series or not, by default True.
- **load_stresses** (*bool, optional*) – Flag indicating whether to load stresses or not, by default True.

Returns

obs_list – List of observation objects created from the Menyanthes file.

Return type

list

hydropandas.io.menyanthes.**read_oseries**(*mat*)

Read the oseries from a mat file from menyanthes.

Parameters

mat (*dict*) – A dictionary object containing the Menyanthes file data.

Returns

A dictionary containing oseries data, with oseries names as keys and their corresponding metadata and values as values.

Return type

dict

Notes

This function reads the oseries data from a Menyanthes file in .mat format and returns it in a dictionary format. The oseries data contains the following metadata:

- name: The name of the oseries.
- x: The x-coordinate of the oseries location.
- y: The y-coordinate of the oseries location.
- source: The data source.
- unit: The unit of measurement.

In addition to the metadata, the oseries data also contains a pandas Series object named 'values', which contains the time series data for the oseries.

Examples

```
>>> mat = loadmat('menyanthes_file.mat')
>>> d_h = read_oseries(mat)
```

hydropandas.io.menyanthes.**read_stresses**(*mat*)

Reads the stresses from a mat file from menyanthes.

Parameters
mat (*dict*) – A dictionary object containing the mat file.

Returns
A dictionary object containing the stresses data.

Return type
dict

hydropandas.io.modflow module

hydropandas.io.modflow.**interp_weights**(*xy*, *uv*, *d=2*)

Calculate interpolation weights[1].

Parameters

- **xy** (*np.array*) – array containing x-coordinates in first column and y-coordinates in second column
- **uv** (*np.array*) – array containing coordinates at which interpolation weights should be calculated, x-data in first column and y-data in second column
- **d** (*int, optional*) – dimension of data? (the default is 2, which works for 2D data)

Returns

- **vertices** (*np.array*) – array containing interpolation vertices
- **weights** (*np.array*) – array containing interpolation weights per point

References

speedup-scipy-griddata-for-multiple-interpolations-between-two-irregular-grids

hydropandas.io.modflow.**interpolate**(*values*, *vtx*, *wts*, *fill_value=nan*)

Interpolate values at locations defined by vertices and points[2], as calculated by interp_weights function.

Parameters

- **values** (*np.array*) – array containing values to interpolate
- **vtx** (*np.array*) – array containing interpolation vertices, see interp_weights()
- **wts** (*np.array*) – array containing interpolation weights, see interp_weights()
- **fill_value** (*float*) – fill value for points that have to be extrapolated (e.g. at or beyond edges of the known points)

[1] https://stackoverflow.com/questions/20915502/
[2] https://stackoverflow.com/questions/20915502/

Returns
arr – array containing interpolated values at locations as given by vtx and wts

Return type
np.array

References

speedup-scipy-griddata-for-multiple-interpolations-between-two-irregular-grids

hydropandas.io.modflow.**read_imod_results**(*obs_collection*, *ml*, *runfile*, *mtime*, *model_ws*, *modelname=''*, *nlay=None*, *exclude_layers=0*)

Read imod model results at point locations.

Parameters

- **obs_collection** (ObsCollection) – collection of observations at which points imod results will be read
- **ml** (*flopy.modflow.mf.model*) – modflow model
- **runfile** (*Runfile*) – imod runfile object
- **mtime** (*list of datetimes*) – datetimes corresponding to the model periods
- **model_ws** (*str*) – model workspace with imod model
- **nlay** (*int, optional*) – number of layers if None the number of layers from ml is used.
- **modelname** (*str*) – modelname
- **exclude_layers** (*int*) – exclude modellayers from being read from imod

hydropandas.io.modflow.**read_modflow_results**(*obs_collection*, *ml*, *hds_arr*, *mtime*, *modelname=''*, *nlay=None*, *exclude_layers=None*, *method='linear'*)

Read modflow groundwater heads at points in obs_collection.

Parameters

- **obs_collection** (ObsCollection) – locations of model observation
- **ml** (*flopy.modflow.mf.model*) – modflow model
- **hds_arr** (*numpy array*) – heads with shape (ntimesteps, nlayers, nrow, ncol)
- **mtime** (*list of datetimes*) – dates for each model timestep
- **modelname** (*str, optional*) – modelname
- **nlay** (*int, optional*) – number of layers if None the number of layers from ml is used.
- **exclude_layers** (*list of int, optional*) – exclude the observations in these modellayers
- **method** (*str, optional*) – interpolation method, either 'linear' or 'nearest', default is linear.

hydropandas.io.pastas module

Created on Wed Sep 12 12:15:42 2018.

@author: Artesia

hydropandas.io.pastas.**create_pastastore**(*oc*, *pstore*, *pstore_name=''*, *conn=None*, *add_metadata=True*, *col=None*, *kind='oseries'*, *overwrite=False*)

add observations to a new or existing pastastore.

Parameters

- **oc** (*observation.ObsCollection*) – collection of observations
- **pstore** (*pastastore.PastaStore, optional*) – Existing pastastore, if None a new pastastore is created
- **pstore_name** (*str, optional*) – Name of the pastastore only used if pstore is None
- **conn** (*pastastore.connectors*) – connector for database
- **col** (*str or None, optional*) – the column of the obs dataframe to use in pastas. The first numeric column is used if col is None, by default None.
- **kind** (*str, optional*) – The kind of series that is added to the pastastore
- **add_metadata** (*boolean, optional*) – If True metadata from the observations added to the pastastore
- **overwrite** (*boolean, optional*) – if True, overwrite existing series in pastastore, default is False

Returns

pstore – the pastastore with the series from the ObsCollection

Return type

pastastore.PastaStore

hydropandas.io.pastas.**read_pastastore_item**(*pstore*, *libname*, *name*)

Read item from pastastore library.

Parameters

- **pstore** (*pastastore.PastaStore*) – pastastore object
- **libname** (*str*) – name of library containing item
- **name** (*str*) – name of item

Returns

- **series** (*pd.Series*) – time series for item
- **meta** (*dict*) – dictionary containing metadata

Raises

ValueError – if library is not oseries or stresses

hydropandas.io.pastas.**read_pastastore_library**(*pstore*, *libname*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *metadata_mapping=None*)

Read pastastore library.

Parameters

- **pstore** (*pastastore.PastaStore*) – pastastore object
- **libname** (*str*) – name of library to read
- **ObsClass** (*Obs, optional*) – type of Obs to read data as, by default GroundwaterObs
- **metadata_mapping** (*dict, optional*) – dictionary containing map between metadata field names in pastastore and metadata field names expected by hydropandas, by default None.

Returns
obs_list – list of Obs containing data

Return type
list of *Obs*

hydropandas.io.pystore module

hydropandas.io.waterinfo module

hydropandas.io.waterinfo.**read_waterinfo_file**(*path*, *index_cols=None*, *return_metadata=False*, *value_col=None*, *location_col=None*, *xcol=None*, *ycol=None*, *transform_coords=True*)

Read waterinfo file (CSV or zip)

Parameters
path (*str*) – path to waterinfo file (.zip or .csv)

Returns

- **df** (*pandas.DataFrame*) – DataFrame containing file content
- **metadata** (*dict, optional*) – dict containing metadata, returned if return_metadata is True, default is False

hydropandas.io.waterinfo.**read_waterinfo_obs**(*file_or_dir*, *ObsClass*, *progressbar=False*, ***kwargs*)

Read waterinfo file or directory and extract locations and observations.

Parameters

- **file_or_dir** (*str*) – path to file or directory
- **ObsClass** (*Obs type*) – type of Obs to store data in
- **progressbar** (*bool, optional*) – show progressbar if True, default is False

Returns
obs_collection – list of Obs objects

Return type
list

hydropandas.io.wiski module

hydropandas.io.wiski.**read_wiski_dir**(*dirname*, *ObsClass=None*, *suffix='.csv'*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, *keep_all_obs=True*, ***kwargs*)

Reads WISKI CSV files from a directory and returns a list of observation objects.

Parameters

- **dirname** (*str*) – The path of the directory containing the WISKI CSV files.
- **ObsClass** (*object, optional*) – The observation class to use for creating observation objects. Default is None.
- **suffix** (*str, optional*) – The file extension of the WISKI CSV files. Default is ".csv".
- **unpackdir** (*str, optional*) – The directory to which the files should be unpacked. Default is None.
- **force_unpack** (*bool, optional*) – If True, forces the files to be unpacked even if they are already in the target directory. Default is False.
- **preserve_datetime** (*bool, optional*) – If True, preserves the original modification times of the files when unpacking them. Default is False.
- **keep_all_obs** (*bool, optional*) – If True, keeps all observation objects even if they have no metadata available. Default is True.
- ****kwargs** – Additional keyword arguments to pass to the *from_wiski* method of the *ObsClass* object.

Returns
A list of observation objects created from the WISKI CSV files in the directory.

Return type
list

Raises
FileNotFoundError – If no WISKI CSV files are found in the directory.

hydropandas.io.wiski.**read_wiski_file**(*path*, *sep=';'*, *header_sep=None*, *header_identifier='#'*, *read_series=True*, *translate_dic=None*, *tz_localize=True*, *unit=''*, ***kwargs*)

Read data from a WISKI file.

Parameters:

path
[str] The path of the file to be read.

sep
[str, optional (default=";")] The delimiter used to separate fields in the file.

header_sep
[str, optional (default=None)] The delimiter used to separate fields in the header. If None, the function will try to automatically detect the separator.

header_identifier
[str, optional (default="#")] The character used to identify header lines.

read_series
[bool, optional (default=True)] Whether to read the time series data from the file.

translate_dic
[dict, optional (default=None)] A dictionary mapping header field names to the desired output names.

tz_localize
[bool, optional (default=True)] Whether to localize the datetime index to the machine's timezone.

unit
[str, optional (default="")] The unit of measurement of the data.

****kwargs**
[keyword arguments] Additional arguments to pass to the pandas *read_csv* function.

Returns:

data
[pandas.DataFrame or None] A dataframe containing the time series data from the file. Returns None if *read_series* is False.

metadata
[dict] A dictionary containing metadata about the data in the file.

hydropandas.io.fews module

hydropandas.io.fews.**get_fews_pid**(*name: str*) → Dict[str, *Obs*]

Get matching ParameterId's and HydroPandas Observation Classes

Parameters
name (*str*) – Waterboard name

Returns
Dictonary with ParameterId and the resulting Observation Class

Return type
Dict[str, *Obs*]

hydropandas.io.fews.**iterparse_pi_xml**(*fname: str*, *ObsClass: Union[Obs, Dict[str, Obs]]*, *translate_dic: Optional[Dict[str, str]] = None*, *filterdict: Optional[Dict[str, List[str]]] = None*, *locationIds: Optional[List[str]] = None*, *return_events: bool = True*, *keep_flags: Tuple[int] = (0, 1)*, *return_df: bool = False*, *tags: Tuple[str] = ('series', 'header', 'event')*)

Read a FEWS XML-file with measurements, memory efficient.

Parameters

- **fname** (*str*) – full path to file
- **ObsClass** (*Union[Obs, Dict[str, Obs]],*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'monitoring_well'}.
- **locationIds** (*tuple or list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read.

- **filterdict** (*dict, optional*) – dictionary with tag name to apply filter to as keys, and list of accepted names as dictionary values to keep in final result, i.e. {"locationId": ["B001", "B002"]}
- **return_events** (*bool, optional*) – return all event-information in a DataFrame per location, instead of just a Series (defaults to False). Overrules keep_flags kwarg.
- **keep_flags** (*list of ints, optional*) – keep the values with these flags (defaults to 0 and 1). Only used when return_events is False.
- **tags** (*list of strings, optional*) – Select the tags to be parsed. Defaults to series, header and event
- **return_df** (*bool, optional*) – return a DataFame with the data, instead of two lists (default is False)

Returns

- **df** (*pandas.DataFrame*) – a DataFrame containing the metadata and the series if 'return_df' is True
- **obs_list** (*list of pandas Series*) – list of timeseries if 'return_df' is False

hydropandas.io.fews.**read_xml_filelist**(*fnames: List[str]*, *ObsClass: Union[Obs, Dict[str, Obs]]*, *directory: Optional[str] = None*, *locations: Optional[List[str]] = None*, *translate_dic: Optional[Dict[str, str]] = None*, *filterdict: Optional[Dict[str, List[str]]] = None*, *remove_nan: bool = False*, *low_memory: bool = True*, ***kwargs: dict*)

Read a list of xml files into a list of observation objects.

Parameters

- **fnames** (*TYPE*) – DESCRIPTION.
- **ObsClass** (*Union[Obs, Dict[str, Obs]]*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **directory** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **locations** (*tuple or list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read.
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'monitoring_well'}.
- **filterdict** (*dict, optional*) – dictionary with tag name to apply filter to as keys, and list of accepted names as dictionary values to keep in final result, i.e. {"locationId": ["B001", "B002"]}
- **remove_nan** (*boolean, optional*) – remove nan values from measurements, flag information about the nan values is also lost, only used if low_memory=False
- **low_memory** (*bool, optional*) – whether to use xml-parsing method with lower memory footprint, default is True

Returns

list of timeseries stored in ObsClass objects

Return type

list of ObsClass objects

hydropandas.io.fews.**read_xml_fname**(*fname: str*, *ObsClass: Union[Obs, Dict[str, Obs]]*, *translate_dic: Optional[Dict[str, str]] = None*, *low_memory: bool = True*, *locationIds: Optional[List[str]] = None*, *filterdict: Optional[Dict[str, List[str]]] = None*, *return_events: bool = True*, *keep_flags: Tuple[int] = (0, 1)*, *return_df: bool = False*, *tags: Tuple[str] = ('series', 'header', 'event')*, *remove_nan: bool = False*, ***kwargs: dict*)

Read an xml filename into a list of observations objects.

Parameters

- **fname** (*str*) – full path to file
- **ObsClass** (*Union[Obs, Dict[str, Obs]]*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'monitoring_well'}.
- **low_memory** (*bool, optional*) – whether to use xml-parsing method with lower memory footprint, default is True
- **locationIds** (*tuple or list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read.
- **filterdict** (*dict, optional*) – dictionary with tag name to apply filter to as keys, and list of accepted names as dictionary values to keep in final result, i.e. {"locationId": ["B001", "B002"]}
- **return_events** (*bool, optional*) – return all event-information in a DataFrame per location, instead of just a Series (defaults to False). Overrules keep_flags kwarg.
- **keep_flags** (*list of ints, optional*) – keep the values with these flags (defaults to 0 and 1). Only used when return_events is False.
- **tags** (*list of strings, optional*) – Select the tags to be parsed. Defaults to series, header and event
- **return_df** (*bool, optional*) – return a DataFame with the data, instead of two lists (default is False)
- **remove_nan** (*boolean, optional*) – remove nan values from measurements, flag information about the nan values is also lost, only used if low_memory=False

Returns
list of timeseries stored in ObsClass objects

Return type
list of ObsClass objects

hydropandas.io.fews.**read_xml_root**(*root: Element*, *ObsClass: Union[Obs, Dict[str, Obs]]*, *translate_dic: Dict[str, str] = None*, *locationIds: List[str] = None*, *remove_nan: bool = False*)

Read a FEWS XML-file with measurements, return list of ObsClass objects.

Parameters

- **root** (*xml.etree.ElementTree.Element*) – root element of a fews xml
- **ObsClass** (*Union[Obs, Dict[str, Obs]],*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'monitoring_well'}.

- **locationIds** (*tuple or list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read.
- **remove_nan** (*boolean, optional*) – remove nan values from measurements, flag information about the nan values is also lost

Returns
list of timeseries stored in ObsClass objects

Return type
list of ObsClass objects

hydropandas.io.fews.**read_xmlstring**(*xmlstring: str*, *ObsClass: Union[Obs, Dict[str, Obs]]*, *translate_dic: Optional[Dict[str, str]] = None*, *filterdict: Optional[Dict[str, List[str]]] = None*, *locationIds: Optional[List[str]] = None*, *low_memory: bool = True*, *remove_nan: bool = False*)

Read xmlstring into an list of Obs objects. Xmlstrings are usually obtained using a fews api.

Parameters

- **xmlstring** (*str*) – xml string to be parsed. Typically from a fews api.
- **ObsClass** (*Union[Obs, Dict[str, Obs]]*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'monitoring_well'}.
- **locationIds** (*tuple or list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read.
- **low_memory** (*bool, optional*) – whether to use xml-parsing method with lower memory footprint, default is True
- **remove_nan** (*boolean, optional*) – remove nan values from measurements, flag information about the nan values is also lost, only used if low_memory=False

Returns
list of timeseries stored in ObsClass objects

Return type
list of ObsClass objects

hydropandas.io.fews.**write_pi_xml**(*obs_coll*, *fname: str*, *timezone: float = 1.0*, *version: str = '1.24'*)

Write TimeSeries object to PI-XML file.

Parameters
fname (*path*) – path to XML file

1.4.2 hydropandas.obs_collection module

Module with ObsCollection class for a collection of observations.

The ObsCollection class is a subclass of a pandas DataFrame with additional attributes and methods.

More information about subclassing pandas DataFrames can be found here: http://pandas.pydata.org/pandas-docs/stable/development/extending.html#extending-subclassing-pandas

class hydropandas.obs_collection.**ObsCollection**(**args*, ***kwargs*)

Bases: DataFrame

Class for a collection of point observations.

An ObsCollection object is a subclass of a pandas.DataFrame and allows for additional attributes and methods. Additional attributes are defined in the '_metadata' attribute.

Parameters

- **observations** (**args*) –
- **DataFrame** (*list of observations or a pandas*) –

:param : :param **kwargs can be one of these:

name
[str] name of the observation collection

meta
[dic] metadata of the observation collection

add_meta_to_df(*key='all'*)

Get the values from the meta dictionary of each observation object and add these to the ObsCollection as a column.

to the ObsCollection

Parameters
key (*str, int, tuple, list, set or None, optional*) – key in meta dictionary of observation object. If key is 'all', all keys are added. The default is 'all'.

add_obs_collection(*obs_collection*, *check_consistency=True*, *inplace=False*, ***kwargs*)

Add one observation collection to another observation collection. See add_observation method for more details.

Parameters

- **obs_collection** (*hpd.ObsCollection*) – ObsCollection object.
- **check_consistency** (*bool, optional*) – If True the consistency of both collections is first checked. The default is True.
- **inplace** (*bool, optional*) – If True, modifies the ObsCollection in place (do not create a new object). The default is False.
- **Obs.merge_observation** (***kwargs passed to*) –

  merge_metadata
  [bool, optional] If True and observations are merged the metadata of the two objects are merged. If there are any differences the overlap parameter is used to determine which metadata is used. If merge_metadata is False, the metadata of the original observation is always used for the merged observation. The default is True.

  overlap
  [str, optional] How to deal with overlapping timeseries with different values. Options are: - error : Raise a ValueError - use_left : use the overlapping part from the existing observations - use_right : use the overlapping part from the new observation Default is 'error'.

Raises
RuntimeError – when the observation collection is inconsistent.

Returns
merged ObsCollection if `inplace=True`.

Return type
ObsCollection or None

`add_observation`(*o*, *check_consistency=True*, ***kwargs*)

Add an observation to an existing observation collection. If the observation exists the two observations are merged.

Parameters

- **o** (*hpd.observation.Obs*) – Observation object.
- **check_consistency** (*bool, optional*) – If True the consistency of the collection is first checked. The default is True.
- **Obs.merge_observation** (***kwargs passed to*) –

  merge_metadata
  [bool, optional] If True and observations are merged the metadata of the two objects are merged. If there are any differences the overlap parameter is used to determine which metadata is used. If merge_metadata is False, the metadata of the original observation is always used for the merged observation. The default is True.

  overlap
  [str, optional] How to deal with overlapping timeseries with different values. Options are: - error : Raise a ValueError - use_left : use the overlapping part from the existing observations - use_right : use the overlapping part from the new observation Default is 'error'.

Raises

- **RuntimeError** – when the observation collection is inconsistent.
- **TypeError** – when the observation type is wrong.

Return type
None.

`copy`(*deep=False*)

Make a copy of this object's indices and data.

Parameters
deep (*bool, default True*) – Make a deep copy, including a deep copy of the observation objects. With `deep=False` neither the indices nor the data are copied.

Return type
ObsCollection

`classmethod from_artdino_dir`(*dirname=None*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *subdir='csv'*, *suffix='.csv'*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, *keep_all_obs=True*, *name=None*, ***kwargs*)

Read a dino directory.

Parameters

- **extent** (*list, optional*) – get dinodata online within this extent [xmin, xmax, ymin, ymax]
- **dirname** (*str, optional*) – directory name, can be a .zip file or the parent directory of subdir

- **ObsClass** (*type*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **subdir** (*str*) – subdirectory of dirname with data files
- **suffix** (*str*) – suffix of files in subdir that will be read
- **unpackdir** (*str*) – destination directory of the unzipped file
- **force_unpack** (*boolean, optional*) – force unpack if dst already exists
- **preserve_datetime** (*boolean, optional*) – use date of the zipfile for the destination file
- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even without data or metadata
- **name** (*str, optional*) – the name of the observation collection
- **kwargs** – kwargs are passed to the hydropandas.io.dino.read_dino_dir() function

Returns
cls(obs_df) – collection of multiple point observations

Return type
ObsCollection

classmethod from_bro(*extent=None*, *bro_id=None*, *name=''*, *tmin=None*, *tmax=None*, *only_metadata=False*, *keep_all_obs=True*, *epsg=28992*, *ignore_max_obs=False*)

Get all the observations within an extent or within a groundwatermonitoring net.

Parameters

- **extent** (*list, tuple, numpy-array or None, optional*) – get groundwater monitoring wells within this extent [xmin, xmax, ymin, ymax]
- **bro_id** (*str or None, optional*) – starts with 'GMN'.
- **name** (*str, optional*) – name of the observation collection
- **tmin** (*str or None, optional*) – start time of observations. The default is None.
- **tmax** (*str or None, optional*) – end time of observations. The default is None.
- **only_metadata** (*bool, optional*) – if True download only metadata, significantly faster. The default is False.
- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even without measurements
- **epsg** (*int, optional*) – epsg code of the extent. The default is 28992 (RD).
- **ignore_max_obs** (*bool, optional*) – by default you get a prompt if you want to download over a 1000 observations at once. if ignore_max_obs is True you won't get the prompt. The default is False

Returns
ObsCollection DataFrame with the 'obs' column

Return type
ObsCollection

classmethod from_bronhouderportaal_bro(*dirname*, *full_meta=False*)

Get all the metadata from dirname.

Parameters

- **dirname** (*str*) – name of dirname that holds XML files
- **full_meta** (*bool , optional*) – process all metadata. The default is False.

Returns
ObsCollection DataFrame without the 'obs' column

Return type
ObsCollection

classmethod from_dataframe(*df*, *obs_list=None*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*)

Create an observation collection from a DataFrame by adding a column with empty observations.

Parameters

- **df** (*pandas DataFrame*) – input dataframe. If this dataframe has a column named 'obs' the column is replaced with empty observation objects.
- **obs_list** (*list of observation.Obs, optional*) – list of observations. Default is None
- **ObsClass** (*class, optional*) – observation class used to create empty obs object, by default obs.GroundwaterObs

Returns
ObsCollection DataFrame with the 'obs' column

Return type
ObsCollection

classmethod from_dino(*dirname=None*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *subdir='Grondwaterstanden_Put'*, *suffix='1.csv'*, *keep_all_obs=True*, *name=None*, ***kwargs*)

Read dino data within an extent from the server or from a directory with downloaded files.

Parameters

- **dirname** (*str, optional*) – directory name, can be a .zip file or the parent directory of subdir
- **ObsClass** (*type*) – class of the observations, so far only GroundwaterObs is supported
- **subdir** (*str*) – subdirectory of dirname with data files
- **suffix** (*str*) – suffix of files in subdir that will be read
- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even the points without measurements or metadata
- **name** (*str, optional*) – the name of the observation collection
- **kwargs** – kwargs are passed to the hydropandas.io.dino.read_dino_dir() function

Returns
cls(obs_df) – collection of multiple point observations

Return type
ObsCollection

classmethod from_excel(*path*, *meta_sheet_name='metadata'*)

Create an observation collection from an excel file. The excel file should have the same format as excel files created with the *to_excel* method of an ObsCollection.

Parameters

- **path** (*str*) – full file path (including extension) of the excel file.
- **meta_sheet_name** (*str, optional*) – sheetname with metadata. The default is "metadata".

Return type
ObsCollection

Notes

if you write an excel file using the 'to_excel' method and read an excel with the 'read_excel' method you lose this information: - The 'name' and 'meta' attributes of the ObsCollection - metadata of each Observation stored in the 'meta' attribute

If you don't want to lose this data consider using the *to_pickle* and *read_pickle* function.

classmethod from_fews_xml(*file_or_dir=None*, *xmlstring=None*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *name='fews'*, *translate_dic=None*, *filterdict=None*, *locations=None*, *remove_nan=True*, *low_memory=True*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, ***kwargs*)

Read one or several FEWS PI-XML files.

Parameters

- **file_or_dir** (*str*) – zip, xml or directory with zips or xml files to read
- **xmlstring** (*str or None*) – string with xml data, only used if file_or_dir is None. Default is None
- **ObsClass** (*type*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **name** (*str, optional*) – name of the observation collection, 'fews' by default
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'locatie'}.
- **filterdict** (*dict, optional*) – dictionary with tag name to apply filter to as keys, and list of accepted names as dictionary values to keep in final result, i.e. {"locationId": ["B001", "B002"]}
- **locations** (*list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read. Only supported by low_memory=True method!
- **low_memory** (*bool, optional*) – whether to use xml-parsing method with lower memory footprint, default is True
- **remove_nan** (*boolean, optional*) – remove nan values from measurements, flag information about the nan values is also lost, only used if low_memory=False
- **unpackdir** (*str*) – destination directory to unzip file if path is a .zip
- **force_unpack** (*boolean, optional*) – force unpack if dst already exists
- **preserve_datetime** (*boolean, optional*) – whether to preserve datetime from zip archive

Returns
cls(obs_df) – collection of multiple point observations

Return type
ObsCollection

classmethod from_imod(*obs_collection*, *ml*, *runfile*, *mtime*, *model_ws*, *modelname=''*, *nlay=None*, *exclude_layers=0*)

Read imod model results at point locations.

Parameters

- **obs_collection** (*ObsCollection*) – collection of observations at which points imod results will be read
- **ml** (*flopy.modflow.mf.model*) – modflow model
- **runfile** (*Runfile*) – imod runfile object
- **mtime** (*list of datetimes*) – datetimes corresponding to the model periods
- **model_ws** (*str*) – model workspace with imod model
- **nlay** (*int, optional*) – number of layers if None the number of layers from ml is used.
- **modelname** (*str*) – modelname
- **exclude_layers** (*int*) – exclude modellayers from being read from imod

classmethod from_knmi(*locations=None*, *stns=None*, *xy=None*, *meteo_vars=('RH',)*, *name=''*, *starts=None*, *ends=None*, *ObsClasses=None*, *fill_missing_obs=False*, *interval='daily'*, *use_api=True*, *raise_exceptions=True*)

Get knmi observations from a list of locations or a list of stations.

Parameters

- **locations** (*pandas DataFrame or None*) – dataframe with columns 'x' and 'y' as coordinates. The default is None
- **stns** (*list of str or None*) – list of knmi stations. The default is None
- **xy** (*list or numpy array, optional*) – xy coordinates of the locations. e.g. [[10,25], [5,25]]
- **meteo_vars** (*list or tuple of str*) – meteo variables e.g. ["RH", "EV24"]. The default is ("RH"). See list of all possible variables in the hpd.read_knmi docstring.
- **name** (*str, optional*) – name of the obscollection. The default is ''
- **starts** (*None, str, datetime or list, optional*) – start date of observations per meteo variable. The start date is included in the time series. If start is None the start date will be January 1st of the previous year. If start is str it will be converted to datetime. If start is a list it should be the same length as meteo_vars and the start time for each variable. The default is None
- **ends** (*list of str, datetime or None*) – end date of observations per meteo variable. The end date is included in the time series. If end is None the start date will be January 1st of the previous year. If end is a str it will be converted to datetime. If end is a list it should be the same length as meteo_vars and the end time for each meteo variable. The default is None
- **ObsClasses** (*list of type or None*) – class of the observations, can be PrecipitationObs, EvaporationObs or MeteoObs. If None the type of observations is derived from the meteo_vars.

- **fill_missing_obs** (*bool, optional*) – if True nan values in time series are filled with nearby time series. The default is False.
- **interval** (*str, optional*) – desired time interval for observations. Options are 'daily' and 'hourly'. The default is 'daily'.
- **use_api** (*bool, optional*) –

  if True the api is used to obtain the data, API documentation is here:
  https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script

  if False a text file is downloaded into a temporary folder and the data is read from there. Default is True since the api is back online (July 2021).
- **raise_exceptions** (*bool, optional*) – if True you get errors when no data is returned. The default is False.
- ****kwargs** – kwargs are passed to the *hydropandas.io.knmi.get_knmi_obslist* function

classmethod from_list(*obs_list*, *name=''*)

Read observations from a list of obs objects.

Parameters

- **obs_list** (*list of observation.Obs*) – list of observations
- **name** (*str, optional*) – name of the observation collection

classmethod from_lizard(*extent=None*, *codes=None*, *name=''*, *tube_nr='all'*, *tmin=None*, *tmax=None*, *type_timeseries='merge'*, *only_metadata=False*)

Get all observations within a specified extent.

Parameters

- **extent** (*list, shapefile path or None*) – get groundwater monitoring wells wihtin this extent [xmin, ymin, xmax, ymax] or within a predefined Polygon from a shapefile
- **codes** (*lst of str or None*) – codes of the monitoring wells
- **tube_nr** (*lst of str*) – list of tube numbers of the monitoring wells that should be selected. By default 'all' available tubes are selected.
- **tmin** (*str YYYY-m-d, optional*) – start of the observations, by default the entire serie is returned
- **tmax** (*Ttr YYYY-m-d, optional*) – end of the observations, by default the entire serie is returned
- **type_timeseries** (*str, optional*) – hand: returns only hand measurements diver: returns only diver measurements merge: the hand and diver measurements into one time series (default) combine: keeps hand and diver measurements separeted The default is merge.
- **only_metadata** (*bool, optional*) – if True only metadata is returned and no time series data. The default is False.

Returns

ObsCollection DataFrame with the 'obs' column

Return type

ObsCollection

classmethod from_menyanthes(*path*, *name=''*, *ObsClass=<class 'hydropandas.observation.Obs'>*, *load_oseries=True*, *load_stresses=True*)

classmethod from_modflow(*obs_collection*, *ml*, *hds_arr*, *mtime*, *modelname=''*, *nlay=None*, *exclude_layers=None*, *method='linear'*)

Read modflow groundwater heads at points in obs_collection.

Parameters

- **obs_collection** (ObsCollection) – locations of model observation
- **ml** (*flopy.modflow.mf.model*) – modflow model
- **hds_arr** (*numpy array*) – heads with shape (ntimesteps, nlayers, nrow, ncol)
- **mtime** (*list of datetimes*) – dates for each model timestep
- **modelname** (*str, optional*) – modelname
- **nlay** (*int, optional*) – number of layers if None the number of layers from ml is used.
- **exclude_layers** (*list of int, optional*) – exclude the observations in these model layers
- **method** (*str, optional*) – interpolation method, either 'linear' or 'nearest', default is linear

classmethod from_pastastore(*pstore*, *libname*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *metadata_mapping=None*)

Read pastastore library.

Parameters

- **pstore** (*pastastore.PastaStore*) – PastaStore object
- **libname** (*str*) – name of library (e.g. oseries or stresses)
- **ObsClass** (Obs, *optional*) – type of Obs to read data as, by default obs.GroundwaterObs
- **metadata_mapping** (*dict, optional*) – dictionary containing map between metadata field names in pastastore and metadata field names expected by hydropandas, by default None.

Returns

ObsCollection containing data

Return type

ObsCollection

classmethod from_waterinfo(*file_or_dir*, *name=''*, *ObsClass=<class 'hydropandas.observation.WaterlvlObs'>*, *progressbar=True*, ***kwargs*)

Read waterinfo file or directory.

Parameters

- **file_or_dir** (*str*) – path to file or directory. Files can be .csv or .zip
- **name** (*str, optional*) – name of the collection, by default ""
- **ObsClass** (Obs, *optional*) – type of Obs to read data as, by default obs.WaterlvlObs
- **progressbar** (*bool, optional*) – show progressbar, by default True

Returns
ObsCollection containing data

Return type
ObsCollection

classmethod from_wiski(*dirname*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *suffix='.csv'*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, *keep_all_obs=True*, ***kwargs*)

geo

alias of *GeoAccessor*

get_obs(*name=None*, ***kwargs*)

get an observation object from a collection

Parameters

- **name** (*str or None, optional*) – name of the observation you want to select, by default None
- ****kwargs** (*any metadata, value pair e.g. for a collection of GroundwaterObs:*) – tube_nr = 1 or source = 'BRO'

Returns
Observation object from the collection.

Return type
hpd.Obs

Raises

- **ValueError** – If multiple observations in the collection match the given attribute values.
- **ValueError** – If no observation in the collection match the given attribute values.

get_series(*tmin=None*, *tmax=None*, *col=None*)

Parameters

- **tmin** (*datetime, optional*) – start time for series. The default is None.
- **tmax** (*datetime, optional*) – end time for series. The default is None.
- **col** (*str or None, optional*) – the column of the obs dataframe to get measurements from. The first numeric column is used if col is None, by default None.

Returns
series of a series of observations within a time frame.

Return type
series of Series

gwobs

alias of *GwObsAccessor*

interpolate(*xy: List[List[float]]*, *kernel: str = 'thin_plate_spline'*, *kernel2: str = 'linear'*, *epsilon: Optional[int] = None*, *col: Optional[str] = None*)

Interpolation method for ObsCollections using the Scipy radial basis function (RBF)

Parameters

- **xy** (*List[List[float]]*) – xy coordinates of locations of interest e.g. [[10,25], [5,25]]
- **kernel** (*str, optional*) – Type of radial basis funtion, by default thin_plate_spline. Other options are linear, gaussian, inverse_quadratic, multiquadric, inverse_multiquadric, cubic or quintic.
- **kernel2** (*str, optional*) – Kernel in case there are not enough observations (3 or 6) for time step, by default linear. Other options are gaussian, inverse_quadratic, multiquadric, or inverse_multiquadric.
- **epsilon** (*float, optional*) – Shape parameter that scales the input to the RBF. If kernel is linear, thin_plate_spline, cubic, or quintic, this defaults to 1. Otherwise this must be specified.
- **col** (*str, optional*) – Name of the column in the Obs dataframe to be used. If None the first numeric column in the Obs Dataframe is used.

Return type
ObsCollection

plots

alias of *CollectionPlots*

stats

alias of *StatsAccessor*

to_excel(*path*, *meta_sheet_name='metadata'*)

Write an ObsCollection to an excel, the first sheet in the excel contains the metadata, the other tabs are the timeseries of each observation.

The excel can be read using the read_excel function of hydropandas.

Parameters

- **path** (*str*) – full path of xlsx file.
- **meta_sheet_name** (*str, optional*) – sheetname with metadata. The default is "metadata".

Raises
RuntimeError – If the ObsCollection is inconsistent.

Return type
None.

Notes

The following data is NOT written to the excel file: - The 'name' and 'meta' attributes of the ObsCollection - metadata of each Observation stored in the 'meta' dictionary

If you don't want this consider using the *to_pickle* method.

to_gdf(*xcol='x'*, *ycol='y'*, *crs=28992*, *drop_obs=True*)

Convert ObsCollection to GeoDataFrame.

Parameters

- **xcol** (*str*) – column name with x values
- **ycol** (*str*) – column name with y values

- **crs** (*int, optional*) – coordinate reference system, by default 28992 (RD new).
- **drop_obs** (*bool, optional*) – drop the column with observations. Useful for basic geodataframe manipulations that require JSON serializable columns. The default is True.

Returns
gdf

Return type
geopandas.GeoDataFrame

to_pastastore(*pstore=None*, *pstore_name=''*, *col=None*, *kind='oseries'*, *add_metadata=True*, *conn=None*, *overwrite=False*)

Add observations to a new or existing pastastore.

Parameters

- **pstore** (*pastastore.PastaStore, optional*) – Existing pastastore, if None a new pastastore is created
- **pstore_name** (*str, optional*) – Name of the pastastore only used if pstore is None
- **col** (*str, optional*) – Name of the column in the Obs dataframe to be used. If None the first numeric column in the Obs Dataframe is used.
- **kind** (*str, optional*) – The kind of series that is added to the pastastore. Use 'oseries' for observations and anything else for stresses.
- **add_metadata** (*boolean, optional*) – If True metadata from the observations added to the pastastore
- **conn** (*pastastore.connectors or None, optional*) – type of connector, if None the DictConnector is used. Default is None.
- **overwrite** (*boolean, optional*) – if True, overwrite existing series in pastastore, default is False

Returns
pstore – the pastastore with the series from the ObsCollection

Return type
pastastore.PastaStore

to_pi_xml(*fname*, *timezone=''*, *version='1.24'*)

to_shapefile(*path*, *xcol='x'*, *ycol='y'*)

Save ObsCollection as shapefile.

Parameters

- **path** (*str*) – filename of shapefile (.shp) or geopackage (.gpkg). A geopackage has the advantage that column names will not be truncated.
- **xcol** (*str*) – column name with x values
- **ycol** (*str*) – column name with y values

hydropandas.obs_collection.**read_bro**(*extent=None*, *bro_id=None*, *name=''*, *tmin=None*, *tmax=None*, *only_metadata=False*, *keep_all_obs=True*, *epsg=28992*, *ignore_max_obs=False*)

Get all the observations within an extent or within a groundwatermonitoring net.

Parameters

- **extent** (*list, tuple, numpy-array or None, optional*) – get groundwater monitoring wells within this extent [xmin, xmax, ymin, ymax]
- **bro_id** (*str or None, optional*) – starts with 'GMN'.
- **name** (*str, optional*) – name of the observation collection
- **tmin** (*str or None, optional*) – start time of observations. The default is None.
- **tmax** (*str or None, optional*) – end time of observations. The default is None.
- **only_metadata** (*bool, optional*) – if True download only metadata, significantly faster. The default is False.
- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even without measurements
- **epsg** (*int, optional*) – epsg code of the extent. The default is 28992 (RD).
- **ignore_max_obs** (*bool, optional*) – by default you get a prompt if you want to download over a 1000 observations at once. if ignore_max_obs is True you won't get the prompt. The default is False

Returns
ObsCollection DataFrame with the 'obs' column

Return type
ObsCollection

hydropandas.obs_collection.**read_bronhouderportaal_bro**(*dirname*, *full_meta=False*, *add_to_df=False*)

get all the metadata from files in a directory. Files are GMW files of well construction, and are subbmitted to https://www.bronhouderportaal-bro.nl .

Parameters

- **dirname** (*str*) – name of directory that holds XML files
- **full_meta** (*bool, optional*) – process not only the standard metadata to ObsCollection
- **add_to_df** (*bool, optional*) – add all the metadata to the ObsCollection DataFrame

Returns
ObsCollection DataFrame without the 'obs' column

Return type
ObsCollection

hydropandas.obs_collection.**read_dino**(*dirname=None*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *subdir='Grondwaterstanden_Put'*, *suffix='1.csv'*, *keep_all_obs=True*, *name=None*, ***kwargs*)

Read dino observations within an extent from the server or from a directory with downloaded files.

Parameters

- **dirname** (*str, optional*) – directory name, can be a .zip file or the parent directory of subdir
- **ObsClass** (*type*) – class of the observations, so far only GroundwaterObs is supported
- **subdir** (*str*) – subdirectory of dirname with data files
- **suffix** (*str*) – suffix of files in subdir that will be read

- **keep_all_obs** (*boolean, optional*) – add all observation points to the collection, even the points without measurements or metadata
- **name** (*str, optional*) – the name of the observation collection
- **kwargs** – kwargs are passed to the hydropandas.io.dino.read_dino_dir() function

Returns
collection of multiple point observations

Return type
ObsCollection

hydropandas.obs_collection.**read_excel**(*path*, *meta_sheet_name='metadata'*)

Create an observation collection from an excel file. The excel file should have the same format as excel files created with the *to_excel* method of an ObsCollection.

Parameters

- **path** (*str*) – full file path (including extension) of the excel file.
- **meta_sheet_name** (*str, optional*) – sheetname with metadata. The default is "metadata".

Return type
ObsCollection

Notes

if you write an excel file using the 'to_excel' method and read an excel with the 'read_excel' method you lose this information: - The 'name' and 'meta' attributes of the ObsCollection - metadata of each Observation stored in the 'meta' attribute

If you don't want to lose this data consider using the *to_pickle* and *read_pickle* function.

hydropandas.obs_collection.**read_fews**(*file_or_dir=None*, *xmlstring=None*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *name='fews'*, *translate_dic=None*, *filterdict=None*, *locations=None*, *remove_nan=True*, *low_memory=True*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, ***kwargs*)

Read one or several FEWS PI-XML files.

Parameters

- **file_or_dir** (*str*) – zip, xml or directory with zips or xml files to read
- **xmlstring** (*str or None*) – string with xml data, only used if file_or_dir is None. Default is None
- **ObsClass** (*type*) – class of the observations, e.g. GroundwaterObs or WaterlvlObs
- **name** (*str, optional*) – name of the observation collection, 'fews' by default
- **translate_dic** (*dic or None, optional*) – translate names from fews. If None this default dictionary is used: {'locationId': 'locatie'}.
- **filterdict** (*dict, optional*) – dictionary with tag name to apply filter to as keys, and list of accepted names as dictionary values to keep in final result, i.e. {"locationId": ["B001", "B002"]}

- **locations** (*list of str, optional*) – list of locationId's to read from XML file, others are skipped. If None (default) all locations are read. Only supported by low_memory=True method!
- **low_memory** (*bool, optional*) – whether to use xml-parsing method with lower memory footprint, default is True
- **remove_nan** (*boolean, optional*) – remove nan values from measurements, flag information about the nan values is also lost, only used if low_memory=False
- **unpackdir** (*str*) – destination directory to unzip file if file_or_dir is a .zip
- **force_unpack** (*boolean, optional*) – force unpack if dst already exists
- **preserve_datetime** (*boolean, optional*) – whether to preserve datetime from zip archive

Returns
collection of multiple point observations

Return type
ObsCollection

hydropandas.obs_collection.**read_imod**(*obs_collection*, *ml*, *runfile*, *mtime*, *model_ws*, *modelname=''*, *nlay=None*, *exclude_layers=0*)

Read imod model results at point locations.

Parameters

- **obs_collection** (ObsCollection) – collection of observations at which points imod results will be read
- **ml** (*flopy.modflow.mf.model*) – modflow model
- **runfile** (*Runfile*) – imod runfile object
- **mtime** (*list of datetimes*) – datetimes corresponding to the model periods
- **model_ws** (*str*) – model workspace with imod model
- **nlay** (*int, optional*) – number of layers if None the number of layers from ml is used.
- **modelname** (*str*) – modelname
- **exclude_layers** (*int*) – exclude modellayers from being read from imod

Returns
collection of multiple point observations

Return type
ObsCollection

hydropandas.obs_collection.**read_knmi**(*locations=None*, *stns=None*, *xy=None*, *meteo_vars=('RH',)*, *name=''*, *starts=None*, *ends=None*, *ObsClasses=None*, *fill_missing_obs=False*, *interval='daily'*, *use_api=True*, *raise_exceptions=True*)

Get knmi observations from a list of locations or a list of stations.

Parameters

- **locations** (*pandas DataFrame or None*) – dataframe with columns 'x' and 'y' as coordinates. The default is None
- **stns** (*list of str or None*) – list of knmi stations. The default is None

- **xy** (*list or numpy array, optional*) – xy coordinates of the locations. e.g. [[10,25], [5,25]]
- **meteo_vars** (*list or tuple of str*) – meteo variables e.g. ["RH", "EV24"]. The default is ("RH"). See list of all possible variables below
- **name** (*str, optional*) – name of the obscollection. The default is ''
- **starts** (*None, str, datetime or list, optional*) – start date of observations per meteo variable. The start date is included in the time series. If start is None the start date will be January 1st of the previous year. If start is str it will be converted to datetime. If start is a list it should be the same length as meteo_vars and the start time for each variable. The default is None
- **ends** (*list of str, datetime or None*) – end date of observations per meteo variable. The end date is included in the time series. If end is None the start date will be January 1st of the previous year. If end is a str it will be converted to datetime. If end is a list it should be the same length as meteo_vars and the end time for each meteo variable. The default is None
- **ObsClasses** (*list of type or None*) – class of the observations, can be PrecipitationObs, EvaporationObs or MeteoObs. If None the type of observations is derived from the meteo_vars.
- ****kwargs** – kwargs are passed to the hydropandas.io.knmi.get_knmi_obslist function
- **variables** (*List of possible*) – neerslagstations: RD = de 24-uurs neerslagsom, gemeten van 0800 utc op de voorafgaande dag tot 0800 utc op de vermelde datum meteostations: DDVEC = Vectorgemiddelde windrichting in graden (360=noord, 90=oost, 180=zuid, 270=west, 0=windstil/variabel). Zie http://www.knmi.nl/kennis-en-datacentrum/achtergrond/klimatologische-brochures-en-boeken / Vector mean wind direction in degrees (360=north, 90=east, 180=south, 270=west, 0=calm/variable) FHVEC = Vectorgemiddelde windsnelheid (in 0.1 m/s). Zie http://www.knmi.nl/kennis-en-datacentrum/achtergrond/klimatologische-brochures-en-boeken / Vector mean windspeed (in 0.1 m/s) FG = Etmaalgemiddelde windsnelheid (in 0.1 m/s) / Daily mean windspeed (in 0.1 m/s) FHX = Hoogste uurgemiddelde windsnelheid (in 0.1 m/s) / Maximum hourly mean windspeed (in 0.1 m/s) FHXH = Uurvak waarin FHX is gemeten / Hourly division in which FHX was measured FHN = Laagste uurgemiddelde windsnelheid (in 0.1 m/s) / Minimum hourly mean windspeed (in 0.1 m/s) FHNH = Uurvak waarin FHN is gemeten / Hourly division in which FHN was measured FXX = Hoogste windstoot (in 0.1 m/s) / Maximum wind gust (in 0.1 m/s) FXXH = Uurvak waarin FXX is gemeten / Hourly division in which FXX was measured TG = Etmaalgemiddelde temperatuur (in 0.1 graden Celsius) / Daily mean temperature in (0.1 degrees Celsius) TN = Minimum temperatuur (in 0.1 graden Celsius) / Minimum temperature (in 0.1 degrees Celsius) TNH = Uurvak waarin TN is gemeten / Hourly division in which TN was measured TX = Maximum temperatuur (in 0.1 graden Celsius) / Maximum temperature (in 0.1 degrees Celsius) TXH = Uurvak waarin TX is gemeten / Hourly division in which TX was measured T10N = Minimum temperatuur op 10 cm hoogte (in 0.1 graden Celsius) / Minimum temperature at 10 cm above surface (in 0.1 degrees Celsius) T10NH = 6-uurs tijdvak waarin T10N is gemeten / 6-hourly division in which T10N was measured; 6=0-6 UT, 12=6-12 UT, 18=12-18 UT, 24=18-24 UT SQ = Zonneschijnduur (in 0.1 uur) berekend uit de globale straling (-1 voor <0.05 uur) / Sunshine duration (in 0.1 hour) calculated from global radiation (-1 for <0.05 hour) SP = Percentage van de langst mogelijke zonneschijnduur / Percentage of maximum potential sunshine duration Q = Globale straling (in J/cm2) / Global radiation (in J/cm2) DR = Duur van de neerslag (in 0.1 uur) / Precipitation duration (in 0.1 hour) RH = Etmaalsom van de neerslag (in 0.1 mm) (-1 voor <0.05 mm) / Daily precipitation amount

(in 0.1 mm) (-1 for <0.05 mm) RHX = Hoogste uursom van de neerslag (in 0.1 mm) (-1 voor <0.05 mm) / Maximum hourly precipitation amount (in 0.1 mm) (-1 for <0.05 mm) RHXH = Uurvak waarin RHX is gemeten / Hourly division in which RHX was measured PG = Etmaalgemiddelde luchtdruk herleid tot zeeniveau (in 0.1 hPa) berekend uit 24 uurwaarden / Daily mean sea level pressure (in 0.1 hPa) calculated from 24 hourly values PX = Hoogste uurwaarde van de luchtdruk herleid tot zeeniveau (in 0.1 hPa) / Maximum hourly sea level pressure (in 0.1 hPa) PXH = Uurvak waarin PX is gemeten / Hourly division in which PX was measured PN = Laagste uurwaarde van de luchtdruk herleid tot zeeniveau (in 0.1 hPa) / Minimum hourly sea level pressure (in 0.1 hPa) PNH = Uurvak waarin PN is gemeten / Hourly division in which PN was measured P = Luchtdruk (in 0.1 hPa) herleid tot zeeniveau, op het moment van meten / Air pressure (in 0.1 hPa) reduced to mean sea level, at the time of observation VVN = Minimum opgetreden zicht / Minimum visibility; 0: <100 m, 1:100-200 m, 2:200-300 m,..., 49:4900-5000 m, 50:5-6 km, 56:6-7 km, 57:7-8 km,..., 79:29-30 km, 80:30-35 km, 81:35-40 km, ..., 89: >70 km) VVNH = Uurvak waarin VVN is gemeten / Hourly division in which VVN was measured VVX = Maximum opgetreden zicht / Maximum visibility; 0: <100 m, 1:100-200 m, 2:200-300 m,..., 49:4900-5000 m, 50:5-6 km, 56:6-7 km, 57:7-8 km,..., 79:29-30 km, 80:30-35 km, 81:35-40 km, ..., 89: >70 km) VVXH = Uurvak waarin VVX is gemeten / Hourly division in which VVX was measured NG = Etmaalgemiddelde bewolking (bedekkingsgraad van de bovenlucht in achtsten, 9=bovenlucht onzichtbaar) / Mean daily cloud cover (in octants, 9=sky invisible) UG = Etmaalgemiddelde relatieve vochtigheid (in procenten) / Daily mean relative atmospheric humidity (in percents) UX = Maximale relatieve vochtigheid (in procenten) / Maximum relative atmospheric humidity (in percents) UXH = Uurvak waarin UX is gemeten / Hourly division in which UX was measured UN = Minimale relatieve vochtigheid (in procenten) / Minimum relative atmospheric humidity (in percents) UNH = Uurvak waarin UN is gemeten / Hourly division in which UN was measured EV24 = Referentiegewasverdamping (Makkink) (in 0.1 mm) / Potential evapotranspiration (Makkink) (in 0.1 mm)

Returns

collection of multiple point observations

Return type

ObsCollection

hydropandas.obs_collection.**read_lizard**(*extent=None*, *codes=None*, *name=''*, *tube_nr='all'*, *tmin=None*, *tmax=None*, *type_timeseries='merge'*, *only_metadata=False*)

Get all observations from a list of codes of the monitoring wells and a list of tube numbers.

Parameters

- **extent** (*list, shapefile path or None*) – get groundwater monitoring wells within this extent [xmin, ymin, xmax, ymax] or within a predefined Polygon from a shapefile
- **codes** (*lst of str or None*) – codes of the monitoring wells
- **tube_nr** (*lst of str*) – list of tube numbers of the monitoring wells that should be selected. By default 'all' available tubes are selected.
- **tmin** (*str YYYY-m-d, optional*) – start of the observations, by default the entire time series is returned
- **tmax** (*Ttr YYYY-m-d, optional*) – end of the observations, by default the entire time series is returned
- **type_timeseries** (*str, optional*) – hand: returns only hand measurements diver: returns only diver measurements merge: the hand and diver measurements into one time series (default) combine: keeps hand and diver measurements separeted

- **only_metadata** (*bool, optional*) – if True only metadata is returned and no time series data. The default is False.

Returns
ObsCollection DataFrame with the 'obs' column

Return type
ObsCollection

hydropandas.obs_collection.**read_menyanthes**(*path*, *name=''*, *ObsClass=<class 'hydropandas.observation.Obs'>*, *load_oseries=True*, *load_stresses=True*)

Read a Menyanthes file.

Parameters

- **path** (*str*) – full path of the .men file.
- **name** (*str, optional*) – name of the observation collection. The default is "".
- **ObsClass** (*type, optional*) – class of the observations, e.g. GroundwaterObs. The default is obs.Obs.
- **load_oseries** (*bool, optional*) – if True the observations are read. The default is True.
- **load_stresses** (*bool, optional*) – if True the stresses are read. The default is True.

Returns
collection of multiple point observations

Return type
ObsCollection

hydropandas.obs_collection.**read_modflow**(*obs_collection*, *ml*, *hds_arr*, *mtime*, *modelname=''*, *nlay=None*, *exclude_layers=None*, *method='linear'*)

Read modflow groundwater heads at locations in obs_collection.

Parameters

- **obs_collection** (ObsCollection) – locations of model observation
- **ml** (*flopy.modflow.mf.model*) – modflow model
- **hds_arr** (*numpy array*) – heads with shape (ntimesteps, nlayers, nrow, ncol)
- **mtime** (*list of datetimes*) – dates for each model timestep
- **modelname** (*str, optional*) – modelname
- **nlay** (*int, optional*) – number of layers if None the number of layers from ml is used.
- **exclude_layers** (*list of int, optional*) – exclude the observations in these model layers
- **method** (*str, optional*) – interpolation method, either 'linear' or 'nearest', default is linear

Returns
collection of multiple point observations

Return type
ObsCollection

hydropandas.obs_collection.**read_pastastore**(*pstore*, *libname*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *metadata_mapping=None*)

Read pastastore library.

Parameters

- **pstore** (*pastastore.PastaStore*) – PastaStore object
- **libname** (*str*) – name of library (e.g. oseries or stresses)
- **ObsClass** (*Obs, optional*) – type of Obs to read data as, by default obs.GroundwaterObs
- **metadata_mapping** (*dict, optional*) – dictionary containing map between metadata field names in pastastore and metadata field names expected by hydropandas, by default None.

Returns

ObsCollection containing data

Return type

ObsCollection

hydropandas.obs_collection.**read_pickle**(*filepath_or_buffer*, *compression='infer'*, *storage_options=None*)

Wrapper around pd.read_pickle.

Parameters

- **filepath_or_buffer** (*str, path object, or file-like object*) – String, path object (implementing `os.PathLike[str]`), or file-like object implementing a binary `readlines()` function.

  Changed in version 1.0.0.

  Accept URL. URL is not limited to S3 and GCS.

- **compression** (*str or dict, default 'infer'*) – For on-the-fly decompression of on-disk data. If 'infer' and 'filepath_or_buffer' is path-like, then detect compression from the following extensions: '.gz','.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). If using 'zip', the ZIP file must contain only one data file to be read in. Set to `None` for no decompression. Can also be a dict with key `'method'` set to one of {`'zip'`, `'gzip'`, `'bz2'`, `'zstd'`} and other key-value pairs are forwarded to `zipfile.ZipFile`, `gzip.GzipFile`, `bz2.BZ2File`, or `zstandard.ZstdDecompressor`, respectively. As an example, the following could be passed for Zstandard decompression using a custom compression dictionary: `compression={'method': 'zstd', 'dict_data': my_compression_dict}`.

  Changed in version 1.4.0: Zstandard support.

- **storage_options** (*dict, optional*) – Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc. For HTTP(S) URLs the key-value pairs are forwarded to `urllib` as header options. For other URLs (e.g. starting with "s3://", and "gcs://") the key-value pairs are forwarded to `fsspec`. Please see `fsspec` and `urllib` for more details.

  New in version 1.2.0.

Returns

ObsCollection

Return type
same type as object stored in file

hydropandas.obs_collection.**read_waterinfo**(*file_or_dir*, *name=''*, *ObsClass=<class 'hydropandas.observation.WaterlvlObs'>*, *progressbar=True*, ***kwargs*)

Read waterinfo file or directory.

Parameters

- **file_or_dir** (*str*) – path to file or directory. Files can be .csv or .zip
- **name** (*str, optional*) – name of the collection, by default ""
- **ObsClass** (*Obs, optional*) – type of Obs to read data as, by default obs.WaterlvlObs
- **progressbar** (*bool, optional*) – show progressbar, by default True

Returns
ObsCollection containing data

Return type
ObsCollection

hydropandas.obs_collection.**read_wiski**(*dirname*, *ObsClass=<class 'hydropandas.observation.GroundwaterObs'>*, *suffix='.csv'*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*, *keep_all_obs=True*, ***kwargs*)

Parameters

- **dirname** (*str*) – path of the zipfile with wiski data.
- **ObsClass** (*type, optional*) – type of Obs. The default is obs.GroundwaterObs.
- **suffix** (*str, optional*) – extension of filenames to read. The default is ".csv".
- **unpackdir** (*str or None, optional*) – directory to unpack zipped directory. The default is None.
- **force_unpack** (*bool, optional*) – force unzip, by default False.
- **preserve_datetime** (*bool, optional*) – preserve datetime of unzipped files, by default False (useful for checking whether data has changed)
- **keep_all_obs** (*bool, optional*) – If True keep all observations even those without metadata. The default is True.
- ****kwargs** –

Returns
ObsCollection containing observation data

Return type
ObsCollection

1.4.3 hydropandas.observation module

Module with observation classes.

The Obs class is a subclass of a pandas DataFrame with additional attributes and methods. The specific classes (GroundwaterObs, WaterlvlObs, ...) are subclasses of the Obs class.

The subclasses of a dataframe can have additional attributes and methods. Additional attributes have to be defined in the '_metadata' attribute. In order to keep the subclass methods and attributes when selecting or slicing an object you need the '_constructor' method.

More information about subclassing pandas DataFrames can be found here: http://pandas.pydata.org/pandas-docs/stable/development/extending.html#extending-subclassing-pandas

class hydropandas.observation.**EvaporationObs**(**args*, ***kwargs*)

Bases: *MeteoObs*

Class for evaporation timeseries.

Subclass of the MeteoObs class

classmethod from_knmi(*meteo_var='EV24'*, *stn=None*, *fname=None*, *xy=None*, *start=None*, *end=None*, *fill_missing_obs=False*, *interval='daily'*, *use_api=True*, *raise_exceptions=True*, *startdate=None*, *enddate=None*)

Get an EvaporationObs timeseries from the KNMI evaporation in m.

Parameters

- **meteo_var** (*str, optional*) – meteo variable should be "EV24".
- **stn** (*int, str or None, optional*) – measurement station e.g. 829. The default is None.
- **fname** (*str, path object, file-like object or None, optional*) – filename of a knmi file. The default is None.
- **xy** (*list, tuple or None, optional*) – RD coördinates of a location in the Netherlands. The station nearest to this location used. The Default is None.
- **start** (*str, datetime or None, optional*) – start date of observations. The default is None.
- **end** (*str, datetime or None, optional*) – end date of observations. The default is None.
- **fill_missing_obs** (*bool, optional*) – if True nan values in time series are filled with nearby time series. The default is False.
- **interval** (*str, optional*) – desired time interval for observations. Options are 'daily' and 'hourly'. The default is 'daily'.
- **inseason** (*boolean, optional*) – flag to obtain inseason data. The default is False
- **raise_exceptions** (*bool, optional*) – if True you get errors when no data is returned. The default is False.
- **use_api** (*bool, optional*) –

  if True the api is used to obtain the data, API documentation is here:
  https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script

  if False a text file is downloaded into a temporary directory and the data is read from there. Default is True since the api is back online (July 2021).

Return type
EvaporationObs object with an evaporation time series and attributes

class hydropandas.observation.**GroundwaterObs**(**args*, ***kwargs*)

Bases: *Obs*

Class for groundwater quantity observations.

Subclass of the Obs class. Has the following attributes:

- monitoring_well: 2 tubes at one piezometer should have the same 'monitoring_well'
- tube_nr: 2 tubes at one piezometer should have a different 'tube_nr'.
- screen_top: top op the filter in m above date (NAP)
- screen_bottom: bottom of the filter in m above date (NAP)
- ground_level: surface level in m above date (NAP) (maaiveld in Dutch)
- tube_top: top of the tube in m above date (NAP)
- metadata_available: boolean indicating if metadata is available for the measurement point.

classmethod from_artdino_file(*path=None*, ***kwargs*)

Read a dino csv file (artdiver style).

Parameters

- **path** (*str, optional*) – path of dino csv filename
- **kwargs** (*key-word arguments*) – these arguments are passed to hydropandas.io._dino.read_dino_groundwater_csv

classmethod from_bro(*bro_id*, *tube_nr=None*, *tmin='1900-01-01'*, *tmax='2040-01-01'*, *to_wintertime=True*, *drop_duplicate_times=True*, *only_metadata=False*)

Download BRO groundwater observations from the server.

Parameters

- **bro_id** (*str*) – can be a GLD id or GMW id. If a GMW id is given a tube number is required as well. e.g. 'GLD000000012893'.
- **tube_nr** (*str or None, optional*) – if the bro_id is a GMW object the tube number should be given.
- **tmin** (*str or None, optional*) – start date in format YYYY-MM-DD
- **tmax** (*str or None, optional*) – end date in format YYYY-MM-DD
- **to_wintertime** (*bool, optional*) – if True the time index is converted to Dutch winter time. The default is True.
- **drop_duplicate_times** (*bool, optional*) – if True rows with a duplicate time stamp are removed keeping only the first row. The default is True.
- **only_metadata** (*bool, optional*) – if True only metadata is returned and no time series data. The default is False

Returns
DESCRIPTION.

Return type
TYPE

classmethod from_bronhouderportaal_bro(*path*, *tube_nr*, *full_meta=False*)

Load BRO groundwater metadata from XML file. Mind that bro_id is applicable, because file is not yet imported in BRO.

Parameters

- **path** (*str*) – filepath of XML file.
- **tube_nr** (*int*) – tube number.
- **full_meta** (*bool*) – process not only the standard metadata to ObsCollection.

Returns
ObsCollection containing observations from XML file.

Return type
ObsCollection

classmethod from_dino(*path=None*, ***kwargs*)

Download dino data from the server.

Parameters

- **path** (*str, optional*) – path of dino csv file
- **kwargs** (*key-word arguments*) – these arguments are passed to hydropandas.io.dino.read_dino_groundwater_csv if path is not None and otherwise to hydropandas.io.dino.findMeetreeks

classmethod from_lizard(*code*, *tube_nr=None*, *tmin=None*, *tmax=None*, *type_timeseries='merge'*, *only_metadata=False*)

Extracts the metadata and timeseries of a observation well from a LIZARD-API based on the code of a monitoring well.

Parameters

- **code** (*str*) – code of the measuring well
- **tube_nr** (*int, optional*) – select specific tube top Default selects tube_nr = 1
- **tmin** (*str YYYY-m-d, optional*) – start of the observations, by default the entire serie is returned
- **tmax** (*Ttr YYYY-m-d, optional*) – end of the observations, by default the entire serie is returned
- **type_timeseries** (*str, optional*) – hand: returns only hand measurements diver: returns only diver measurements merge: the hand and diver measurements into one time series (default) combine: keeps hand and diver measurements separated
- **only_metadata** (*bool, optional*) – if True only metadata is returned and no time series data. The default is False.

Returns
Returns a DataFrame with metadata and timeseries

Return type
ObsCollection

classmethod from_pastastore(*pstore*, *libname*, *name*, *metadata_mapping=None*)

Read item from pastastore library.

Parameters

- **pstore** (*pastastore.PastaStore*) – pastastore object
- **libname** (*str*) – name of library containinig item
- **name** (*str*) – name of item
- **metadata_mapping** (*dict, optional*) – dictionary containing map between metadata field names in pastastore (keys) and metadata field names expected by hydropandas (values), by default None.

classmethod from_solinst(*path*, *transform_coords=True*, *screen_bottom=None*, *screen_top=None*, *ground_level=None*, *tube_nr=None*, *tube_top=None*)

Read data from Solinst xle file.

Parameters
path (*str*) – path to file (file can zip or xle)

classmethod from_wiski(*path*, ***kwargs*)

Read data from a WISKI file.

Parameters:

path
[str] The path of the file to be read.

sep
[str, optional (default=";")] The delimiter used to separate fields in the file.

header_sep
[str, optional (default=None)] The delimiter used to separate fields in the header. If None, the function will try to automatically detect the separator.

header_identifier
[str, optional (default="#")] The character used to identify header lines.

read_series
[bool, optional (default=True)] Whether to read the time series data from the file.

translate_dic
[dict, optional (default=None)] A dictionary mapping header field names to the desired output names.

tz_localize
[bool, optional (default=True)] Whether to localize the datetime index to the machine's timezone.

unit
[str, optional (default="")] The unit of measurement of the data.

****kwargs**
[keyword arguments] Additional arguments to pass to the pandas *read_csv* function.

class hydropandas.observation.**MeteoObs**(**args*, ***kwargs*)

Bases: *Obs*

Class for meteorological timeseries.

Subclass of the Obs class

classmethod from_knmi(*meteo_var*, *stn=None*, *fname=None*, *xy=None*, *start=None*, *end=None*, *fill_missing_obs=False*, *interval='daily'*, *use_api=True*, *raise_exceptions=True*, *startdate=None*, *enddate=None*)

Get a MeteoObs timeseries from the KNMI meteo data.

Parameters

- **meteo_var** (*str*) – meteo variable e.g. "RH" or "EV24". For a list of possible variables see the hydropandas.read_knmi function.
- **stn** (*int, str or None, optional*) – measurement station e.g. 829. The default is None.
- **fname** (*str, path object, file-like object or None, optional*) – filename of a knmi file. The default is None.
- **xy** (*list, tuple or None, optional*) – RD coördinates of a location in the Netherlands. The station nearest to this location used. The Default is None.
- **start** (*str, datetime or None, optional*) – start date of observations. The default is None.
- **end** (*str, datetime or None, optional*) – end date of observations. The default is None.
- **fill_missing_obs** (*bool, optional*) – if True nan values in time series are filled with nearby time series. The default is False.
- **interval** (*str, optional*) – desired time interval for observations. Options are 'daily' and 'hourly'. The default is 'daily'.
- **use_api** (*bool, optional*) –

  if True the api is used to obtain the data, API documentation is here:
  https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script

  if False a text file is downloaded into a temporary directory and the data is read from there. Default is True since the api is back online (July 2021).
- **raise_exceptions** (*bool, optional*) – if True you get errors when no data is returned. The default is False.

Return type
MeteoObs object with meteorological observations

classmethod from_wow(*meteo_var: str*, *stn: str = None*, *xy: List[float] = None*, *start: Optional[Timestamp] = None*, *end: Optional[Timestamp] = None*)

Get a MeteoObs timeseries from a wow.knmi.nl station.

Parameters

- **meteo_var** (*str*) – wow meteo variable
- **stn** (*Optional[int, str], optional*) – station name
- **xy** (*Optinal[List[float]], optinial*) – longitude latitude of location [lon, lat] eg: [4.85, 51.95]
- **start** (*Optional[pd.Timestamp], optional*) – start date of observations, by default None
- **end** (*Optional[pd.Timestamp], optional*) – start date of observations, by default None

Return type
MeteoObs

class hydropandas.observation.**ModelObs**(**args*, ***kwargs*)

Bases: *Obs*

Class for model point results.

Subclass of the Obs class

class hydropandas.observation.**Obs**(**args*, ***kwargs*)

Bases: DataFrame

Generic class for a time series with measurements at a certain location.

Unless specified explicitly the first numeric column in the observation is used for analysis and plotting.

Parameters

- **name** (*str*) – name
- **x** (*int or float*) – x coordinate of observation point
- **y** (*int or float*) – y coordinate of observation point
- **meta** (*dictionary*) – metadata
- **filename** (*str*) – filename with data of observation point
- **source** (*str*) – source of the observation e.g. BRO or KNMI
- **unit** (*str*) – unit of the first numerical column in the observation

copy(*deep=True*)

Create a copy of the observation.

When deep=True (default), a new object will be created with a copy of the calling object's data and indices. Modifications to the data or indices of the copy will not be reflected in the original object (see notes below).

When deep=False, a new object will be created without copying the calling object's data or index (only references to the data and index are copied). Any changes to the data of the original will be reflected in the shallow copy (and vice versa).

Parameters
deep (*bool, default True*) – Make a deep copy, including a copy of the data and the indices. With deep=False neither the indices nor the data are copied.

Returns
o – DESCRIPTION.

Return type
TYPE

geo

alias of *GeoAccessorObs*

gwobs

alias of *GeoAccessorObs*

merge_metadata(*right*, *overlap='error'*)

Merge the metadata of an Obs object with metadata from another Obs object.

Parameters

- **right** (*dict*) – dictionary with the metadata of an Obs object.

- **overlap** (*str, optional*) – How to deal with overlapping metadata with different values. Options are:

  error : Raise a ValueError use_left : Use the metadata from self use_right : Use the given metadata

  Default is 'error'.

Raises
ValueError – if the metadata differs and overlap='error'.

Returns
new_metadata – metadata after merge.

Return type
dict

merge_observation(*right*, *overlap='error'*, *merge_metadata=True*)

Merge with another observation of the same type.

Parameters

- **right** (*hpd.observation.Obs*) – Observation object.
- **overlap** (*str, optional*) – How to deal with overlapping timeseries or metadata with different values. Options are:

  error : Raise a ValueError use_left : use the part of the overlapping timeseries from self use_right : use the part of the overlapping timeseries from right

  Default is 'error'.
- **merge_metadata** (*bool, optional*) – If True the metadata of the two objects are merged. If there are any differences the overlap parameter is used to determine which metadata is used. If merge_metadata is False, the metadata of self is always used for the merged observation. The default is True.

Raises

- **TypeError** – when the observation types are not the same.
- **ValueError** – when the time series have different values on the same date or different values for the same metadata.

Return type
Observation object.

plots

alias of *ObsPlots*

stats

alias of *StatsAccessorObs*

to_collection_dict(*include_meta=False*)

Get dictionary with registered attributes and their values of an Obs object.

This method can be used to create a dataframe from a collection of Obs objects.

Parameters
include_meta (*boolean, optional*) – include the meta dictionary in the collection dictionary, default is false

Returns
d – dictionary with Obs information

Return type
dictionary

class hydropandas.observation.**PrecipitationObs**(**args*, ***kwargs*)

Bases: *MeteoObs*

Class for precipitation timeseries.

Subclass of the MeteoObs class

classmethod **from_knmi**(*meteo_var='RH'*, *stn=None*, *fname=None*, *xy=None*, *start=None*, *end=None*, *fill_missing_obs=False*, *interval='daily'*, *use_api=True*, *raise_exceptions=True*, *startdate=None*, *enddate=None*)

Get a PrecipitationObs timeseries from the KNMI precipitation. The precipitation is the Daily precipitation amount (in 0.1 mm) (-1 for.

<0.05 mm).

There are 3 different ways to obtain precipitation data from the knmi:

1. Daily data from precipitation (neerslag) stations
2. Daily data from meteo stations
3. Hourly data from meteo stations

1. For daily data from a precipitation station (neerslagstation) meteo_var should be 'RD'. 2. For daily data from a meteo station meteo_var should be 'RH' and interval should be 'daily' (default). 3. For hourly data from a meteo station meteo_var should be 'RH' and interval should be 'hourly'.

More information about the differences between neerslag and meteo stations can be found in the hydropandas documentation -> 02_knmi_observations notebook.

Parameters

- **meteo_var** (*str, optional*) – meteo variable can be "RH" or "RD". "RD" if you want data from a precipitation station (neerslagstation). "RH" if you want data from a meteo station. The default is "RH".
- **stn** (*int, str or None, optional*) – measurement station e.g. 829. The default is None.
- **fname** (*str, path object, file-like object or None, optional*) – filename of a knmi file. The default is None.
- **xy** (*list, tuple or None, optional*) – RD coördinates of a location in the Netherlands. The station nearest to this location used. The Default is None.
- **start** (*str, datetime or None, optional*) – start date of observations. The default is None.
- **end** (*str, datetime or None, optional*) – end date of observations. The default is None.
- **fill_missing_obs** (*bool, optional*) – if True nan values in time series are filled with nearby time series. The default is False.
- **interval** (*str, optional*) – desired time interval for observations. Options are 'daily' and 'hourly'. The default is 'daily'.
- **use_api** (*bool, optional*) –

if True the api is used to obtain the data, API documentation is here:
https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script

if False a text file is downloaded into a temporary folder and the data is read from there. Default is True since the api is back online (July 2021).

- **raise_exceptions** (*bool, optional*) – if True you get errors when no data is returned. The default is False.

Return type
PrecipitationObs object with a precipitation time series and attributes

classmethod from_wow(*stn: str = None*, *xy: List[float] = None*, *start: Optional[Timestamp] = None*, *end: Optional[Timestamp] = None*)

Get a PrecipitationObs timeseries from a wow.knmi.nl station.

Parameters

- **stn** (*Optional[int, str], optional*) – station name
- **xy** (*Optinal[List[float]], optinial*) – longitude latitude of location [lon, lat] eg: [4.85, 51.95]
- **start** (*Optional[pd.Timestamp], optional*) – start date of observations, by default None
- **end** (*Optional[pd.Timestamp], optional*) – start date of observations, by default None

Return type
PrecipitationObs

class hydropandas.observation.**WaterQualityObs**(**args*, ***kwargs*)

Bases: *Obs*

Class for water quality ((grond)watersamenstelling) point observations.

Subclass of the Obs class

classmethod from_dino(*path*, ***kwargs*)

Read dino file with groundwater quality data.

Parameters

- **path** (*str*) – path of dino txt filename
- **kwargs** (*key-word arguments*) – these arguments are passed to hydropandas.io.dino.read_dino_groundwater_quality_txt

class hydropandas.observation.**WaterlvlObs**(**args*, ***kwargs*)

Bases: *Obs*

Class for water level point observations.

Subclass of the Obs class

classmethod from_dino(*path*, ***kwargs*)

Read a dino file with waterlvl data.

Parameters

- **path** (*str*) – path of dino csv filename

- **kwargs** (*key-word arguments*) – these arguments are passed to hydropandas.io.dino.read_dino_waterlvl_csv

classmethod from_waterinfo(*path*, ***kwargs*)

Read data from waterinfo csv-file or zip.

Parameters
path (*str*) – path to file (file can zip or csv)

Returns
df – WaterlvlObs object

Return type
WaterlvlObs

Raises
ValueError – if file contains data for more than one location

1.4.4 hydropandas.util module

Created on Wed Sep 12 12:15:42 2018.

@author: Artesia

class hydropandas.util.**ColoredFormatter**(**args*, *colors: Optional[Dict[str, str]] = None*, ***kwargs*)

Bases: Formatter

Colored log formatter.

Taken from https://gist.github.com/joshbode/58fac7ababc700f51e2a9ecdebe563ad

format(*record*) → str

Format the specified record as text.

hydropandas.util.**df2gdf**(*df*, *xcol='x'*, *ycol='y'*, *crs=28992*)

Create a GeoDataFrame from a DataFrame with xy points.

Parameters

- **df** (*pd.DataFrame*) – input dataframe
- **xcol** (*str, optional*) – column name with x values. The default is 'x'.
- **ycol** (*str, optional*) – column name with y values. The default is 'x'.
- **crs** (*int, optional*) – coordinate reference system, by default 28992 (RD new).

Returns
geodataframe

Return type
geopandas GeoDataFrame

hydropandas.util.**get_color_logger**(*level='INFO'*)

hydropandas.util.**get_files**(*file_or_dir*, *ext*, *unpackdir=None*, *force_unpack=False*, *preserve_datetime=False*)

Internal method to get list of files with specific extension from dirname.

Parameters

- **file_or_dir** (*str*) – file or path to data.

- **ext** (*str*) – extension of filenames to store in list.
- **unpackdir** (*str*) – directory to story unpacked zip file, only used in case of a zipfile.
- **force_unpack** (*bool, optional*) – force unzip, by default False.
- **preserve_datetime** (*bool, optional*) – preserve datetime of unzipped files, by default False. Used for checking whether data has changed.

hydropandas.util.**interpolate**(*xy: List[List[float]]*, *obsdf: DataFrame*, *obsloc: DataFrame*, *kernel: str = 'thin_plate_spline'*, *kernel2: str = 'linear'*, *epsilon: Optional[int] = None*) → DataFrame

Interpolation method using the Scipy radial basis function (RBF)

Parameters

- **xy** (*List[List[float]]*) – xy coordinates of locations of interest e.g. [[10,25], [5,25]]
- **obsdf** (*DataFrame*) – Dataframe containing the observation locations as columns and the observations at a measurement time in each row.
- **obsloc** (*DataFrame*) – Dataframe containing the observation locations coordinates with observation locations as index and columns ["x", "y"]
- **kernel** (*str, optional*) – Type of radial basis funtion, by default thin_plate_spline. Other options are linear, gaussian, inverse_quadratic, multiquadric, inverse_multiquadric, cubic or quintic.
- **kernel2** (*str, optional*) – Kernel in case there are not enough observations (3 or 6) for time step, by default linear. Other options are gaussian, inverse_quadratic, multiquadric, or inverse_multiquadric.
- **epsilon** (*Optional[int], optional*) – Shape parameter that scales the input to the RBF. If kernel is linear, thin_plate_spline, cubic, or quintic, this defaults to 1. Otherwise this must be specified.

Returns
DataFrame with locations of interest as columns and interpolated values at a measurement time in each row.

Return type
DataFrame

hydropandas.util.**oc_to_df**(*oc*, *col: Optional[str] = None*) → DataFrame

Convert an observation collection to a DataFrame where every column has one observation.

Parameters

- **oc** (*hydropandas ObsCollection*) – observation collection
- **col** (*Optional[str], optional*) – Name of a column in hte observation collection, by default None

Returns
description

Return type
DataFrame

hydropandas.util.**show_versions**()

Method to print the version of dependencies.

hydropandas.util.**unzip_file**(*src*, *dst*, *force=False*, *preserve_datetime=False*)

Unzip file.

Parameters

- **src** (*str*) – source zip file
- **dst** (*str*) – destination directory
- **force** (*boolean, optional*) – force unpack if dst already exists
- **preserve_datetime** (*boolean, optional*) – use date of the zipfile for the destination file

Returns

1 of True

Return type

int

1.5 Contribute

HydroPandas is an open source software project that depends on contributions from the community. Everyone is welcome, each small contribution is valuable, no matter if it is a fix of a typo in the documentation, bug report, an idea, or a question.

1.5.1 Questions, bug reports and feature requests

If you have question about the use of HydroPandas, feel free to ask them in the GitHub Discussions. Bugs and feature requests can be submitted via GitHub Issues.

1.5.2 Version control, Git, and GitHub

The code is hosted on GitHub. To contribute you will need to sign up for a free GitHub account. We use Git for version control to allow many people to work together on the project. If you have no experience with Git we recommend to install Github Desktop.

1.5.3 Contributing guidelines

Proposals for changes to the Hydropandas code base can be submitted via a pull request. You can find a pull request (or PR) tutorial in the GitHub's Help Docs..

There are roughly 6 steps for contributing to HydroPandas:

1. Fork the HydroPandas git repository
2. Create a development environment
3. Install HydroPandas dependencies
4. Make changes to code and add tests
5. Update the documentation
6. Submit a pull request

For pull request we use the following guidelines (similar to the geopandas guidelines):

- All existing tests should pass. Please make sure that the test suite passes, both locally and on GitHub Actions. Status on Github Actions will be visible on a pull request. To trigger a check, make a PR to your own fork.
- New functionality should include tests. Please write reasonable tests for your code and make sure that they pass on your pull request.
- Classes, methods, functions, etc. should have docstrings. The first line of a docstring should be a standalone summary. Parameters and return values should be documented explicitly.
- Follow PEP 8 when possible. We use Black and Flake8 to ensure a consistent code format throughout the project.
- We use isort to automatically sort imports.
- We encourage backward compatability between HydroPandas versions but do not ensure it (yet) because of the rapid changes to the code base.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

G

H

I

M

N

O

P

R

S

T

U

W