wwdata Documentation

Release 0.1.0

Chaim De Mulder

Nov 15, 2017

CONTENTS

Contents:

::include:: ../README.rst

CHAPTER

ONE

INSTALLATION

1.1 Stable release

To install wwdata, run this command in your terminal:

```
$ pip install wwdata
```

This is the preferred method to install wwdata, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.2 From sources

The sources for wwdata can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/cdemulde/wwdata
```

Or download the tarball:

```
$ curl  -OL https://github.com/cdemulde/wwdata/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

TWO

USAGE

To use wwdata in a project:

```
import wwdata as ww
```

As the package was developed with the help of Jupyter Notebook (version 4.4.1) and visualisation is an important part of it, it is highly recommended to make use of a Jupyter Notebook when using the package.

Once the package is imported, the suggested workflow is as follows:

1. Read data and convert it to a pandas DataFrame (see [the pandas documentation](https://pandas.pydata.org) for more information); format it in the way you like.
2. Create an object of any of the three classes in the wwdata package, e.g. the OnlineSensorBased class:

```
data = ww.OnlineSensorBased(dataframe,timedata_column="time",data_type="WWTP",
↪experiment_tag="Data 2017")
```

3. Explore and format the data (convert to datetime, make absolute time index, make some plots...), e.g.:

```
data.to_datetime(time_column="time",time_format="%dd-%mm-%yy")
data.get_avg()
```

4. Tag non-valid data points. The way to do this depends on the data you are working with, but the general approach would be:

```
data.tag_nan()
data.moving_slope_filter(xdata="time",data_name="series1",cutoff=3,arange=['1/1/
↪2017','1/2/2017'])
```

i.e. to simply apply any of the filtering functions to the class object.

5. Apply any other functionalities to the object. In the below example, this is the filling of the gaps introduced by filtering data in the previous step (for details on the meaning of the arguments, please refer to the documentation provided within the source code):

```
data.fill_missing_interpolation("series1",range_=12,arange=['1/1/2017','1/2/2017
↪'])
data.fill_missing_model("series1","model_data_series",arange=['1/1/2017','1/2/2017
↪'])
```

CHAPTER

THREE

WWDATA

3.1 wwdata package

3.1.1 Submodules

3.1.2 wwdata.Class_HydroData module

Class_HydroData provides functionalities for handling data obtained in the context of (waste)water treatment.

class wwdata.Class_HydroData.**HydroData**(*data*, *timedata_column='index'*, *data_type='WWTP'*, *experiment_tag='No tag given'*, *time_unit=None*, *units=[]*)

Bases: object

Methods

absolute_to_relative(*time_data='index'*, *unit='d'*, *inplace=True*, *save_abs=True*, *decimals=5*)

converts a pandas series with datetime timevalues to relative timevalues in the given unit, starting from 0

Parameters **time_data** : str

name of the column containing the time data. If this is the index column, just give 'index' (also default)

unit : str

unit to which to convert the time values (sec, min, hr or d)

add_to_meta_valid(*column_names*)

Adds (a) column(s) with the given column_name(s) to the self.meta_filled DataFrame, where all tags are set to 'original'. This makes sure that also data that already is very reliable can be used further down the process (e.g. filling etc.)

Parameters **column_names** : array

array containing the names of the columns to add to the meta_valied dataframe

calc_daily_profile(*column_name*, *arange*, *quantile=0.9*, *plot=False*, *plot_method='quantile'*, *clear=False*, *only_checked=False*)

Calculates a typical daily profile based on data from the indicated consecutive days. Also saves this average day, along with standard deviation and lower and upper percentiles as given in the arguments. Plotting is possible.

Parameters **column_name** : str

name of the column containing the data to calculate an average day for

arange : 2-element array of ints

contains the beginning and end day of the period to use for average day calculation

quantile : float between 0 and 1

value to use for the calculation of the quantiles

plot : bool

plot or not

plot_method : str

method to use for plotting. Available: "quantile" or "stdev"

clear : bool

wether or not to clear the key in the self.daily_profile dictionary that is already present

Returns

——

None; creates a dictionary self.daily_profile containing information

on the average day as calculated.

calc_ratio(*data_1*, *data_2*, *arange*, *only_checked=False*)

Given two datasets or -columns, calculates the average ratio between the first and second dataset, within the given range. Also the standard deviation on this is calculated

Parameters **data_1** : str

name of the data column containing the data to be in the numerator of the ratio calculation

data_2 : str

name of the data column containing the data to be in the denominator of the ratio calculation

arange : array of two values

the range within which the ratio needs to be calculated

only_checked : bool

if 'True', filtered values are excluded; default to 'False'

Returns The average ratio of the first data column over the second one within

the given range and including the standard deviation

calc_slopes(*xdata*, *ydata*, *time_unit=None*, *slope_range=None*)

Calculates slopes for given xdata and data_name; if a time unit is given as an argument, the time values (xdata) will first be converted to this unit, which will then be used to calculate the slopes with.

Parameters **xdata** : str

name of the column containing the xdata for slope calculation (e.g. time). If 'index', the index is used as xdata. If datetime objects, a time_unit is expected to calculate the slopes.

data_name : str

name of the column containing the data_name for slope calculation

time_unit : str

time unit to be used for the slope calculation (in case this is based on time); if None, slopes are simply calculated based on the values given !! This value has no impact if the xdata column is the index and is not a datetime type. If that is the case, it is assumed that the user knows the unit of the xdata !!

Returns pd.Series

pandas Series object containing the slopes calculated for the chosen variable

compare_ratio(*data_1*, *data_2*, *arange*, *only_checked=False*)

Compares the average ratios of two datasets in multiple different ranges and returns the most reliable one, based on the standard deviation on the ratio values

Parameters **data_1** : str

name of the data column containing the data to be in the numerator of the ratio calculation

data_2 : str

name of the data column containing the data to be in the denominator of the ratio calculation

arange : int

the range (in days) for which the ratios need to be calculated and compared

only_checked : bool

if 'True', filtered values are excluded; default to 'False'

Returns The average ratio within the range that has been found to be the most

reliable one

drop_index_duplicates()

drop rows with a duplicate index, ASSUMING THEY HAVE THE SAME DATA IN THEM!! Also updates the meta_valid dataframe

fill_index(*arange*, *index_type='float'*)

function to fill in missing index values

get_avg(*name=None*, *only_checked=True*)

Gets the averages of all or certain columns in a dataframe

Parameters **name** : arary of str

name(s) of the column(s) containing the data to be averaged; defaults to ['none'] and will calculate average for every column

Returns pd.DataFrame :

pandas dataframe, containing the average slopes of all or certain columns

get_correlation(*data_1*, *data_2*, *arange*, *zero_intercept=False*, *only_checked=False*, *plot=False*)
Calculates the linear regression coefficients that relate data_1 to data_2

Parameters **data_1 and data_2** : str

names of the data columns containing the data between which the correlation will be calculated.

arange : array

array containing the beginning and end value between which the correlation needs to be calculated

zero_intercept : bool

indicates whether or not to assume a zero-intercept

only_checked: bool

if 'True', filtered values are excluded from calculation and plotting; default to 'False' if a value in one column is filtered, the corresponding value in the second column also gets excluded!

Returns the linear regression coefficients of the correlation, as well as the

r-squared -value

get_highs(*data_name*, *bound_value*, *arange*, *method='percentile'*, *plot=False*)
creates a dataframe with tags indicating what indices have data-values higher than a certain value; example: the definition/tagging of rain events.

data_name [str] name of the column to execute the function on

bound_value [float] the boundary value above which points will be tagged

arange [array of two values] the range within which high values need to be tagged

method: str (value or percentile) when percentile, the bound value is a given percentile above which data points will be tagged, when value, bound_values is used directly to tag data points.

get_std(*name=None*, *only_checked=True*)
Gets the standard deviations of all or certain columns in a dataframe

Parameters **dataframe** : pd.DataFrame

dataframe containing the columns to calculate the standard deviation for

name : arary of str

name(s) of the column(s) containing the data to calculate standard deviation for; defaults to ['none'] and will calculate standard deviation for every column

plot : bool

if True, plots the calculated standard deviations, defaults to False

Returns pd.DataFrame :

pandas dataframe, containing the average slopes of all or certain columns

head(*n=5*)
CONFIRMED piping pandas head function

index()
piping pandas index function

moving_average_filter(*data_name*, *window*, *cutoff_frac*, *arange*, *clear=False*, *inplace=False*, *log_file=None*, *plot=False*, *final=False*)
Filters out the peaks/outliers in a dataset by comparing its values to a smoothened representation of the dataset (Moving Average Filtering). The filtered values are replaced by NaN values.

Parameters **data_name** : str

name of the column containing the data that needs to be filtered

window : int

the number of values from the dataset that are used to take the average at the current point.

cutoff_frac: float

the cutoff value (in fraction 0-1) to compare the data and smoothened data: a deviation higher than a certain percentage drops the data- point.

arange : array of two values

the range within which the moving average filter needs to be applied

clear : bool

if True, the tags added to datapoints before will be removed and put back to 'original'.

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned)

log_file : str

string containing the directory to a log file to be written out when using this function

plot : bool

if true, a plot is made, comparing the original dataset with the new, filtered dataset

final : bool

if true, the values are actually replaced with nan values (either inplace or in a new hp object)

Returns HydroData object (if inplace=False)

the dataframe from which the double values of 'data' are removed

None (if inplace=True)

moving_slope_filter(*xdata*, *data_name*, *cutoff*, *arange*, *time_unit=None*, *clear=False*, *inplace=False*, *log_file=None*, *plot=False*, *final=False*)
CONFIRMED Filters out datapoints based on the difference between the slope in one point and the next (sudden changes like noise get filtered out), based on a given cut off value. Replaces the dropped values with NaN values.

Parameters **xdata** : str

name of the column containing the xdata for slope calculation (e.g. time). If 'index', the index is used as xdata. If datetime objects, a time_unit is expected to calculate the slopes.

data_name : str

name of the column containing the data that needs to be filtered

cutoff: int

the cutoff value to compare the slopes with to apply the filtering.

arange : array of two values

the range within which the moving slope filter needs to be applied

time_unit : str

time unit to be used for the slope calculation (in case this is based on time); if None, slopes are calculated based on the values given

clear : bool

if True, the tags added to datapoints before will be removed and put back to 'original'.

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned)

log_file : str

string containing the directory to a log file to be written out when using this function

plot : bool

if true, a plot is made, comparing the original dataset with the new, filtered dataset

final : bool

if true, the values are actually replaced with nan values (either inplace or in a new hp object)

Returns HydroData object (if inplace=False)

the dataframe from which the double values of 'data' are removed

None (if inplace=True)

plot_analysed(*data_name*, *time_range='default'*, *only_checked=False*)

plots the values and their types (original, filtered, filled) of a given column in the given time range.

Parameters **data_name** : str

name of the column containing the data to plot

time_range : array of two values

the range within which the values are plotted; default is all

only_checked : bool

if 'True', filtered values are excluded; default to 'False'

Returns Plot

replace(*to_replace*, *value*, *inplace=False*)

CONFIRMED piping pandas replace function

savgol(*data_name*, *window=55*, *polyorder=2*, *plot=False*, *inplace=False*)

Uses the scipy.signal Savitzky-Golay filter to smoothen the data of a column; The values are either replaced or a new dataframe is returned.

Parameters **data_name** : str

name of the column containing the data that needs to be filtered

window : int

the length of the filter window; default to 55

polyorder : int

The order of the polynomial used to fit the samples. polyorder must be less than window. default to 1

plot : bool

if true, a plot is made, comparing the original dataset with the new, filtered dataset

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned)

Returns HydroData object (if inplace=False)

None (if inplace=True)

set_index(*keys*, *key_is_time=False*, *drop=True*, *inplace=False*, *verify_integrity=False*, *save_prev_index=True*)

piping pandas set_index function, including the option to define the new index as being the time data key_is_time : bool

when true, the new index will we known as the time data from here on

(other arguments cfr pd.set_index)

set_tag(*tag*)

CONFIRMED

set_time_unit(*unit*)

set_units(*units*)

simple_moving_average(*arange*, *window*, *data_name=None*, *inplace=False*, *plot=True*)

CONFIRMED Calculate the Simple Moving Average of a dataseries from a dataframe, using a window within which the datavalues are averaged.

Parameters **arange** : array of two values

the range within which the moving average needs to be calculated

window : int

the number of values from the dataset that are used to take the average at the current point. Defaults to 10

data_name : str or array of str

name of the column(s) containing the data that needs to be smoothened. If None, smoothened data is computed for the whole dataframe. Defaults to None

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned)

plot : bool

if True, a plot is given for comparison between original and smooth data

Returns HydroData (or subclass) object

either a new object (inplace=False) or an adjusted object, con- taining the smoothened data values

tag_doubles(*data_name*, *bound*, *arange=None*, *clear=False*, *inplace=False*, *log_file=None*, *plot=False*, *final=False*)

tags double values that subsequently occur in a measurement series. This is relevant in case a sensor has failed and produces a constant signal. A band is provided within which the signal can vary and still be filtered out

Parameters **data_name** : str

column name of the column from which double values will be sought

bound : float

boundary value of the band to use. When the difference between a point and the next one is smaller then the bound value, the latter datapoint is tagged as 'filtered'.

arange : array of two values

the range within which double values need to be tagged

clear : bool

if True, the tags added to datapoints before will be removed and put back to 'original'.

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned). (This argument only comes into play when the 'final' argument is True)

log_file : str

string containing the directory to a log file to be written out when using this function

plot : bool

whether or not to make a plot of the newly tagged data points

final : bool

if true, the values are actually replaced with nan values (either inplace or in a new hp object)

Returns HydroData object (if inplace=False)

the dataframe from which the double values of 'data' are removed or replaced

None (if inplace=True)

tag_extremes(*data_name*, *arange=None*, *limit=0*, *method='below'*, *clear=False*, *plot=False*)

Tags values above or below a given limit.

Parameters **data_name** : str

name of the column containing the data to be tagged

arange : array of two values

the range within which extreme values need to be tagged

limit : int/float

limit below or above which values need to be tagged

method : 'below' or 'above'

below tags all the values below the given limit, above tags the values above the limit

clear : bool

if True, the tags added before will be removed and put back to 'original'.

plot : bool

whether or not to make a plot of the newly tagged data points

Returns None;

tag_nan (*data_name*, *arange=None*, *clear=False*)
adds a tag 'filtered' in self.meta_valid for every NaN value in the given column

Parameters **data_name** : str

column name of the column to apply the function to

arange : array of two values

the range within which nan values need to be tagged

clear : bool

when true, resets the tags in meta_valid for the data in column data_name

Returns None

tail (*n=5*)
CONFIRMED piping pandas tail function

to_datetime (*time_column='index'*, *time_format='%dd-%mm-%yy'*, *unit='D'*)
CONFIRMED flexibly piping pandas to_datetime function

to_float (*columns='all'*)
convert values in given columns to float values

columns [array of strings] column names of the columns where values need to be converted to floats

write (*filename*, *filepath='/Users/chaimdemulder/Documents/Work/github/wwdata/docs'*, *method='all'*)

Parameters **filepath** : str

the path the output file should be saved to

filename : str

the name of the output file

method : str (all,filtered,filled)

depending on the method choice, different values will be written out: all values, only the filtered values or the filled values

for_WEST : bool

include_units : bool

Returns None; write an output file

wwdata.Class_HydroData.**total_seconds** (*timedelta_value*)

3.1.3 wwdata.Class_LabExperimBased module

Class_LabExperimBased provides functionalities for data handling of data obtained in lab experiments in the field of (waste)water treatment. Copyright (C) 2016 Chaim De Mulder

class wwdata.Class_LabExperimBased.**LabExperimBased**(*data*, *timedata_column='index'*, *data_type='NAT'*, *experiment_tag='No tag given'*, *time_unit=None*)

Bases: *wwdata.Class_HydroData.HydroData*

Superclass for a HydroData object, expanding the functionalities with specific functions for data gathered is lab experiments.

Attributes

(see HydroData docstring)	

Methods

add_conc(*column_name*, *x*, *y*, *new_name='default'*)

calculates the concentration values of the given column and adds them as a new column to the DataFrame.

Parameters **column_name** : str

column with values

x : int

...

y : int

...

new_name : str

name of the new column, default to 'column_name + mg/L'

calc_slope(*columns*, *time_column='h'*)

calculates the slope of the selected columns

Parameters **columns** : array of strings

columns to calculate the slope for

time_column : str

time used for calculation; default to 'h'

check_ph(*ph_column='pH'*, *thresh=0.4*)

gives the maximal change in pH

Parameters **ph_column** : str

column with pH-values, default to 'pH'

threshold : int

threshold value for warning, default to '0.4'

hours(*time_column='index'*)
calculates the hours from the relative values

Parameters **time_column** : string

column containing the relative time values; default to index

in_out(*columns*)
(start_values-end_values)

Parameters **columns** : array of strings

plot(*columns*, *time_column='index'*)
calculates the slope of the selected columns

Parameters **columns** : array of strings

columns to plot

time_column : str

time used for calculation; default to 'h'

removal(*columns*)
total removal of nitrogen (1-(end_values/start_values))

Parameters **columns** : array of strings

3.1.4 wwdata.Class_LabSensorBased module

Class_LabSensorBased provides functionalities for data handling of data obtained in lab experiments with online sensors in the field of (waste)water treatment. Copyright (C) 2016 Chaim De Mulder

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see http://www.gnu.org/licenses/.

class wwdata.Class_LabSensorBased.**LabSensorBased**(*data*, *experiment_tag='None'*)
Bases: *wwdata.Class_HydroData.HydroData*

Superclass for a HydroData object, expanding the functionalities with specific functions for data gathered is lab experiments

Attributes

(see HydroData docstring)	

Methods

drop_peaks(*data_name*, *cutoff*, *inplace=True*, *log_file=None*)
Filters out the peaks larger than a cut-off value in a dataseries

Parameters **data_name** : str

the name of the column to use for the removal of peak values

cutoff : int

cut off value to use for the removing of peaks; values with an absolute value larger than this cut off will be removed from the data

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned)

log_file : str

string containing the directory to a log file to be written out when using this function

Returns LabSensorBased object (if inplace=False)

the dataframe from which the double values of 'data' are removed

None (if inplace=True)

3.1.5 wwdata.Class_OnlineSensorBased module

Class_OnlineSensorBased provides functionalities for data handling of data obtained with online sensors in the field of (waste)water treatment. Copyright (C) 2016 Chaim De Mulder

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see http://www.gnu.org/licenses/.

class wwdata.Class_OnlineSensorBased.**OnlineSensorBased**(*data*, *time-data_column='index'*, *data_type='WWTP'*, *experiment_tag='No tag given'*, *time_unit=None*)
Bases: *wwdata.Class_HydroData.HydroData*

Superclass for a HydroData object, expanding the functionalities with specific functions for data gathered at full scale by continous measurements

Attributes

(see HydroData docstring)	

Methods

add_to_filled(*column_names*)
column_names : array

calc_daily_average(*column_name*, *arange*, *plot=False*)
calculates the daily average of values in the given column and returns them as a 2D-array, containing the days and the average values on the respective days. Plotting is possible.

Parameters **column_name** : str

name of the column containing the data to calculate the average values for

arange : array of two values

the range within which daily averages need to be calculated

plot : bool

plot or not

Returns pd.Dataframe :

pandas dataframe, containing the daily means with standard deviations for the selected column

calc_total_proportional(*Q_tot*, *Q*, *conc*, *new_name='new'*, *unit='mg/l'*, *filled=False*)
CONFIRMED Calculates the total concentration of an incoming flow, based on the given total flow and the separate incoming flows and concentrations

Parameters **Q_tot** : str

name of the column containing the total flow

Q : array of str

names of the columns containing the separate flows

conc : array of str

names of the columns containing the separate concentration values

new_name : str

name of the column to be added

filled : bool

if true, use self.filled to calculate proportions from

!!Order of columns in Q and conc must match!!

Returns None;

creates a hydropy object with added column for the proportional concentration

check_filling_error(*nr_iterations*, *data_name*, *filling_function*, *test_data_range*, *nr_small_gaps=0*, *max_size_small_gaps=0*, *nr_large_gaps=0*, *max_size_large_gaps=0*, ***options*)
Uses the _calculate_filling_error function (refer to that docstring for more specific info) to calculate the error on the data points that are filled with a certain algorithm. Because _calculate_filling_error inserts random gaps, results differ every time it is used. Check_filling_error averages this out.

!! Important !! When checking for the error on data filling, a period (arange argument) with mostly reliable data should be used. If for example large gaps are already present in the given data, this will heavily influence the returned error, as filled values will be compared with the values from the data gap.

Parameters **nr_iterations** : int

The number of iterations to run for the calculation of the imputation error

data_name : string

name of the column containing the data the filling reliability needs to be checked for.

filling function : str, wdata filling function

the name of the filling function to be tested for reliability

test_data_range : array of two values

an array containing the start and end point of the test data to be used. IMPORTANT: for testing filling with correlation, this range needs to include the range for correlation calculation and the filling range.

nr_small_gaps / nr_large_gaps: int

the number of small/large gaps to create in the dataset for testing

max_size_small_gaps / max_size_large_gaps: int

the maximum size of the gaps inserted in the data, expressed in data points

****options:**

Arguments for the filling function; refer to the relevant filling function to know what arguments to give

Returns None; adds the average filling error the self.filling_error dataframe

drop_index_duplicates()

drop rows with a duplicate index, ASSUMING THEY HAVE THE SAME DATA IN THEM!! Also updates the meta_valid, meta_filled and filled dataframes

fill_missing_correlation(*to_fill*, *to_use*, *arange*, *corr_range*, *zero_intercept=False*, *filtered_only=True*, *plot=False*, *clear=False*)

Fills the missing values in a dataset (to_fill), based on the correlation this data shows when comparing to other data (to_use). This happens within the range given by arange.

Parameters **to_fill** : str

name of the column with data to fill

to_use : str

name of the column to use, in combination with the given ratio, to fill in some of the missing data

arange : array of two values

the range within which missing/filtered values need to be replaced

corr_range : array of two values

the range to use for the calculation of the correlation

filtered_only : boolean

if True, fills only the datapoints labeled as filtered. If False, fills/replaces all datapoints in the given range

plot : bool

whether or not to plot the new dataset

clear : bool

whether or not to clear the previoulsy filled values and start from the self.meta_valid dataset again for this particular dataseries.

Returns None;

creates/updates self.filled, containing the adjusted dataset and updates

meta_filled with the correct labels.

fill_missing_daybefore(*to_fill, arange, range_to_replace=[1, 4], filtered_only=True, plot=False, clear=False*)

Fills the missing values in a dataset (to_fill), based on the data values from the day before the range starts. These data values are based on the self.filled dataset and therefor can contain filled datapoints as well. This happens within the range given by arange. !! IMPORTANT !! This function will not work on datasets with non-equidistant data points!

Parameters **to_fill** : str

name of the column with data to fill

arange : array of two values

the range within which missing/filtered values need to be replaced

range_to_replace : array of two int/float values

the minimum and maximum amount of time (i.e. min and max size of gaps in data) where missing datapoints can be replaced using this function, i.e. using values of the last day before measurements went bad.

filtered_only : boolean

if True, fills only the datapoints labeled as filtered. If False, fills/replaces all datapoints in the given range

plot : bool

whether or not to plot the new dataset

clear : bool

whether or not to clear the previoulsy filled values and start from the self.meta_valid dataset again for this particular dataseries.

Returns None;

creates/updates self.filled, containing the adjusted dataset and updates

meta_filled with the correct labels.

fill_missing_interpolation(*to_fill, range_, arange, method='index', plot=False, clear=False*)

Fills the missing values in a dataset (to_fill), based specified interpolation algorithm (method). This happens only if the number of consecutive missing values is smaller than **range_**.

Parameters **to_fill** : str

name of the column containing the data to be filled

range_ : int

the maximum range that the absence of values can be to still allow interpolation to fill in values

arange : array of two values

the range within which missing/filtered values need to be replaced

method : str

interpolation method to be used by the .interpolate function. See pandas docstrings for more info

plot : bool

whether or not to plot the new dataset

clear : bool

whether or not to clear the previoulsy filled values and start from the self.meta_valid dataset again for this particular dataseries.

Returns None;

creates/updates self.filled, containing the adjusted dataset and updates

meta_filled with the correct labels.

fill_missing_model(*to_fill*, *to_use*, *arange*, *filtered_only=True*, *unit='d'*, *plot=False*, *clear=False*)

Fills the missing values in a dataset (to_fill), based on the modeled values given in to_use. This happens within the range given by arange.

Parameters **to_fill** : str

name of the column with data to fill

to_use : pd.Series

pandas series containing the modeled data with which the filtered data can be replaced

arange : array of two values

the range within which missing/filtered values need to be replaced

filtered_only : boolean

if True, fills only the datapoints labeled as filtered. If False, fills/replaces all datapoints in the given range

unit : str

the unit in which the modeled values are given; datetime values will be converted to values with that unit. Possible: sec, min, hr, d

plot : bool

whether or not to plot the new dataset

clear : bool

whether or not to clear the previoulsy filled values and start from the self.meta_valid dataset again for this particular dataseries.

Returns None;

creates/updates self.filled, containing the adjusted dataset and updates

meta_filled with the correct labels.

fill_missing_ratio(*to_fill*, *to_use*, *ratio*, *arange*, *filtered_only=True*, *plot=False*, *clear=False*)

Fills the missing values in a dataset (to_fill), based on the ratio this data shows when comparing to other data (to_use). This happens within the range given by arange.

Parameters **to_fill** : str

name of the column with data to fill

to_use : str

name of the column to use, in combination with the given ratio, to fill in some of the missing data

ratio : float

ratio to multiply the to_use data with to obtain data for filling in in the to_fill data column

arange : array of two values

the range within which missing/filtered values need to be replaced

filtered_only : boolean

if True, fills only the datapoints labeled as filtered. If False, fills/replaces all datapoints in the given range

plot : bool

whether or not to plot the new dataset

clear : bool

whether or not to clear the previoulsy filled values and start from the self.meta_valid dataset again for this particular dataseries.

Returns None;

creates/updates self.filled, containing the adjusted dataset and updates

meta_filled with the correct labels.

fill_missing_standard(*to_fill*, *arange*, *filtered_only=True*, *plot=False*, *clear=False*)

Fills the missing values in a dataset (to_fill), based on the average daily profile calculated by calc_daily_profile(). This happens within the range given by arange.

Parameters **to_fill** : str

name of the column with data to fill

arange : array of two values

the range within which missing/filtered values need to be replaced

filtered_only : boolean

if True, fills only the datapoints labeled as filtered. If False, fills/replaces all datapoints in the given range

plot : bool

whether or not to plot the new dataset

clear : bool

whether or not to clear the previoulsy filled values and start from the self.meta_valid dataset again for this particular dataseries.

Returns None;

creates/updates self.filled, containing the adjusted dataset and updates

meta_filled with the correct labels.

wwdata.Class_OnlineSensorBased.**absolute_to_relative**(*series*, *start_date*, *unit='d'*, *decimals=5*)

converts a pandas series with datetime timevalues to relative timevalues in the given unit, starting from start_date

Parameters **series** : pd.Series

series of datetime of comparable values

unit : str

unit to which to convert the time values (sec, min, hr or d)

wwdata.Class_OnlineSensorBased.**drop_peaks**(*self*, *data_name*, *cutoff*, *inplace=True*, *log_file=None*)

Filters out the peaks larger than a cut-off value in a dataseries

Parameters **data_name** : str

the name of the column to use for the removal of peak values

cutoff : int

cut off value to use for the removing of peaks; values with an absolute value larger than this cut off will be removed from the data

inplace : bool

indicates whether a new dataframe is created and returned or whether the operations are executed on the existing dataframe (nothing is returned)

log_file : str

string containing the directory to a log file to be written out when using this function

Returns LabSensorBased object (if inplace=False)

the dataframe from which the double values of 'data' are removed

None (if inplace=True)

wwdata.Class_OnlineSensorBased.**find_nearest_time**(*value*, *df*, *column*)

value : float

wwdata.Class_OnlineSensorBased.**go_WEST**(*raw_data*, *time_data*, *WEST_name_conversion*)

Saves a WEST compatible file (influent or other inputs)

Parameters **raw_data: str or pd DataFrame**

time_data:

WEST_name_conversion: pd DataFrame with column names: WEST, units and RAW

dataframe containing three columns: the column names for the WEST-compatible file, the units to appear in the WEST-compatible file and the column names of the raw data file.

wwdata.Class_OnlineSensorBased.**total_seconds**(*timedelta_value*)

wwdata.Class_OnlineSensorBased.**vlookup_day**(*value*, *df*, *column*)

3.1.6 wwdata.data_reading_functions module

data_reading_functions provides functionalities for data reading in the context of the wwdata package. Copyright (C) 2016 Chaim De Mulder

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see http://www.gnu.org/licenses/.

wwdata.data_reading_functions.**find_and_replace**(*path*, *ext*, *replace*)

Finds the files with a certain extension in a directory and applies a find- replace action to those files. Removes the old files and produces files with a prefix stating the replacing value.

Parameters **path** : str

the path name of the directory to apply the function to

ext : str

the extension of the files to be searched (excel, text or csv)

replace : array of str

the first value of replace is the string to be replaced by the second value of replace.

wwdata.data_reading_functions.**join_files**(*path*, *files*, *ext='text'*, *sep=','*, *comment='#'*, *encoding='utf8'*, *decimal='.'*)

Reads all files in a given directory, joins them and returns one pd.dataframe

Parameters **path** : str

path to the folder that contains the files to be joined

files : list

list of files to be joined, must be the same extension

ext : str

extention of the files to read; possible: excel, text, csv

sep : str

the separating element (e.g. , or) necessary when reading csv-files

comment : str

comment symbol used in the files

sort : array of bool and str

if first element is true, apply the sort function to sort the data based on the tags in the column mentioned in the second element of the sort array

Returns pd.dataframe:

pandas dataframe containin concatenated files in the given directory

wwdata.data_reading_functions.**list_files**(*path*, *ext*)

wwdata.data_reading_functions.**read_mat**(*path*)

Reads in .mat datafiles and returns them as pd.DataFrame http://stackoverflow.com/questions/24762122/read-matlab-data-file-into-python-need-to-export-to-csv

wwdata.data_reading_functions.**remove_empty_lines**(*path*, *ext*)

wwdata.data_reading_functions.**sort_data**(*data, based_on, reset_index=[False, 'new_index_name'], convert_to_timestamp=[True, 'time_name', '%d.%m.%Y %H:%M:%S']*)

Sorts a dataset based on values in one of the columns and splits them in different dataframes, returned in the form of one dictionary

Parameters **data** : pd.dataframe

the dataframe containing the data that needs to be sorted

based_on : str

the name of the column that contains the names or values the sorting should be based on

reset_index : [bool,str]

array indicating if the index of the sorted datasets should be reset to a new one; if first element is true, the second element is the title of the column to use as new index; default: False

Returns dict :

A dictionary of pandas dataframes with as labels those acquired from the based_on column

wwdata.data_reading_functions.**write_to_WEST**(*df, file_normal, file_west, units, filepath='/Users/chaimdemulder/Documents/Work/github/wwdata/do fillna=True*)

writes a text-file that is compatible with WEST. Adds the units as they are given in the 'units' argument.

Parameters **df** : pd.DataFrame

the dataframe to write to WEST

file_normal : str

name of the original file to write, not yet compatible with WEST

file_west : str

name of the file that needs to be WEST compatible

units : array of strings

array containing the units for the respective columns in df

filepath : str

directory to save the files in; defaults to the current one

fillna : bool

when True, replaces nan values with 0 values (this might avoid WEST problems later one).

Returns None; writes files

3.1.7 wwdata.time_conversion_functions module

time_conversion_functions provides functionalities for converting certain types of time data to other types, in the context of the wwdata package. Copyright (C) 2016 Chaim De Mulder

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see http://www.gnu.org/licenses/.

wwdata.time_conversion_functions.**get_absolute_time**(*value*, *date_type='WindowsDateSystem'*)

Converts a coded time to the absolute date at which the experiment was conducted.

Parameters **value** : int or float

a code for a certain point in time

date_type : str

type of coding used for the time point, probably depending on the software which was used to acquire the data, e.g. the Windows Date System (here as default, see also: http://excelsemipro.com/2010/08/date-and-time-calculation-in-excel/)

Returns datetime.datetime

python datetime object

wwdata.time_conversion_functions.**make_datetime**(*array*)

array with elements 0: year (yy) 1: month (mm) 2: day in month (dd) 3: hour (h or hh) 4: minutes (minmin)

wwdata.time_conversion_functions.**make_month_day_array**()

makes a dataframe containing two columns, one with the number of the month, one with the day of the month. Useful in creating datetime objects based on e.g. date serial numbers from the Window Date System (http://excelsemipro.com/2010/08/date-and-time-calculation-in-excel/)

Returns pd.DataFrame :

dataframe with number of the month and number of the day of the month for a whole year

wwdata.time_conversion_functions.**timedelta_to_abs**(*timedelta*, *unit='d'*)

timedelta : array of timedelta values

wwdata.time_conversion_functions.**to_datetime_singlevalue**(*time*)

In case timedata is in a string format, to convert it to a datetime object, it needs to be in the right format, e.g. dd-mm-yyyy hh:mm:ss (so two of each) This function takes care of that, to a certain extent.

wwdata.time_conversion_functions.**to_days**(*timedelta*)

timedelta : timedelta value

wwdata.time_conversion_functions.**to_hours**(*timedelta*)

timedelta : timedelta value

wwdata.time_conversion_functions.**to_minutes**(*timedelta*)

timedelta : timedelta value

wwdata.time_conversion_functions.**to_seconds**(*timedelta*)

timedelta : timedelta value

3.1.8 Module contents

Top-level package for wwdata.

CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/cdemulde/wwdata/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

wwdata could always use more documentation, whether as part of the official wwdata docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/cdemulde/wwdata/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *wwdata* for local development.

1. Fork the *wwdata* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/wwdata.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv wwdata
$ cd wwdata/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 wwdata tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/cdemulde/wwdata/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_wwdata
```

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- [Chaim De Mulder](mailto:demulderchaim@gmail.com)

5.2 Contributors

None yet. All contributions are welcome!

5.3 Current funder

[Waterboard De Dommel](https://www.dommel.nl/index.html) (The Netherlands)

CHAPTER

SIX

HISTORY

6.1 0.1.0 (2017-09-01)

- First release on PyPI.

The wwdata (wastewater data) package is meant to make data analysis, validation and filling of data gaps more streamlined. It contains code to do all this, while also providing simple visualisations of the whole procedure.

The package was (and is) developed in the framework of PhD research, involving the modelling of a full scale wastewater treatment plant (WWTP). Online measurements at the plant are available, but as with all data, is not perfect and therefor needs validation. The gap filling originated from the need to have high-frequency influent data available to run the WWTP model with.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX

V

W