hydrostats Documentation

Release 0.78

Wade Roberts

Aug 25, 2020

Contents

Hydrostats is a library of tools and functions for users working with time series data, with some tools specific to the field of hydrology (hence **Hydro**stats). All of the tools contained in Hydrostats are built using a few different python libraries including numpy, scipy, pandas, matplotlib, and numba. It is meant to provide a high-level interface for users to be able to perform common tasks regarding time series analysis.

Hydrostats contains tools for for preprocessing data, visualizing data, calculating error metrics on observed and predicted time series, and forecast validation. It contains over 70 error metrics, with many metrics specific to the field of hydrology.

See the examples folder in this repository for a Jupyter notebook highlighting some of the main features of Hydrostats.

CHAPTER 1

Contents

1.1 Installation

Hydrostats is freely available on the Python Package index repository (PyPI). It can be installed with the following command using either pip, virtualenv, or Anaconda:

```
pip install hydrostats
```

The extension is also available through conda package management system. It can be installed with:

```
conda install -c conda-forge hydrostats
```

When installing hydrostats on Mac OSX operating system, you may get the following error when trying to run python scripts using hydrostats:

```
**RuntimeError**: Python is not installed as a framework. The Mac OS X backend will␣
↪not be able
to function correctly if Python is not installed as a framework. See the Python␣
↪documentation for
more information on installing Python as a framework on Mac OS X. Please either␣
↪reinstall Python
as a framework, or try one of the other backends.
```

If this happens, you will need to manually change the backend for use in your IDE. This can be done by running the following commands in the terminal:

```
touch ~/.matplotlib/matplotlibrc  # Create a file called matplotlibrc
```

Then add the following snippet of text to the file you created using vim, nano, or the text editor of your choice:

```
backend: TkAgg  # or whatever backend you would like to use
```

Then save and close the document.

For more information about matplotlib backends, read here

1.2 Preprocessing (hydrostats.data)

1.2.1 hydrostats.data Module

The data module contains tools for preprocessing data. It allows users to merge timeseries, compute daily and monthly summary statistics, and get seasonal periods of a time series.

Functions

julian_to_gregorian(dataframe[, frequency, ...])	Converts the index of the merged dataframe from julian float values to gregorian datetime values.
merge_data([sim_fpath, obs_fpath, sim_df, ...])	Merges two dataframes or csv files, depending on the input.
daily_average(df[, rolling])	Calculates daily seasonal averages of the timeseries data in a DataFrame
daily_std_error(merged_data)	Calculates daily seasonal standard error of the time-series data in a DataFrame
daily_std_dev(merged_data)	Calculates daily seasonal standard deviation of the time-series data in a DataFrame
monthly_average(merged_data)	Calculates monthly seasonal averages of the timeseries data in a DataFrame
monthly_std_error(merged_data)	Calculates monthly seasonal standard error of the time-series data in a DataFrame
monthly_std_dev(merged_data)	Calculates monthly seasonal standard deviation of the timeseries data in a DataFrame
remove_nan_df(merged_dataframe)	Drops rows with NaN, zero, negative, and inf values from a pandas dataframe
seasonal_period(merged_dataframe, daily_period)	Creates a dataframe with a specified seasonal period

julian_to_gregorian

hydrostats.data.**julian_to_gregorian**(*dataframe*, *frequency=None*, *inplace=False*)

Converts the index of the merged dataframe from julian float values to gregorian datetime values.

Parameters

- **dataframe** (*Pandas DataFrame*) – A DataFrame with an index of type float
- **frequency** (*string*) – Optional. Sometimes when converting from julian to gregorian there will be rounding errors due to the inability of computers to store floats as perfect decimals. Providing the frequency will automatically attempt to round the dates. A list of all the frequencies pandas provides is found here. Common frequencies include daily ("D") and hourly ("H").
- **inplace** (*bool*) – Default False. If True, will modify the index of the dataframe in place rather than creating a copy and returning the copy. Use when the time series are very long and making a copy would take a large amount of memory

Returns A pandas DataFrame with gregorian index.

Return type Pandas DataFrame

Examples

```
>>> import pandas as pd
>>> import hydrostats.data as hd
>>> import numpy as np
```

```
>>> # The julian dates in an array
>>> julian_dates = np.array([2444239.5, 2444239.5416666665, 2444239.5833333335,␣
↪2444239.625,
>>>                          2444239.6666666665, 2444239.7083333335, 2444239.75,
>>>                          2444239.7916666665, 2444239.8333333335, 2444239.875])
>>> # Creating a test dataframe
>>> test_df = pd.DataFrame(data=np.random.rand(10, 2),  # Random data in the␣
↪columns
>>>                        columns=("Simulated Data", "Observed Data"),
>>>                        index=julian_dates)
>>> test_df
              Simulated Data  Observed Data
2.444240e+06        0.764719       0.126610
2.444240e+06        0.372736       0.141392
2.444240e+06        0.008645       0.686477
2.444240e+06        0.656825       0.480444
2.444240e+06        0.555247       0.869409
2.444240e+06        0.643896       0.549590
2.444240e+06        0.242720       0.799617
2.444240e+06        0.432421       0.185760
2.444240e+06        0.694631       0.136986
2.444240e+06        0.700422       0.390415
```

```
>>> # Making a new df with gregorian index
>>> test_df_gregorian = hd.julian_to_gregorian(test_df)
>>> test_df_gregorian
                           Simulated Data   Observed Data
1980-01-01 00:00:00.000000        0.585454        0.457238
1980-01-01 01:00:00.028800        0.524764        0.083464
1980-01-01 01:59:59.971200        0.516821        0.416683
1980-01-01 03:00:00.000000        0.948483        0.553874
1980-01-01 04:00:00.028800        0.492280        0.232901
1980-01-01 04:59:59.971200        0.527967        0.296395
1980-01-01 06:00:00.000000        0.650018        0.212802
1980-01-01 07:00:00.028800        0.585592        0.802971
1980-01-01 07:59:59.971200        0.448243        0.665814
1980-01-01 09:00:00.000000        0.137395        0.201721
```

```
>>> # Rounding can be applied due to floating point inaccuracy
>>> test_df_gregorian_rounded = julian_to_gregorian(test_df, frequency="H")  #␣
↪Hourly Rounding Frequency
>>> test_df_gregorian_rounded
                     Simulated Data  Observed Data
1980-01-01 00:00:00        0.309527       0.938991
1980-01-01 01:00:00        0.872284       0.497708
1980-01-01 02:00:00        0.168046       0.225845
1980-01-01 03:00:00        0.954494       0.275607
1980-01-01 04:00:00        0.875885       0.194380
1980-01-01 05:00:00        0.236849       0.992770
1980-01-01 06:00:00        0.639346       0.029808
```

(continues on next page)

(continued from previous page)

```
1980-01-01 07:00:00        0.855828       0.903927
1980-01-01 08:00:00        0.638805       0.916124
1980-01-01 09:00:00        0.273430       0.443980
```

```
>>> # The DataFrame can also be modified in place, increasing efficiency with␣
↪large time series
>>> julian_to_gregorian(test_df, inplace=True, frequency="H")
>>> test_df
                     Simulated Data  Observed Data
1980-01-01 00:00:00        0.309527       0.938991
1980-01-01 01:00:00        0.872284       0.497708
1980-01-01 02:00:00        0.168046       0.225845
1980-01-01 03:00:00        0.954494       0.275607
1980-01-01 04:00:00        0.875885       0.194380
1980-01-01 05:00:00        0.236849       0.992770
1980-01-01 06:00:00        0.639346       0.029808
1980-01-01 07:00:00        0.855828       0.903927
1980-01-01 08:00:00        0.638805       0.916124
1980-01-01 09:00:00        0.273430       0.443980
```

merge_data

hydrostats.data.**merge_data**(*sim_fpath=None*, *obs_fpath=None*, *sim_df=None*, *obs_df=None*, *interpolate=None*, *column_names=('Simulated', 'Observed')*, *simulated_tz=None*, *observed_tz=None*, *interp_type='pchip'*, *return_tz='Etc/UTC'*, *julian=False*, *julian_freq=None*)

Merges two dataframes or csv files, depending on the input.

Parameters

- **sim_fpath** (*str*) – The filepath to the simulated csv of data. Can be a url if the page is formatted correctly. The csv must be formatted with the dates in the left column and the data in the right column.
- **obs_fpath** (*str*) – The filepath to the observed csv. Can be a url if the page is formatted correctly. The csv must be formatted with the dates in the left column and the data in the right column.
- **sim_df** (*DataFrame*) – A pandas DataFrame containing the simulated data. Must be formatted with a datetime index and the simulated data values in column 0.
- **obs_df** (*DataFrame*) – A pandas DataFrame containing the simulated data. Must be formatted with a datetime index and the simulated data values in column 0.
- **interpolate** (*str*) – Must be either 'observed' or 'simulated'. Specifies which data set you would like to interpolate if interpolation is needed to properly merge the data.
- **column_names** (*tuple of str*) – Tuple of length two containing the column names that the user would like to set for the DataFrame that is returned. Note that the simulated data will be in the left column and the observed data will be in the right column
- **simulated_tz** (*str*) – The timezone of the simulated data. A full list of timezones can be found in the *List of Timezones*.
- **observed_tz** (*str*) – The timezone of the simulated data. A full list of timezones can be found in the *List of Timezones*.

- **interp_type** (*str*) – Which interpolation method to use. Uses the default pandas interpolater. Available types are found at http://pandas.pydata.org/pandas-docs/version/0.16.2/generated/pandas.DataFrame.interpolate.html
- **return_tz** (*str*) – What timezone the merged dataframe's index should be returned as. Default is 'Etc/UTC', which is recommended for simplicity.
- **julian** (*bool*) – If True, will parse the first column of the file to a datetime index from julian floating point time representation, this is only valid when supplying the sim_fpath and obs_fpath parameters. Users supplying two DataFrame objects must convert the index from Julian to Gregorian using the julian_to_gregorian function in this module
- **julian_freq** (*str*) – A string representing the frequency of the julian dates so that they can be rounded. See examples for usage.

Notes

The only acceptable time frequencies in the data are 15min, 30min, 45min, and any number of hours or days in between.

There are three scenarios to consider when merging your data:

1. The first scenario is that the timezones and the spacing of the time series matches (eg. 1 Day). In this case, you will want to leave the simulated_tz, observed_tz, and interpolate arguments empty, and the function will simply join the two csv's into a dataframe.
2. The second scenario is that you have two time series with matching time zones but not matching spacing. In this case you will want to leave the simulated_tz and observed_tz empty, and use the interpolate argument to tell the function which time series you would like to interpolate to match the other time series.
3. The third scenario is that you have two time series with different time zones and possibly different spacings. In this case you will want to fill in the simulated_tz, observed_tz, and interpolate arguments. This will then take timezones into account when interpolating the selected time series.

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

daily_average

hydrostats.data.**daily_average**(*df*, *rolling=False*, ***kwargs*)

Calculates daily seasonal averages of the timeseries data in a DataFrame

Parameters

- **df** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.
- **rolling** (*bool*) – If True, will calculate the rolling seasonal average.
- ****kwargs** (*pandas.DataFrame.rolling() properties, optional*) – Options for how to compute the rolling averages. If not provided, the default is to use the following parameters: {window=6, min_periods=1, center=True, closed="right"}. Specifying **kwargs will clear the defaults, however.

Returns A pandas dataframe with a string type index of date representations and the daily seasonal averages as float values in the columns.

Return type DataFrame

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

```
>>> hd.daily_average(merged_df)
       Streamflow Prediction Tool       GLOFAS
01/01                 7331.305278  7676.792460
01/02                 7108.640746  7753.671798
01/03                 6927.147740  7631.900453
01/04                 6738.162886  7483.029897
01/05                 6552.914171  7316.004227
01/06                 6388.213829  7154.650963
01/07                 6258.418600  7012.279722
                           ...          ...
12/25                 8321.367143  8948.101821
12/26                 8149.313143  8903.544978
12/27                 7994.357429  8807.690639
12/28                 7872.819143  8642.877365
12/29                 7791.741143  8435.175677
12/30                 7729.451143  8225.315074
12/31                 7656.042286  8041.918136
[366 rows x 2 columns]
```

daily_std_error

hydrostats.data.**daily_std_error**(*merged_data*)

Calculates daily seasonal standard error of the timeseries data in a DataFrame

Parameters **merged_data** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.

Returns A pandas dataframe with a string type index of date representations and the daily seasonal standard error as float values in the columns.

Return type DataFrame

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

```
>>> hd.daily_std_error(merged_df)
      Streamflow Prediction Tool      GLOFAS
01/01                  558.189895  494.958042
01/02                  553.290181  442.497656
01/03                  535.002487  432.096928
01/04                  514.511095  422.915060
01/05                  489.287216  411.861086
01/06                  463.321927  401.023620
01/07                  441.666108  395.703128
                          ...         ...
12/25                  613.876851  566.669886
12/26                  589.424434  567.179646
12/27                  582.957832  557.932109
12/28                  581.465297  540.021918
12/29                  573.949000  517.494155
12/30                  560.993945  495.040565
12/31                  546.904139  474.742075
[366 rows x 2 columns]
```

daily_std_dev

hydrostats.data.**daily_std_dev**(*merged_data*)

Calculates daily seasonal standard deviation of the timeseries data in a DataFrame

Parameters **merged_data** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.

Returns A pandas dataframe with a string type index of date representations and the daily seasonal standard deviation as float values in the columns.

Return type DataFrame

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

```
>>> hd.daily_std_dev(merged_df)
       Streamflow Prediction Tool       GLOFAS
01/01                 3349.139373  2969.748253
01/02                 3273.308852  2617.851437
01/03                 3165.117397  2556.319898
01/04                 3043.888685  2501.999235
01/05                 2894.662206  2436.603046
01/06                 2741.049485  2372.487729
01/07                 2612.931931  2341.011275
                           ...          ...
12/25                 3631.744428  3352.464257
12/26                 3487.081980  3355.480036
12/27                 3448.825041  3300.770870
12/28                 3439.995086  3194.812751
12/29                 3395.528078  3061.536706
12/30                 3318.884936  2928.699478
12/31                 3235.528520  2808.611992
[366 rows x 2 columns]
```

monthly_average

hydrostats.data.**monthly_average**(*merged_data*)

Calculates monthly seasonal averages of the timeseries data in a DataFrame

Parameters **merged_data** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.

Returns A pandas dataframe with a string type index of date representations and the monthly seasonal averages as float values in the columns.

Return type DataFrame

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

```
>>> hd.monthly_average(merged_df)
        Streamflow Prediction Tool        GLOFAS
01                  5450.558878   6085.033102
02                  4178.249788   4354.072332
03                  4874.788452   4716.785701
04                  7682.920219   7254.073875
05                 13062.175899  11748.583189
06                 12114.431105  11397.032335
07                  9461.472599   9598.017209
08                  8802.643954   8708.876388
09                 10358.254219   9944.071882
10                 12968.474415  12671.180449
11                 13398.111010  13355.019167
12                  9853.288608  10275.652887
```

monthly_std_error

hydrostats.data.**monthly_std_error**(*merged_data*)

Calculates monthly seasonal standard error of the timeseries data in a DataFrame

Parameters **merged_data** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.

Returns A pandas dataframe with a string type index of date representations and the monthly seasonal standard error as float values in the columns.

Return type DataFrame

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

```
>>> hd.monthly_std_error(merged_df)
    Streamflow Prediction Tool      GLOFAS
01                   75.348943   71.206858
02                   65.182159   58.347438
03                   83.865980   72.919782
04                  131.199766  123.486202
05                  165.707528  139.586564
06                  149.938998  136.119774
07                  136.449337  115.436215
08                  129.371343  110.101251
09                  143.367894  123.798265
10                  133.911782  114.008446
11                  146.896826  116.443188
12                  135.092750  117.958715
```

monthly_std_dev

hydrostats.data.**monthly_std_dev**(*merged_data*)

Calculates monthly seasonal standard deviation of the timeseries data in a DataFrame

Parameters **merged_data** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.

Returns A pandas dataframe with a string type index of date representations and the monthly seasonal standard deviation as float values in the columns.

Return type DataFrame

Examples

```
>>> import hydrostats.data as hd
>>> import pandas as pd
>>> pd.options.display.max_rows = 15
```

The data URLs contain streamflow data from two different models, and are provided from the Hydrostats Github page

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('Streamflow␣
↪Prediction Tool', 'GLOFAS'))
```

```
>>> hd.monthly_std_dev(merged_df)
    Streamflow Prediction Tool       GLOFAS
01                 2483.087791  2346.587412
02                 2049.872689  1834.931830
03                 2762.489873  2401.929369
04                 4251.358318  4001.410267
05                 5458.296296  4597.889041
06                 4858.578846  4410.784791
07                 4494.550854  3802.392524
08                 4261.406429  3626.662349
```

(continues on next page)

(continued from previous page)

```
09                 4645.650733  4011.522281
10                 4410.965472  3755.362766
11                 4760.001179  3773.190543
12                 4449.865762  3885.481991
```

remove_nan_df

hydrostats.data.**remove_nan_df**(*merged_dataframe*)

Drops rows with NaN, zero, negative, and inf values from a pandas dataframe

Parameters **merged_dataframe** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.

Returns Pandas dataframe with rows containing NaN, zero, negative, and inf values removed.

Return type DataFrame

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> import hydrostats.data as hd
```

An example dataframe is created with invalid values inserted into it.

```
>>> data = np.random.rand(15, 2)
>>> data[0, 0] = data[1, 1] = np.nan
>>> data[2, 0] = data[3, 1] = np.inf
>>> data[4, 0] = data[5, 1] = 0
>>> data[6, 0] = data[7, 1] = -0.1
>>> example_df = pd.DataFrame(data=data, index=pd.date_range('1980-01-01',␣
↪periods=15))
>>> example_df
                   0         1
1980-01-01       NaN  0.358903
1980-01-02  0.074718       NaN
1980-01-03       inf  0.515931
1980-01-04  0.498002       inf
1980-01-05  0.000000  0.617974
1980-01-06  0.522522  0.000000
1980-01-07 -0.100000  0.116625
1980-01-08  0.588054 -0.100000
1980-01-09  0.691136  0.561178
1980-01-10  0.998170  0.432411
1980-01-11  0.424473  0.599110
1980-01-12  0.330988  0.469158
1980-01-13  0.633894  0.191701
1980-01-14  0.183241  0.784494
1980-01-15  0.681419  0.303280
```

Now the NaN, inf, negative, and zero values can be remove with this function.

```
>>> hd.remove_nan_df(example_df)
                   0         1
```

(continues on next page)

(continued from previous page)

```
1980-01-09  0.691136  0.561178
1980-01-10  0.998170  0.432411
1980-01-11  0.424473  0.599110
1980-01-12  0.330988  0.469158
1980-01-13  0.633894  0.191701
1980-01-14  0.183241  0.784494
1980-01-15  0.681419  0.303280
```

seasonal_period

hydrostats.data.**seasonal_period**(*merged_dataframe*, *daily_period*, *time_range=None*, *numpy=False*)

Creates a dataframe with a specified seasonal period

Parameters

- **merged_dataframe** (*DataFrame*) – A pandas DataFrame with a datetime index and columns containing float type values.
- **daily_period** (*tuple of str*) – A list of length two with strings representing the start and end dates of the seasonal period (e.g. (01-01, 01-31) for Jan 1 to Jan 31.
- **time_range** (*tuple of str*) – A tuple of string values representing the start and end dates of the time range. Format is YYYY-MM-DD.
- **numpy** (*bool*) – If True, two numpy arrays will be returned instead of a pandas dataframe

Returns Pandas dataframe that has been truncated to fit the parameters specified for the seasonal period.

Return type DataFrame

Examples

```
>>> import pandas
>>> pd.options.display.max_rows = 15
>>> import numpy as np
>>> import hydrostats.data as hd
```

Here an example DataFrame is made with appx three years of data.

```
>>> example_df = pd.DataFrame(data=np.random.rand(1000, 2), index=pd.date_range(
↪'2000-01-01', periods=1000), columns=['Simulated', 'Observed'])
            Simulated  Observed
2000-01-01   0.862726  0.056597
2000-01-02   0.979643  0.915072
2000-01-03   0.857667  0.965057
2000-01-04   0.011238  0.033678
2000-01-05   0.011390  0.401728
2000-01-06   0.056505  0.047417
2000-01-07   0.615151  0.134103
               ...       ...
2002-09-20   0.883156  0.272355
2002-09-21   0.595319  0.406609
2002-09-22   0.415106  0.826873
```

(continues on next page)

(continued from previous page)

```
2002-09-23   0.399449  0.656040
2002-09-24   0.243404  0.561899
2002-09-25   0.879932  0.551347
2002-09-26   0.787526  0.887288
[1000 rows x 2 columns]
```

Using this function, a new dataframe containing only the data values in january is returned.

```
>>> seasonal_df_jan = hd.seasonal_period(example_df, ('01-01', '01-31'))
            Simulated  Observed
2000-01-01   0.862726  0.056597
2000-01-02   0.979643  0.915072
2000-01-03   0.857667  0.965057
2000-01-04   0.011238  0.033678
2000-01-05   0.011390  0.401728
2000-01-06   0.056505  0.047417
2000-01-07   0.615151  0.134103
               ...       ...
2002-01-25   0.230580  0.363213
2002-01-26   0.579899  0.370847
2002-01-27   0.317925  0.120410
2002-01-28   0.196034  0.035715
2002-01-29   0.245429  0.974162
2002-01-30   0.156166  0.544797
2002-01-31   0.158595  0.311630
[93 rows x 2 columns]
```

We can also specify a time range if we only want the months of January in the year 2000 and 2001

```
>>> seasonal_df_jan = hd.seasonal_period(example_df, ('01-01', '01-31'), time_
↪range=('2000-01-01', '2001-12-31'))
            Simulated  Observed
2000-01-01   0.862726  0.056597
2000-01-02   0.979643  0.915072
2000-01-03   0.857667  0.965057
2000-01-04   0.011238  0.033678
2000-01-05   0.011390  0.401728
2000-01-06   0.056505  0.047417
2000-01-07   0.615151  0.134103
               ...       ...
2001-01-25   0.119188  0.043076
2001-01-26   0.896280  0.282883
2001-01-27   0.659078  0.230265
2001-01-28   0.667826  0.383687
2001-01-29   0.298459  0.738100
2001-01-30   0.336499  0.189036
2001-01-31   0.571562  0.783718
[62 rows x 2 columns]
```

1.3 Visualization (hydrostats.visual)

1.3.1 hydrostats.visual Module

The visual module contains different plotting functions for time series visualization. It allows users to plot hydrographs, scatter plots, histograms, and quantile-quantile (qq) plots to visualize time series data. In some of the visualization functions, metrics can be added to the plots for a more complete summary of the data.

Functions

plot(merged_data_df[, legend, metrics, ...])	Create a comparison time series line plot of simulated and observed time series data.
hist([merged_data_df, sim_array, obs_array, ...])	Plots a histogram comparing simulated and observed data.
scatter([merged_data_df, sim_array, ...])	Creates a scatter plot of the observed and simulated data.
qqplot([merged_data_df, sim_array, ...])	Plots a Quantile-Quantile plot of the simulated and observed data.

plot

hydrostats.visual.**plot**(*merged_data_df*, *legend=('Simulated Data', 'Observed Data')*, *metrics=None*, *grid=False*, *title=None*, *x_season=False*, *labels=None*, *linestyles=('ro', 'b^')*, *tight_xlim=False*, *fig_size=(10, 6)*, *text_adjust=(-0.35, 0.75)*, *plot_adjust=0.27*, *transparency=0.5*, *ebars=None*, *ecolor=None*, *markersize=2*, *errorevery=1*, *markevery=1*)

Create a comparison time series line plot of simulated and observed time series data.

The time series plot is a function that is available for viewing two times series plotted side by side vs. time. Goodness of fit metrics can also be viewed on the plot to compare the two time series.

Parameters

- **merged_data_df** (*DataFrame*) – DataFrame must contain datetime index and floating point type numbers in the two columns. The left columns must be simulated data and the right column observed data.
- **legend** (*tuple of str*) – Adds a Legend in the 'best' location determined by matplotlib. The entries in the tuple describe the left and right columns of the merged_data_df data.
- **metrics** (*list of str*) – Adds Metrics to the left side of the plot. Any metric from the Hydrostats library can be added to the plot as the abbreviation of the function. The entries must be in a list. (e.g. ['ME', 'r2', 'KGE (2012)']).
- **grid** (*bool*) – If True, adds a grid to the plot.
- **title** (*str*) – If given, adds a title to the plot.
- **x_season** (*bool*) – If True, the x-axis ticks will be monthly. This is a useful feature when plotting seasonal time series comparisons (e.g. daily averages).
- **labels** (*list of str*) – List of two str type inputs specifying x-axis labels and y-axis labels, respectively.
- **linestyles** (*list of str*) – List of two string type inputs thet will change the linestyle of the predicted and recorded data, respectively. Linestyle references can be found

in *Matplotlib Linestyles Help*.

- **tight_xlim** (*bool*) – If true, will set the padding to zero for the lines in the line plot.
- **fig_size** (*tuple of floats*) – Tuple of length two that specifies the horizontal and vertical lengths of the plot in inches, respectively.
- **text_adjust** (*tuple*) – Tuple of length two with float type inputs indicating the relative position of the text (x-coordinate, y-coordinate) when adding metrics to the plot.
- **plot_adjust** (*float*) – Specifies the relative position to shift the plot the the right when adding metrics to the plot.
- **transparency** (*float*) – Value between 0 to 1 indicating the transparency of the two lines that are plotted and error bars if they are plotted, lower means more transparent.
- **ebars** (*DataFrame*) – DataFrame must contain datetime index and two columns of data that specify the error of the plots, with the simulated error on the left and the observed error on the right. These dataframes can be created with the *daily_std_error*, *daily_std_dev*, *monthly_std_error*, and *monthly_std_dev* functions.
- **ecolor** (*tuple of str*) – Tuple of two sting type inputs specifying the colors of the errorbars (e.g. ['r', 'k'] would make red and black errorbars on the simulated and observed data, respectively).
- **markersize** (*float*) – Indicates the size of the markers on the plot, if markers are used.
- **errorevery** (*int*) – Specifies how often to put error bars on the plot.
- **markevery** (*int*) – Specifies how often to put markers on the plot if markers are used.

Returns fig – A matplotlib figure handle is returned, which can be viewed with the matplotlib.pyplot.show() command.

Return type Matplotlib figure instance

Examples

In this example two models are compared.

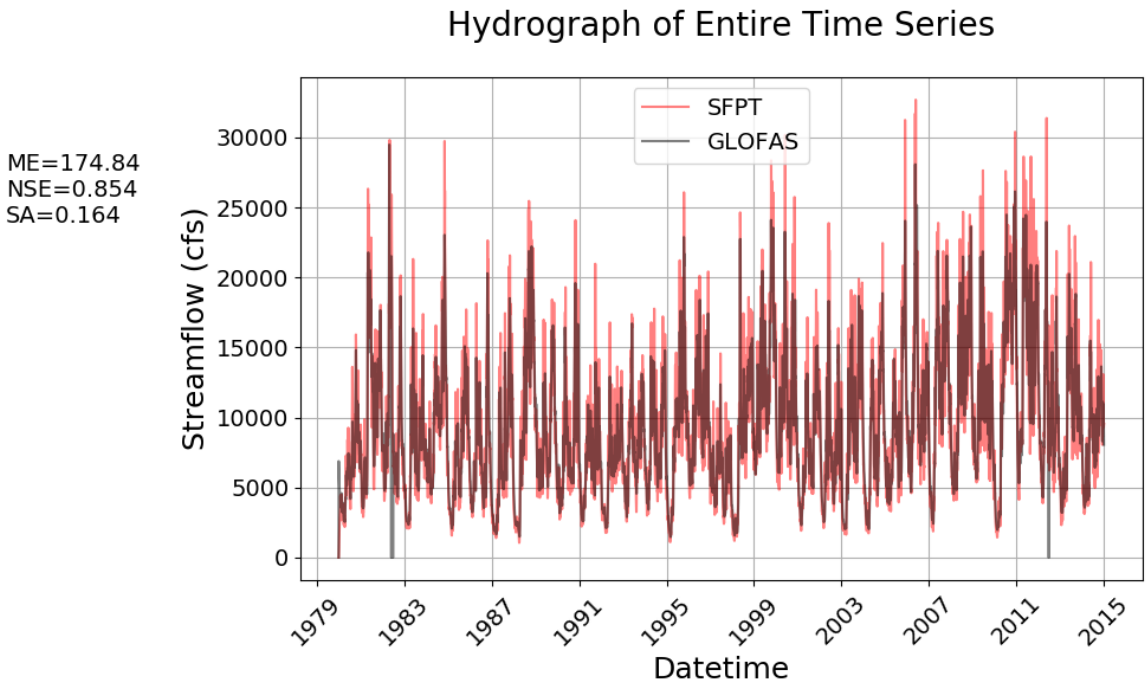
```
>>> import hydrostats.data as hd
>>> import hydrostats.visual as hv
>>> import matplotlib.pyplot as plt
```

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
```

```
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('SFPT', 'GLOFAS
↪'))
>>> seasonal_df = hd.seasonal_period(merged_df, ('04-01', '07-31'), time_range=(
↪'1986-01-01', '1992-12-31'))
>>> daily_avg_df = hd.daily_average(merged_data=merged_df)  # Seasonal Daily␣
↪Averages
>>> daily_std_error = hd.daily_std_error(merged_data=merged_df)  # Seasonal Daily␣
↪Standard Deviation
```

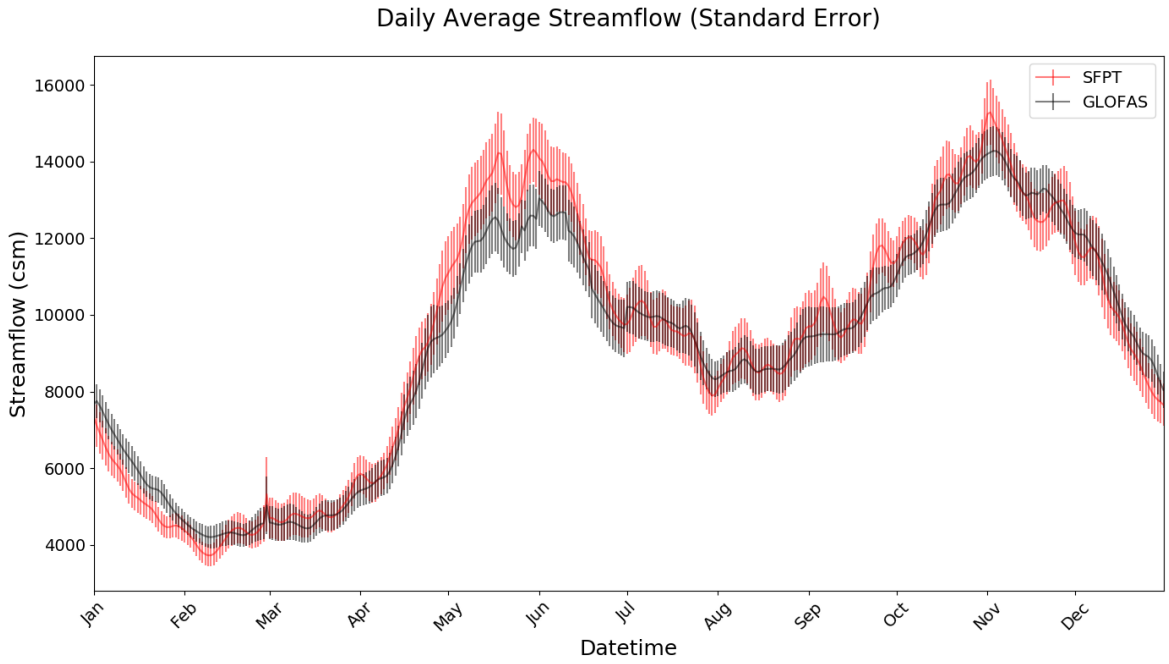
The entire timeseries is plotted below

```
>>> plot(merged_data_df=merged_df,
>>>      title='Hydrograph of Entire Time Series',
>>>      linestyles=['r-', 'k-'],
>>>      legend=('SFPT', 'GLOFAS'),
>>>      labels=['Datetime', 'Streamflow (cfs)'],
>>>      metrics=['ME', 'NSE', 'SA'],
>>>      grid=True)
>>> plt.show()
```



The seasonal averages with standard error bars is plotted below

```
>>> plot(merged_data_df=daily_avg_df,
>>>      title='Daily Average Streamflow (Standard Error)',
>>>      legend=('SFPT', 'GLOFAS'),
>>>      x_season=True,
>>>      labels=['Datetime', 'Streamflow (csm)'],
>>>      linestyles=['r-', 'k-'],
>>>      fig_size=(14, 8),
>>>      ebars=daily_std_error,
>>>      ecolor=('r', 'k'),
>>>      tight_xlim=True
>>>      )
>>> plt.show()
```



hist

hydrostats.visual.**hist**(*merged_data_df=None*, *sim_array=None*, *obs_array=None*, *num_bins=100*, *z_norm=False*, *legend=('Simulated', 'Observed')*, *grid=False*, *title=None*, *labels=None*, *prob_dens=False*, *figsize=(12, 6)*)

Plots a histogram comparing simulated and observed data.

The histogram plot is a function that is available for comparing the histograms of two time series. Data can be Z-score normalized as well as fit in a probability density function.

Parameters

- **merged_data_df** (*DataFrame*) – Dataframe must contain a datetime type index and floating point type numbers in two columns. The left column must be simulated data and the right column must be observed data. If given, sim_array and obs_array must be None.
- **sim_array** (*1D ndarray*) – Array of simulated data. If given, merged_data_df parameter must be None and obs_array must be given.
- **obs_array** (*1D ndarray*) – Array of observed data. If given, merged_data_df parameter must be None and sim_array must be given.
- **num_bins** (*int*) – Specifies the number of bins in the histogram.
- **z_norm** (*bool*) – If True, the data will be Z-score normalized.
- **legend** (*tuple of str*) – Tuple of length two with str inputs. Adds a Legend in the 'best' location determined by matplotlib. The entries in the tuple label the simulated and observed data (e.g. ['Simulated Data', 'Predicted Data']).
- **grid** (*bool*) – If True, adds a grid to the plot.
- **title** (*str*) – If given, sets the title of the plot.

- **labels** (*tuple of str*) – Tuple of two string type objects to set the x-axis labels and y-axis labels, respectively.
- **prob_dens** (*bool*) – If True, normalizes both histograms to form a probability density, i.e., the area (or integral) under each histogram will sum to 1.
- **figsize** (*tuple of float*) – Tuple of length two that specifies the horizontal and vertical lengths of the plot in inches, respectively.

Returns fig – A matplotlib figure handle is returned, which can be viewed with the matplotlib.pyplot.show() command.

Return type Matplotlib figure instance

Examples

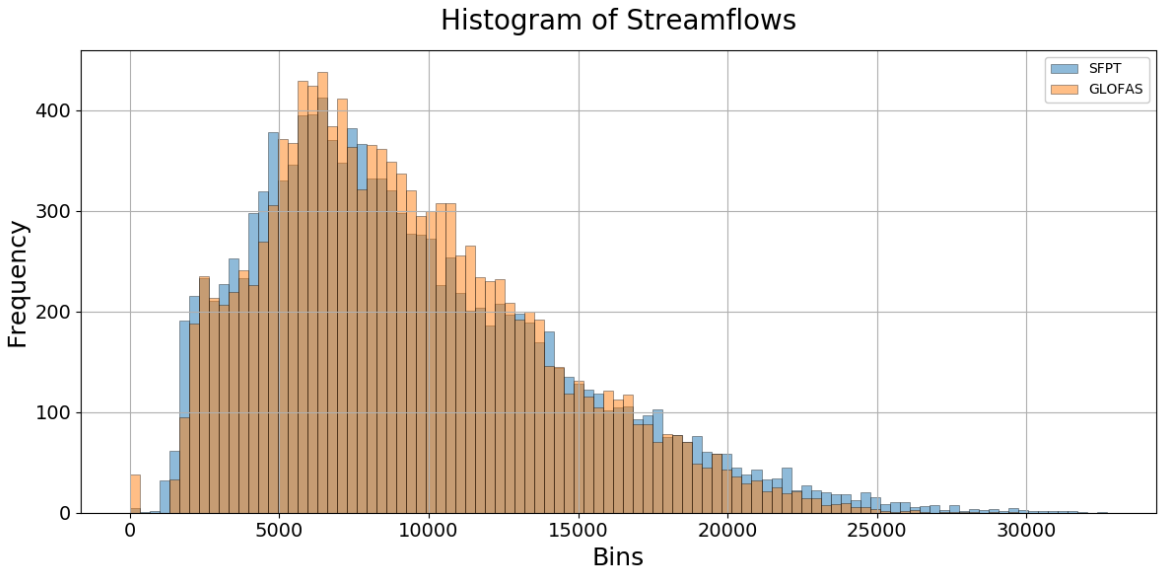
In this example the histograms of two models are compared to check their distributions

```
>>> import hydrostats.data as hd
>>> import hydrostats.visual as hv
>>> import matplotlib.pyplot as plt
```

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('SFPT', 'GLOFAS
↪'))
```

The histogram with 100 bins is plotted below

```
>>> hist(merged_data_df=merged_df,
>>>      num_bins=100,
>>>      title='Histogram of Streamflows',
>>>      legend=('SFPT', 'GLOFAS'),
>>>      labels=('Bins', 'Frequency'),
>>>      grid=True)
>>> plt.show()
```



scatter

hydrostats.visual.**scatter**(*merged_data_df=None*, *sim_array=None*, *obs_array=None*, *grid=False*, *title=None*, *labels=None*, *best_fit=False*, *marker_style='ko'*, *metrics=None*, *log_scale=False*, *line45=False*, *figsize=(12, 8)*)

Creates a scatter plot of the observed and simulated data.

Parameters

- **merged_data_df** (*DataFrame*) – Dataframe must contain a datetime type index and floating point type numbers in two columns. The left column must be simulated data and the right column must be observed data. If given, sim_array and obs_array must be None.
- **sim_array** (*1D ndarray*) – Array of simulated data. If given, merged_data_df parameter must be None and obs_array must be given.
- **obs_array** (*1D ndarray*) – Array of observed data. If given, merged_data_df parameter must be None and sim_array must be given.
- **grid** (*bool*) – If True, adds a grid to the plot.
- **title** (*str*) – If given, sets the title of the plot.
- **labels** (*tuple of str*) – Tuple of two string type objects to set the x-axis labels and y-axis labels, respectively.
- **best_fit** (*bool*) – If True, adds a best linear regression line on the graph with the equation for the line in the legend.
- **marker_style** (*str*) – If give, changes the markerstyle of the points on the scatter plot. Matplotlib styling guides are found in *Matplotlib Linestyles Help*.
- **metrics** (*list of str*) – Adds Metrics to the left side of the plot. Any metric from the Hydrostats library can be added to the plot as the abbreviation of the function. The entries must be in a list. (e.g. ['ME', 'r2', 'KGE (2012)']).
- **log_scale** (*bool*) – If True, log-log scale will be used on the scatter plot.

- **line45** (*bool*) – IF Trre, adds a 45 degree line to the plot and the legend.
- **figsize** (*tuple of float*) – Tuple of length two that specifies the horizontal and vertical lengths of the plot in inches, respectively.

Returns fig – A matplotlib figure handle is returned, which can be viewed with the matplotlib.pyplot.show() command.

Return type Matplotlib figure instance

Examples

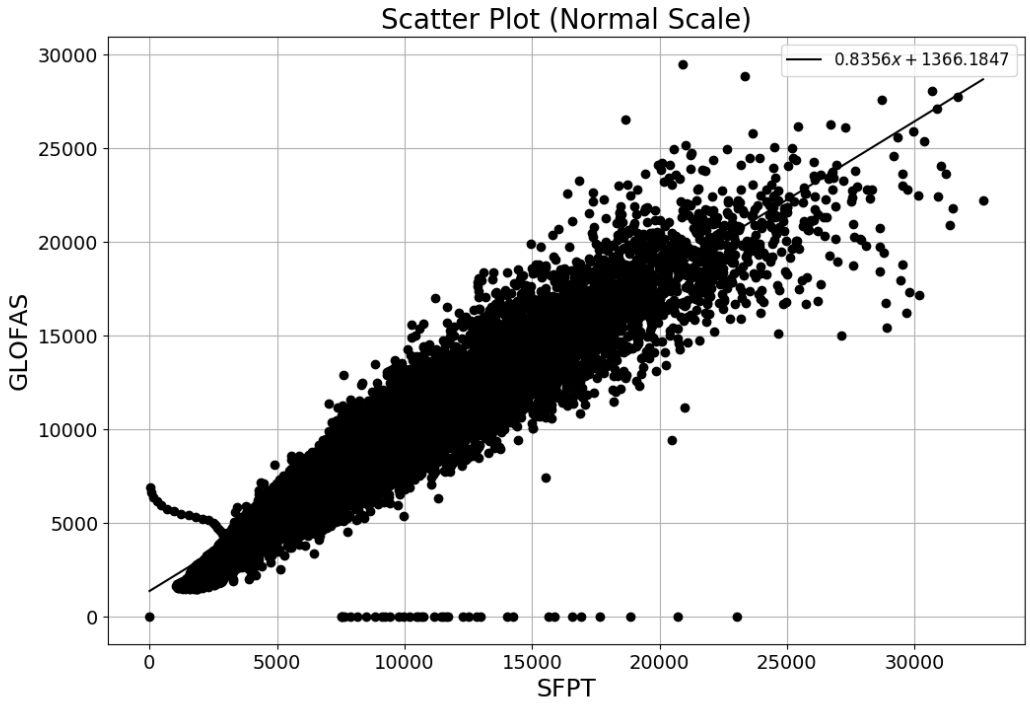
A scatter plot is created in this example comparing two models.

```
>>> import hydrostats.data as hd
>>> import hydrostats.visual as hv
>>> import matplotlib.pyplot as plt
```

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('SFPT', 'GLOFAS
↪'))
```

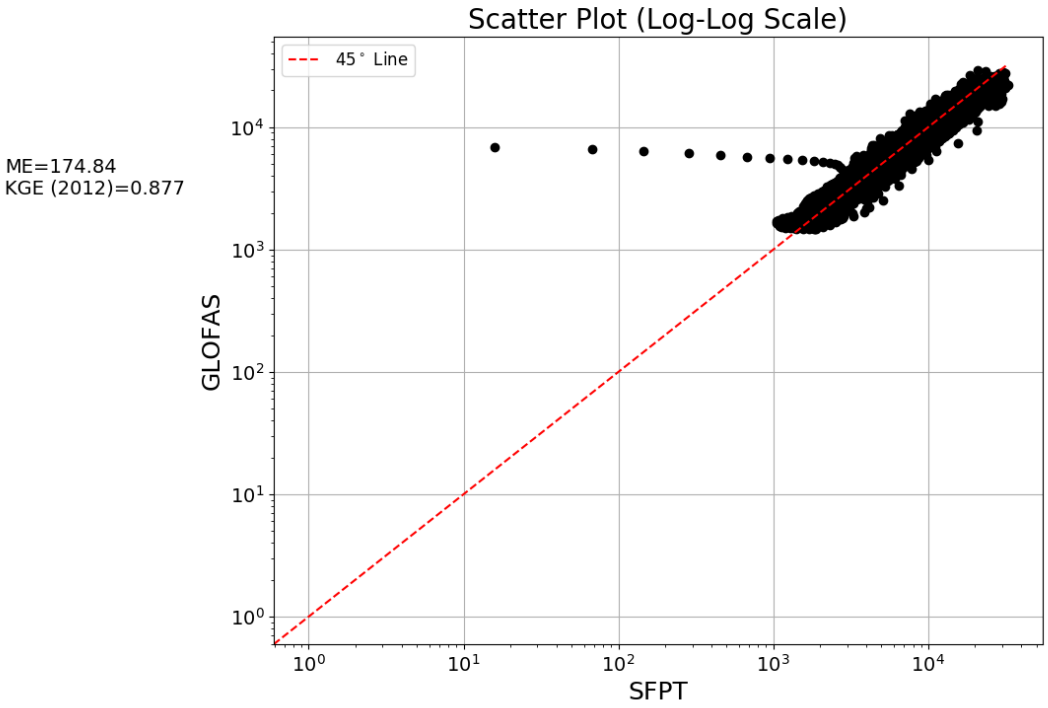
```
>>> sim_array = merged_df.iloc[:, 0].values
>>> obs_array = merged_df.iloc[:, 1].values
```

```
>>> scatter(merged_data_df=merged_df, grid=True, title='Scatter Plot (Normal␣
↪Scale)',
>>>         labels=('SFPT', 'GLOFAS'), best_fit=True)
>>> plt.show()
```



Arrays can be used as well in the parameters, as demonstrated below.

```
>>> scatter(sim_array=sim_array, obs_array=obs_array, grid=True, title='Scatter␣
↪Plot (Log-Log Scale)',
>>>         labels=('SFPT', 'GLOFAS'), line45=True, metrics=['ME', 'KGE (2012)'])
>>> plt.show()
```



qqplot

hydrostats.visual.**qqplot**(*merged_data_df=None*, *sim_array=None*, *obs_array=None*, *interpolate='linear'*, *title=None*, *xlabel='Simulated Data Quantiles'*, *ylabel='Observed Data Quantiles'*, *legend=False*, *replace_nan=None*, *replace_inf=None*, *remove_neg=False*, *remove_zero=False*, *figsize=(12, 8)*)

Plots a Quantile-Quantile plot of the simulated and observed data.

Useful for comparing to see whether the two datasets come from the same distribution.

Parameters

- **merged_data_df** (*DataFrame*) – Dataframe must contain a datetime type index and floating point type numbers in two columns. The left column must be simulated data and the right column must be observed data. If given, sim_array and obs_array must be None.
- **sim_array** (*1D ndarray*) – Array of simulated data. If given, merged_data_df parameter must be None and obs_array must be given.
- **obs_array** (*1D ndarray*) – Array of observed data. If given, merged_data_df parameter must be None and sim_array must be given.
- **interpolate** (*str*) – Specifies the interpolation type when computing quantiles.
- **title** (*str*) – If given, sets the title of the plot.
- **xlabel** (*str*) – The label for the x axis that holds the simulated data quantiles.
- **ylabel** (*str*) – The label for the y axis that holds the observed data quantiles.
- **legend** (*bool*) – If True, a legend to explain the elements on the plot will be added.

- **replace_nan** (*float, optional*) – If given, indicates which value to replace NaN values with in the two arrays. If None, when a NaN value is found at the i-th position in the observed OR simulated array, the i-th value of the observed and simulated array are removed before the computation.
- **replace_inf** (*float, optional*) – If given, indicates which value to replace Inf values with in the two arrays. If None, when an inf value is found at the i-th position in the observed OR simulated array, the i-th value of the observed and simulated array are removed before the computation.
- **remove_neg** (*boolean, optional*) – If True, when a negative value is found at the i-th position in the observed OR simulated array, the i-th value of the observed AND simulated array are removed before the computation.
- **remove_zero** (*boolean, optional*) – If true, when a zero value is found at the i-th position in the observed OR simulated array, the i-th value of the observed AND simulated array are removed before the computation.
- **figsize** (*tuple of float*) – Tuple of length two that specifies the horizontal and vertical lengths of the plot in inches, respectively.

Returns fig – A matplotlib figure handle is returned, which can be viewed with the matplotlib.pyplot.show() command.

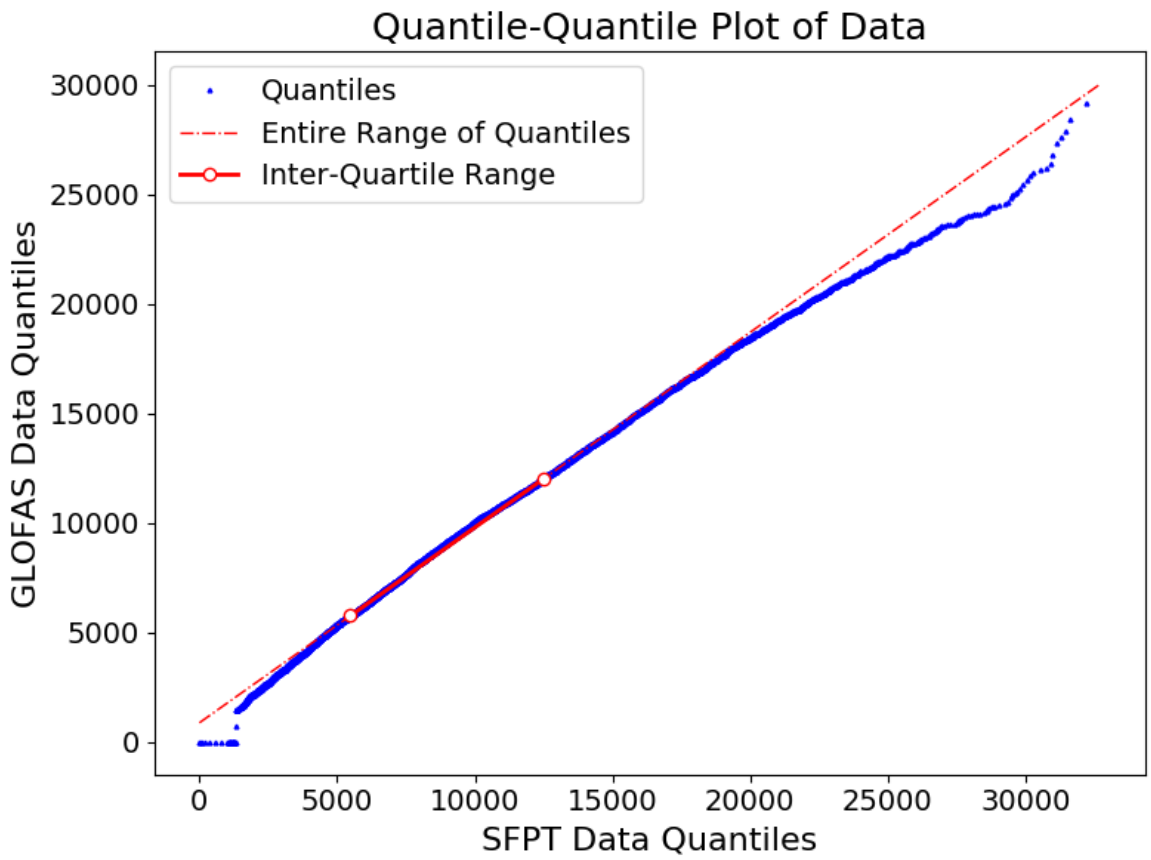
Return type Matplotlib figure instance

Examples

```
>>> import hydrostats.data as hd
>>> import hydrostats.visual as hv
>>> import matplotlib.pyplot as plt
```

```
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('SFPT', 'GLOFAS
↪'))
```

```
>>> qqplot(merged_data_df=merged_df, title='Quantile-Quantile Plot of Data',
>>>         xlabel='SFPT Data Quantiles', ylabel='GLOFAS Data Quantiles',␣
↪legend=True,
>>>         figsize=(8, 6))
>>> plt.show()
```



1.4 Metrics of Hydrological Skill (hydrostats.metrics)

The metrics in Hydrostats are available through the HydroErr package. Below is a link to the HydroErr documentation page that lists all of the metrics contained in the package.

1.4.1 Link

https://hydroerr.readthedocs.io/en/stable/

1.4.2 Examples

The metrics can be imported into your scripts as part of the metrics module. An example is provided below showing how to use the metrics included in the package from HydroErr (if you would prefer to only use the Hydrostats package and not just import the HydroErr package).

```
import hydrostats.metrics as hm
import numpy as np

sim = np.array([5, 7, 9, 2, 4.5, 6.7])
obs = np.array([4.7, 6, 10, 2.5, 4, 6.8])
```

(continues on next page)

(continued from previous page)

```
mean_error = hm.me(sim, obs)
print(mean_error)
```

1.5 Metrics of Ensemble Forecast Skill (hydrostats.ens_metrics)

1.5.1 hydrostats.ens_metrics Module

The ens_metrics module contains all of the metrics included in hydrostats that measure forecast skill. Each forecast metric is contained in a function, and every metric has the ability to treat missing values as well as remove zero and negative values from the timeseries data. Users will be warned which start dates have been removed in the warnings that display during the function execution.

Functions

ens_me(obs[, fcst_ens, remove_zero, ...])	Calculate the mean error between observed values and the ensemble mean.
ens_mae(obs[, fcst_ens, remove_zero, ...])	Calculate the mean absolute error between observed values and the ensemble mean.
ens_mse(obs[, fcst_ens, remove_zero, ...])	Calculate the mean squared error between observed values and the ensemble mean.
ens_rmse(obs[, fcst_ens, remove_zero, ...])	Calculate the root mean squared error between observed values and the ensemble mean.
ens_pearson_r(obs, fcst_ens[, remove_neg, ...])	Calculate the pearson correlation coefficient between observed values and the ensemble mean.
crps_hersbach(obs, fcst_ens[, remove_neg, ...])	Calculate the the continuous ranked probability score (CRPS) as per equation 25-27 in Hersbach et al.
crps_kernel(obs, fcst_ens[, remove_neg, ...])	Compute the kernel representation of the continuous ranked probability score (CRPS).
ens_crps(obs, fcst_ens[, adj, remove_neg, ...])	Calculate the ensemble-adjusted Continuous Ranked Probability Score (CRPS)
ens_brier([fcst_ens, obs, threshold, ...])	Calculate the ensemble-adjusted Brier Score.
auroc([fcst_ens, obs, threshold, ...])	Calculates Area Under the Relative Operating Characteristic curve (AUROC) for a forecast and its verifying binary observation, and estimates the variance of the AUROC
skill_score(scores, bench_scores, perf_score)	Calculate the skill score of the given function.

ens_me

hydrostats.ens_metrics.**ens_me**(*obs*, *fcst_ens=None*, *remove_zero=False*, *remove_neg=False*, *reference='mean'*)

Calculate the mean error between observed values and the ensemble mean.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.

- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **reference** (*str*) – Determines the reference series against which to calculate the error. Choices are 'mean' and 'median'.

Returns The mean error between the observed time series data and the ensemble reference series.

Return type float

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array

```
>>> ensemble_array = (np.random.rand(100, 52) + 1) * 100  # 52 Ensembles
>>> observed_array = (np.random.rand(100) + 1) * 100
```

Computing the ME between the ensemble mean and the observed data. Note that because the data is random the errors cancel out, leaving a low ME value.

```
>>> em.ens_me(obs=observed_array, fcst_ens=ensemble_array)
-2.5217349574908074
```

ens_mae

hydrostats.ens_metrics.**ens_mae**(*obs*, *fcst_ens=None*, *remove_zero=False*, *remove_neg=False*, *reference='mean'*)

Calculate the mean absolute error between observed values and the ensemble mean.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **reference** (*str*) – Determines the reference series against which to calculate the error. Choices are 'mean' and 'median'.

Returns The mean absolute error between the observed time series data and the ensemble mean.

Return type float

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array

```
>>> ensemble_array = (np.random.rand(100, 52) + 1) * 100  # 52 Ensembles
>>> observed_array = (np.random.rand(100) + 1) * 100
```

Computing the ME between the ensemble mean and the observed data. Note that because the data is random the errors cancel out, leaving a low ME value.

```
>>> em.ens_mae(obs=observed_array, fcst_ens=ensemble_array)
26.35428724003365
```

ens_mse

hydrostats.ens_metrics.**ens_mse**(*obs*, *fcst_ens=None*, *remove_zero=False*, *remove_neg=False*, *reference='mean'*)

Calculate the mean squared error between observed values and the ensemble mean.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **reference** (*str*) – Determines the reference series against which to calculate the error. Choices are 'mean' and 'median'.

Returns The mean error between the observed time series data and the ensemble mean.

Return type float

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array

```
>>> ensemble_array = (np.random.rand(100, 52) + 1) * 100  # 52 Ensembles
>>> observed_array = (np.random.rand(100) + 1) * 100
```

Computing the MSE between the ensemble mean and the observed data

```
>>> em.ens_mse(obs=observed_array, fcst_ens=ensemble_array)
910.5648405687582
```

ens_rmse

hydrostats.ens_metrics.**ens_rmse**(*obs*, *fcst_ens=None*, *remove_zero=False*, *remove_neg=False*, *reference='mean'*)

Calculate the root mean squared error between observed values and the ensemble mean.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **reference** (*str*) – Determines the reference series against which to calculate the error. Choices are 'mean' and 'median'.

Returns The mean error between the observed time series data and the ensemble mean.

Return type float

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array

```
>>> ensemble_array = (np.random.rand(100, 52) + 1) * 100  # 52 Ensembles
>>> observed_array = (np.random.rand(100) + 1) * 100
```

Computing the MSE between the ensemble mean and the observed data

```
>>> em.ens_rmse(obs=observed_array, fcst_ens=ensemble_array)
30.17556694693172
```

ens_pearson_r

hydrostats.ens_metrics.**ens_pearson_r**(*obs*, *fcst_ens*, *remove_neg=False*, *remove_zero=False*, *reference='mean'*)

Calculate the pearson correlation coefficient between observed values and the ensemble mean.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.

- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **reference** (*str*) – Determines the reference series against which to calculate the error. Choices are 'mean' and 'median'.

Returns The pearson correlation coefficient between the observed time series data and the ensemble mean.

Return type float

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array

```
>>> ensemble_array = (np.random.rand(100, 52) + 1) * 100
>>> observed_array = (np.random.rand(100) + 1) * 100
```

Computing the MSE between the ensemble mean and the observed data

```
>>> em.ens_pearson_r(obs=observed_array, fcst_ens=ensemble_array)
-0.13236871294739733
```

crps_hersbach

hydrostats.ens_metrics.**crps_hersbach**(*obs*, *fcst_ens*, *remove_neg=False*, *remove_zero=False*)

Calculate the the continuous ranked probability score (CRPS) as per equation 25-27 in Hersbach et al. (2000).

It is strongly recommended to use the *hydrostats.ens_metric.ens_crps()* function for the improved performance, instead of using this particular implementation. This is meant more for a proof of concept of the algorithm presented in the literature.

Parameters

- **obs** (*1D ndarry*) – Array of observations for each start date
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.

Returns

Dictionary contains a number of *experimental* outputs including:

- ["crps"] 1D ndarray of crps values per n start dates.
- ["crpsMean1"] arithmetic mean of crps values.
- ["crpsMean2"] mean crps using eqn. 28 in Hersbach (2000).

Return type dict

Notes

NaN and inf treatment: If any value in obs or fcst_ens is NaN or inf, then the corresponding row in both fcst_ens (for all ensemble members) and in obs will be deleted.

References

- Hersbach, H. (2000) Decomposition of the Continuous Ranked Porbability Score for Ensemble Prediction Systems, Weather and Forecasting, 15, 559-570.

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array with all random numbers

```
>>> ens_array_random = (np.random.rand(15, 52) + 1) * 100
>>> obs_array_random = (np.random.rand(15) + 1) * 100
```

Creating an observed 1D array and an ensemble 2D array with noise.

```
>>> noise = np.random.normal(scale=1, size=(15, 52))
>>> x = np.linspace(1, 10, 15)
>>> observed_array = np.sin(x) + 10
>>> ensemble_array_noise = (np.ones((15, 52)).T * observed_array).T + noise
```

Computing the Hersbach CRPS values between the ensemble mean and the observed data with the random data.

```
>>> crps_dictionary_rand = em.crps_hersbach(obs_array_random, ens_array_random)
>>> print(crps_dictionary_rand['crps'])
[ 7.73360237  9.59248626 34.46719655 30.10271075  7.451665   16.07882352
 14.59543529  8.55181637 15.4833089   8.32422363 16.55108154 19.20821296
  8.39452279 12.59949378 27.82543302]
>>> crps_dictionary_rand['crpsMean1']
15.797334183277723
>>> crps_dictionary_rand['crpsMean2']
15.797334183277725
```

Computing the Hersbach CRPS values between the ensemble mean and the observed data with noise in the ensemble data.

```
>>> crps_dictionary_noise = em.crps_hersbach(obs=observed_array, fcst_
↪ens=ensemble_array_noise)
>>> print(crps_dictionary_noise['crps'])
[0.26921152 0.21388687 0.24927151 0.26047667 0.30234843 0.1996493
 0.2779844  0.29478927 0.275383   0.25682693 0.21485236 0.22824711
 0.2813889  0.21264652 0.18141063]
>>> crps_dictionary_noise['crpsMean1']
0.24789156041214705
>>> crps_dictionary_noise['crpsMean2']
0.24789156041214705
```

crps_kernel

hydrostats.ens_metrics.**crps_kernel**(*obs*, *fcst_ens*, *remove_neg=False*, *remove_zero=False*)

Compute the kernel representation of the continuous ranked probability score (CRPS).

Calculates the kernel representation of the continuous ranked probability score (CRPS) as per equation 3 in Leutbecher et al. (2018) and the adjusted (or fair) crps as per equation 6 in the same paper. Note that it was Gneiting and Raftery (2007) who show the kernel representation as calculated here is equivalent to the standard definition based on the integral over the squared error of the cumulative distribution.

It is strongly recommended to use the *hydrostats.ens_metric.ens_crps()* function for the improved performance, instead of using this particular implementation, this is meant more for a proof of concept and an algorithm implementation.

Parameters

- **obs** (*1D ndarray*) – array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.

Returns

Dictionary of outputs includes:

- ["crps"] 1D ndarray with crps values of length n.
- ["crpsAdjusted"] 1D ndarray with adjusted crps values of length n.
- ["crpsMean"] Arithmetic mean of crps values as a float.
- ["crpsAdjustedMean"] Arithmetic mean of adjusted crps values as a float.

Return type dict

Notes

NaN treatment: If any start date in obs is NaN, then the corresponding row in fcst (for all ensemble members) will also be deleted.

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array with all random numbers

```
>>> ens_array_random = (np.random.rand(15, 52) + 1) * 100
>>> obs_array_random = (np.random.rand(15) + 1) * 100
```

Creating an observed 1D array and an ensemble 2D array with noise.

```
>>> noise = np.random.normal(scale=1, size=(15, 52))
>>> x = np.linspace(1, 10, 15)
>>> observed_array = np.sin(x) + 10
>>> ensemble_array_noise = (np.ones((15, 52)).T * observed_array).T + noise
```

Computing the Hersbach CRPS values between the ensemble mean and the observed data with the random data.

```
>>> crps_dictionary_rand = em.crps_kernel(obs_array_random, ens_array_random)
>>> print(crps_dictionary_rand['crps'])
[ 7.73360237  9.59248626 34.46719655 30.10271075  7.451665   16.07882352
 14.59543529  8.55181637 15.4833089   8.32422363 16.55108154 19.20821296
  8.39452279 12.59949378 27.82543302]
>>> print(crps_dictionary_rand['crpsAdjusted'])
[ 7.43000827  9.29100065 34.14067524 29.76359191  7.14776152 15.75147589
 14.25192856  8.23647876 15.19419171  8.05998301 16.26113448 18.90686679
  8.09725139 12.24021268 27.45673444]
>>> crps_dictionary_rand['crpsMean']
15.797334183277723
>>> crps_dictionary_rand['crpsAdjustedMean']
15.481953018707593
```

Computing the Hersbach CRPS values between the ensemble mean and the observed data with noise in the ensemble data.

```
>>> crps_dictionary_noise = em.crps_kernel(obs=observed_array, fcst_ens=ensemble_
↪array_noise)
>>> print(crps_dictionary_noise['crps'])
[0.26921152 0.21388687 0.24927151 0.26047667 0.30234843 0.1996493
 0.2779844  0.29478927 0.275383   0.25682693 0.21485236 0.22824711
 0.2813889  0.21264652 0.18141063]
>>> print(crps_dictionary_noise['crpsAdjusted'])
[0.25850726 0.20482797 0.23908004 0.25032814 0.28996894 0.18961905
 0.2670867  0.2821429  0.2634554  0.24573507 0.20457832 0.21730326
 0.26951946 0.2034818  0.17198615]
>>> crps_dictionary_noise['crpsMean']
0.2478915604121471
>>> crps_dictionary_noise['crpsAdjustedMean']
0.23717469718824744
```

References

- Gneiting, T. and Raftery, A. E. (2007) Strictly proper scoring rules, prediction and estimation, J. Amer. Stat. Asoc., 102, 359-378.

- Leutbecher, M. (2018) Ensemble size: How suboptimal is less than infinity?, Q. J. R. Meteorol., Accepted.

ens_crps

hydrostats.ens_metrics.**ens_crps**(*obs*, *fcst_ens*, *adj=nan*, *remove_neg=False*, *remove_zero=False*, *llvm=True*)

Calculate the ensemble-adjusted Continuous Ranked Probability Score (CRPS)

Parameters

- **obs** (*1D ndarray*) – array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **adj** (*float or int*) – A positive number representing ensemble size for which the scores should be adjusted. If np.nan (default) scores will not be adjusted. This value can be 'np.inf', in which case the adjusted (or fair) crps values will be calculated as per equation 6 in Leutbecher et al. (2018).
- **remove_neg** (*bool*) – If True, when a negative value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **remove_zero** (*bool*) – If true, when a zero value is found at the i-th position in the observed OR ensemble array, the i-th value of the observed AND ensemble array are removed before the computation.
- **llvm** (*bool*) – If true (default) the crps will be calculated using the numba module, which uses the LLVM compiler infrastructure for enhanced performance. If this is not wanted, then set it to false for a pure python implementation.

Returns Dictionary contains two keys, crps and crpsMean. The value of crps is a list of the crps values. Note that if the ensemble forecast or the observed values contained NaN or inf values, or negative or zero values if specified, these start dates will not show up in the crps values. The crpsMean value is the arithmatic mean of the crps values.

Return type dict

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array with all random numbers

```
>>> ens_array_random = (np.random.rand(15, 52) + 1) * 100  # 52 Ensembles
>>> obs_array_random = (np.random.rand(15) + 1) * 100
```

Creating an observed 1D array and an ensemble 2D array with noise.

```
>>> noise = np.random.normal(scale=1, size=(15, 52))
>>> x = np.linspace(1, 10, 15)
>>> observed_array = np.sin(x) + 10
>>> ensemble_array_noise = (np.ones((15, 52)).T * observed_array).T + noise  # 52␣
↪Ensembles
```

Computing the crps values between the ensemble mean and the observed data with the random data. Note that the crps is relatively high because it is random.

```
>>> crps_dictionary_rand = em.ens_crps(obs_array_random, ens_array_random)
>>> print(crps_dictionary_rand['crps'])
[ 7.73360237  9.59248626 34.46719655 30.10271075  7.451665   16.07882352
 14.59543529  8.55181637 15.4833089   8.32422363 16.55108154 19.20821296
  8.39452279 12.59949378 27.82543302]
>>> crps_dictionary_rand['crpsMean']
15.797334183277709
```

Computing the crps values between the ensemble mean and the observed data with noise in the ensemble data. Note that the crps values are better because the forecast is closer to observed values.

```
>>> crps_dictionary_noise = em.ens_crps(obs=observed_array, fcst_ens=ensemble_
↪array_noise)
>>> print(crps_dictionary_noise['crps'])
[0.26921152 0.21388687 0.24927151 0.26047667 0.30234843 0.1996493
 0.2779844  0.29478927 0.275383   0.25682693 0.21485236 0.22824711
 0.2813889  0.21264652 0.18141063]
>>> crps_dictionary_noise['crpsMean']
0.24789156041214638
```

References

- Gneiting, T. and Raftery, A. E. (2007) Strictly proper scoring rules, prediction and estimation, J. Amer. Stat. Asoc., 102, 359-378.
- Leutbecher, M. (2018) Ensemble size: How suboptimal is less than infinity?, Q. J. R. Meteorol., Accepted.
- Ferro CAT, Richardson SR, Weigel AP (2008) On the effect of ensemble size on the discrete and continuous ranked probability scores. Meteorological Applications. doi: 10.1002/met.45
- Stefan Siegert (2017). SpecsVerification: Forecast Verification Routines for Ensemble Forecasts of Weather and Climate. R package version 0.5-2. https://CRAN.R-project.org/package=SpecsVerification

ens_brier

hydrostats.ens_metrics.**ens_brier**(*fcst_ens=None*, *obs=None*, *threshold=None*, *ens_threshold=None*, *obs_threshold=None*, *fcst_ens_bin=None*, *obs_bin=None*, *adj=None*)

Calculate the ensemble-adjusted Brier Score.

Range: 0 Brier 1, lower is better.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **threshold** (*float*) – The threshold for an event (e.g. if the event is a 100 year flood, the streamflow value that a 100 year flood would have to exceed.
- **ens_threshold** (*float*) – If different threshholds for the ensemble forecast and the observed data is desired, then this parameter can be set along with the 'obs_threshold' parameter to set different thresholds.

- **obs_threshold** (*float*) – If different threshholds for the ensemble forecast and the observed data is desired, then this parameter can be set along with the 'ens_threshold' parameter to set different thresholds.
- **fcst_ens_bin** (*1D ndarray*) – Binary array of observations for each start date. 1 for an event and 0 for a non-event.
- **obs_bin** (*2D ndarray*) – Binary array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members. 1 for an event and 0 for a non-event.
- **adj** (*float or int*) – A positive number representing ensemble size for which the scores should be adjusted. If None (default) scores will not be adjusted. This value can be 'np.inf', in which case the adjusted (or fair) Brier scores will be calculated.

Returns Array of length with the ensemble-adjusted Brier scores. Length may not equal n if data has been removed due to NaN or Inf values.

Return type 1D ndarray

Notes

NaN and inf treatment: If any value in obs or fcst_ens is NaN or inf, then the corresponding row in both fcst_ens (for all ensemble members) and in obs will be deleted.

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)  # For reproducibility
```

Creating an observed 1D array and an ensemble 2D array

```
>>> ensemble_array = (np.random.rand(15, 52) + 1) * 100  # 52 Ensembles
>>> observed_array = (np.random.rand(15) + 1) * 100
```

Computing the ensemble-adjusted Brier score between the ensemble mean and the observed data.

```
>>> print(em.ens_brier(obs=observed_array, fcst_ens=ensemble_array,␣
↪threshold=175))
[0.08321006 0.05325444 0.53402367 0.45303254 0.02995562 0.08321006
 0.08321006 0.03698225 0.02366864 0.0625     0.04474852 0.71597633
 0.04474852 0.04474852 0.09467456]
>>> np.mean(em.ens_brier(obs=observed_array, fcst_ens=ensemble_array,␣
↪threshold=175))
0.15919625246548325
```

When we manually create binary data we get the same result

```
>>> ensemble_array_bin = (ensemble_array > 175).astype(np.int)
>>> observed_array_bin = (observed_array > 175).astype(np.int)
>>> print(em.ens_brier(obs_bin=observed_array_bin, fcst_ens_bin=ensemble_array_
↪bin))
[0.08321006 0.05325444 0.53402367 0.45303254 0.02995562 0.08321006
 0.08321006 0.03698225 0.02366864 0.0625     0.04474852 0.71597633
 0.04474852 0.04474852 0.09467456]
```

(continues on next page)

(continued from previous page)

```
>>> np.mean(em.ens_brier(obs_bin=observed_array_bin, fcst_ens_bin=ensemble_array_
↪bin))
0.15919625246548325
```

References

- Ferro CAT, Richardson SR, Weigel AP (2008) On the effect of ensemble size on the discrete and continuous ranked probability scores. Meteorological Applications. doi: 10.1002/met.45
- Stefan Siegert (2017). SpecsVerification: Forecast Verification Routines for Ensemble Forecasts of Weather and Climate. R package version 0.5-2. https://CRAN.R-project.org/package=SpecsVerification

auroc

hydrostats.ens_metrics.**auroc**(*fcst_ens=None*, *obs=None*, *threshold=None*, *ens_threshold=None*, *obs_threshold=None*, *fcst_ens_bin=None*, *obs_bin=None*)

Calculates Area Under the Relative Operating Characteristic curve (AUROC) for a forecast and its verifying binary observation, and estimates the variance of the AUROC

Range: 0 AUROC 1, Higher is better.

Parameters

- **obs** (*1D ndarray*) – Array of observations for each start date.
- **fcst_ens** (*2D ndarray*) – Array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members.
- **threshold** (*float*) – The threshold for an event (e.g. if the event is a 100 year flood, the streamflow value that a 100 year flood would have to exceed.
- **ens_threshold** (*float*) – If different threshholds for the ensemble forecast and the observed data is desired, then this parameter can be set along with the 'obs_threshold' parameter to set different thresholds.
- **obs_threshold** (*float*) – If different threshholds for the ensemble forecast and the observed data is desired, then this parameter can be set along with the 'ens_threshold' parameter to set different thresholds.
- **fcst_ens_bin** (*1D ndarray*) – Binary array of observations for each start date. 1 for an event and 0 for a non-event.
- **obs_bin** (*2D ndarray*) – Binary array of ensemble forecast of dimension n x M, where n = number of start dates and M = number of ensemble members. 1 for an event and 0 for a non-event.

Notes

NaN and inf treatment: If any value in obs or fcst_ens is NaN or inf, then the corresponding row in both fcst_ens (for all ensemble members) and in obs will be deleted. A warning will be shown that informs the user of the rows that have been removed.

Returns An array of two elements, the AUROC and the estimated variance, respectively.

Return type 1D ndarray

Examples

```
>>> import numpy as np
>>> import hydrostats.ens_metrics as em
>>> np.random.seed(3849590438)
```

Creating an observed 1D array and an ensemble 2D array with all random numbers

```
>>> ens_array_random = (np.random.rand(100, 52) + 1) * 100
>>> obs_array_random = (np.random.rand(100) + 1) * 100
```

Creating an observed 1D array and an ensemble 2D array with noise.

```
>>> noise = np.random.normal(scale=1, size=(100, 52))
>>> x = np.linspace(1, 10, 100)
>>> observed_array = np.sin(x) + 10
>>> ensemble_array_noise = (np.ones((100, 52)).T * observed_array).T + noise
```

Calculating the ROC with random values. Note that the area under the curve is close to 0.5 because the data is random.

```
>>> print(em.auroc(obs=obs_array_random, fcst_ens=ens_array_random,␣
↪threshold=175))
[0.45560516 0.06406262]
```

Calculating the ROC with noise in the forecast values. Note that the ROC value is high because the forecast is more accurate.

```
>>> print(em.auroc(obs=observed_array, fcst_ens=ensemble_array_noise,␣
↪threshold=10))
[0.99137931 0.00566026]
```

References

- DeLong et al (1988): Comparing the Areas under Two or More Correlated Receiver Operating Characteristic Curves: A Nonparametric Approach. Biometrics. doi: 10.2307/2531595
- Sun and Xu (2014): Fast Implementation of DeLong's Algorithm for Comparing the Areas Under Correlated Receiver Operating Characteristic Curves. IEEE Sign Proc Let 21(11). doi: 10.1109/LSP.2014.2337313
- Stefan Siegert (2017). SpecsVerification: Forecast Verification Routines for Ensemble Forecasts of Weather and Climate. R package version 0.5-2. https://CRAN.R-project.org/package=SpecsVerification

skill_score

hydrostats.ens_metrics.**skill_score**(*scores*, *bench_scores*, *perf_score*, *eff_sample_size=None*, *remove_nan_inf=False*)

Calculate the skill score of the given function.

Parameters

- **scores** (*float or ndarray*) – The verification scores, or the mean of the verification scores in an ndarray (float).

- **bench_scores** (*float or ndarray*) – The reference or benchmark verification scores, or the mean of the benchmark scores (float).
- **perf_score** (*int or float*) – The perfect score of the score, typically 1 or 0.
- **eff_sample_size** (*float*) – The effective sample size of the data to be used when estimating the sampling uncertainty. Default is None, which will set the eff_sample_size to the length of scores.
- **remove_nan_inf** (*bool*) – If True, removes NaN and Inf values in the scores if they exist, pairwise. If False (default), the function will raise an exception.

Returns Dictionary containing: {"skillScore": Float, the skill score, "standardDeviation": Float, the estimated standard deviation of the skill score} If the scores and bench scores given were floats, the standard deviation will be NaN.

Return type dict

References

- Stefan Siegert (2017). SpecsVerification: Forecast Verification Routines for Ensemble Forecasts of Weather and Climate. R package version 0.5-2. https://CRAN.R-project.org/package=SpecsVerification

Examples

1.6 Analysis (hydrostats.analyze)

1.6.1 hydrostats.analyze Module

The analyze module contains functions that perform a more complex analysis of simulated and observed time series data. It allows users to make tables with metrics that they choose as well as different date ranges. It also allows users to run a time lag analysis of two time series.

Functions

make_table(merged_dataframe, metrics[, ...])	Create a table of user selected metrics with optional seasonal analysis.
time_lag(merged_dataframe, metrics[, ...])	Check metric values between simulated and observed data at different time lags.

make_table

hydrostats.analyze.**make_table**(*merged_dataframe*, *metrics*, *seasonal_periods=None*, *mase_m=1*, *dmod_j=1*, *nse_mod_j=1*, *h6_mhe_k=1*, *h6_ahe_k=1*, *h6_rmshe_k=1*, *d1_p_obs_bar_p=None*, *lm_x_obs_bar_p=None*, *kge2009_s=(1, 1, 1)*, *kge2012_s=(1, 1, 1)*, *replace_nan=None*, *replace_inf=None*, *remove_neg=False*, *remove_zero=False*, *location=None*)

Create a table of user selected metrics with optional seasonal analysis.

Creates a table with metrics as specified by the user. Seasonal periods can also be specified in order to compare different seasons and how well the simulated data matches the observed data. Has options to save the table to

either a csv or an excel workbook. Also has an option to add a column for the location of the data.

Parameters

- **merged_dataframe** (*DataFrame*) – A pandas dataframe that has two columns of predicted data (Col 0) and observed data (Col 1) with a datetime index.
- **metrics** (*list of str*) – A list of all the metrics that the user wants to calculate. The metrics abbreviations must be used (e.g. the abbreviation for the mean error is "ME". Each function has an attribute with the name and abbreviation, so this can be used instead (see example). Also, strings can be typed and found in the quick reference table in this documentation.
- **seasonal_periods** (*2D list of str, optional*) – If given, specifies the seasonal periods that the user wants to analyze (e.g. [['06-01', '06-30'], ['08-12', '11-23']] would analyze the dates from June 1st to June 30th and also August 8th to November 23). Note that the entire time series is analyzed with the selected metrics by default.
- **mase_m** (*int, Optional*) – Parameter for the mean absolute scaled error (MASE) metric.
- **dmod_j** (*int or float, optional*) – Parameter for the modified index of agreement (dmod) metric.
- **nse_mod_j** (*int or float, optional*) – Parameter for the modified Nash-Sutcliffe (nse_mod) metric.
- **h6_mhe_k** (*int or float, optional*) – Parameter for the H6 (MHE) metric.
- **h6_ahe_k** (*int or float, optional*) – Parameter for the H6 (AHE) metric
- **h6_rmshe_k** (*int or float, optional*) – Parameter for the H6 (RMSHE) metric
- **d1_p_obs_bar_p** (*float, optional*) – Parameter fot the Legate McCabe Index of Agreement (d1_p).
- **lm_x_obs_bar_p** (*float, optional*) – Parameter for the Lagate McCabe Efficiency Index (lm_index).
- **kge2009_s** (*tuple of floats*) – A tuple of floats of length three signifying how to weight the three values used in the Kling Gupta (2009) metric.
- **kge2012_s** (*tuple of floats*) – A tuple of floats of length three signifying how to weight the three values used in the Kling Gupta (2012) metric.
- **replace_nan** (*float, optional*) – If given, indicates which value to replace NaN values with in the two arrays. If None, when a NaN value is found at the i-th position in the observed OR simulated array, the i-th value of the observed and simulated array are removed before the computation.
- **replace_inf** (*float, optional*) – If given, indicates which value to replace Inf values with in the two arrays. If None, when an inf value is found at the i-th position in the observed OR simulated array, the i-th value of the observed and simulated array are removed before the computation.
- **remove_neg** (*boolean, optional*) – If True, when a negative value is found at the i-th position in the observed OR simulated array, the i-th value of the observed AND simulated array are removed before the computation.
- **remove_zero** (*boolean, optional*) – If true, when a zero value is found at the i-th position in the observed OR simulated array, the i-th value of the observed AND simulated array are removed before the computation.

- **location** (*str*) – The name of the location that will be created as a column in the table that is created. Useful for creating a large table with different datasets.

Returns Dataframe with rows containing the metric values at the different time ranges, and columns containing the metrics specified.

Return type DataFrame

Notes

If desired, users can export the tables to a CSV or Excel Workbook. This can be done using the built in methods of pandas. A link to CSV method can be found at https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html and a link to the Excel method can be found at https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html

Examples

First we need to get some data. The data here is pulled from the Streamflow Predication Tool model and the ECMWF forecasting model. We are comparing the two models in this example.

```
>>> import hydrostats.analyze as ha
>>> import hydrostats.data as hd
>>> from hydrostats.metrics import mae, r_squared, nse, kge_2012
>>>
>>> # Defining the URLs of the datasets
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
>>> # Merging the data
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('SFPT', 'GLOFAS
↪'))
```

Here we make a table and print the results:

```
>>> my_metrics = [mae.abbr, r_squared.abbr, nse.abbr, kge_2012.abbr]  # HydroErr␣
↪1.24 or greater is required to use these properties
>>> seasonal = [['01-01', '03-31'], ['04-01', '06-30'], ['07-01', '09-30'], ['10-
↪01', '12-31']]
>>> table = ha.make_table(merged_df, my_metrics, seasonal, remove_neg=True,␣
↪remove_zero=True, location='Magdalena')
>>> table
                        Location          MAE     ...           NSE  KGE (2012)
Full Time Series        Magdalena  1157.669988     ...      0.873684    0.872871
January-01:March-31     Magdalena   631.984177     ...      0.861163    0.858187
April-01:June-30        Magdalena  1394.640050     ...      0.813737    0.876890
July-01:September-30    Magdalena  1188.542871     ...      0.829492    0.831188
October-01:December-31  Magdalena  1410.852917     ...      0.793927    0.791257
```

time_lag

hydrostats.analyze.**time_lag**(*merged_dataframe*, *metrics*, *interp_freq='6H'*, *interp_type='pchip'*, *shift_range=(-30, 30)*, *mase_m=1*, *dmod_j=1*, *nse_mod_j=1*, *h6_mhe_k=1*, *h6_ahe_k=1*, *h6_rmshe_k=1*, *d1_p_obs_bar_p=None*, *lm_x_obs_bar_p=None*, *replace_nan=None*, *replace_inf=None*, *remove_neg=False*, *remove_zero=False*, *plot=False*, *plot_title='Metric Values as Different Lags'*, *ylabel='Metric Value'*, *xlabel='Number of Lags'*, *save_fig=None*, *figsize=(10, 6)*, *station=None*)

Check metric values between simulated and observed data at different time lags.

Runs a time lag analysis to check for potential timing errors in the simulated data. Can also create a plot using matplotlib of the metric values at different shifts.

Parameters

- **merged_dataframe** (*DataFrame*) – A pandas dataframe that has two columns of predicted data (Col 0) and observed data (Col 1) with a datetime index.
- **metrics** (*list of str*) – Metric abbreviations that the user would like to use in their lag analysis. Each metric function has a property that contains their abbreviation for convenience (see example).
- **interp_freq** (*str, optional*) – Frequency of the interpolation for both the simulated and observed time series.
- **interp_type** (*str, optional*) – Type of interpolation. Options are 'linear', 'pchip', or 'cubic'.
- **shift_range** (*tuple of 2 floats, optional*) – If given, specifies the range of the lag shifts. For example, if (-10, 10) was given, the function would shift the simulated array for all of the range between -10 and 10.
- **mase_m** (*int, Optional*) – Parameter for the mean absolute scaled error (MASE) metric.
- **dmod_j** (*int or float, optional*) – Parameter for the modified index of agreement (dmod) metric.
- **nse_mod_j** (*int or float, optional*) – Parameter for the modified Nash-Sutcliffe (nse_mod) metric.
- **h6_mhe_k** (*int or float, optional*) – Parameter for the H6 (MHE) metric.
- **h6_ahe_k** (*int or float, optional*) – Parameter for the H6 (AHE) metric
- **h6_rmshe_k** (*int or float, optional*) – Parameter for the H6 (RMSHE) metric
- **d1_p_obs_bar_p** (*float, optional*) – Parameter fot the Legate McCabe Index of Agreement (d1_p).
- **lm_x_obs_bar_p** (*float, optional*) – Parameter for the Lagate McCabe Efficiency Index (lm_index).
- **replace_nan** (*float, optional*) – If given, indicates which value to replace NaN values with in the two arrays. If None, when a NaN value is found at the i-th position in the observed OR simulated array, the i-th value of the observed and simulated array are removed before the computation.
- **replace_inf** (*float, optional*) – If given, indicates which value to replace Inf values with in the two arrays. If None, when an inf value is found at the i-th position in the

observed OR simulated array, the i-th value of the observed and simulated array are removed before the computation.

- **remove_neg** (*boolean, optional*) – If True, when a negative value is found at the i-th position in the observed OR simulated array, the i-th value of the observed AND simulated array are removed before the computation.
- **remove_zero** (*boolean, optional*) – If true, when a zero value is found at the i-th position in the observed OR simulated array, the i-th value of the observed AND simulated array are removed before the computation.
- **plot** (*bool, optional*) – If True, a plot will be created that visualizes the different error metrics at the different time lags.
- **plot_title** (*str, optional*) – If the plot parameter is true, this parameter will set the title of the plot.
- **ylabel** (*str, optional*) – If the plot parameter is true, this parameter will set the y axis label of the plot.
- **xlabel** (*str, optional*) – if the plot parameter is true, this parameter will set the x axis label of the plot.
- **save_fig** (*str, optional*) – If given, will save the matplotlib plot to the specified directory with the specified file name (e.g. /path/to/plot.png). A list of all available types of figures is given in *List of Plot File Types*.
- **figsize** (*tuple of length 2*) – Tuple that specifies the horizontal and vertical lengths of the plot, in inches.
- **station** (*str*) – The station of analysis. Includes the station in the table if given.

Notes

If desired, users can export the tables to a CSV or Excel Workbook. This can be done using the built in methods of pandas. A link to CSV method can be found at https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html and a link to the Excel method can be found at https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html

Returns The first DataFrame contains all of the metric values at different time lags, while the second dataframe contains the maximum and minimum metric values throughout the time lag, and the index of the maximum and minimum time lag values.

Return type Tuple of DataFrames

Examples

Using data from the Streamflow prediction tool RAPID model and the ECMWF model, we can conpare the two at different time lags

```
>>> import hydrostats.analyze as ha
>>> import hydrostats.data as hd
>>> from hydrostats.metrics import me, r_squared, rmse, kge_2012, nse, dr
>>>
>>> # Defining the URLs of the datasets
>>> sfpt_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/sfpt_data/magdalena-calamar_interim_data.csv'
>>> glofas_url = r'https://github.com/waderoberts123/Hydrostats/raw/master/Sample_
↪data/GLOFAS_Data/magdalena-calamar_ECMWF_data.csv'
```

(continues on next page)

(continued from previous page)

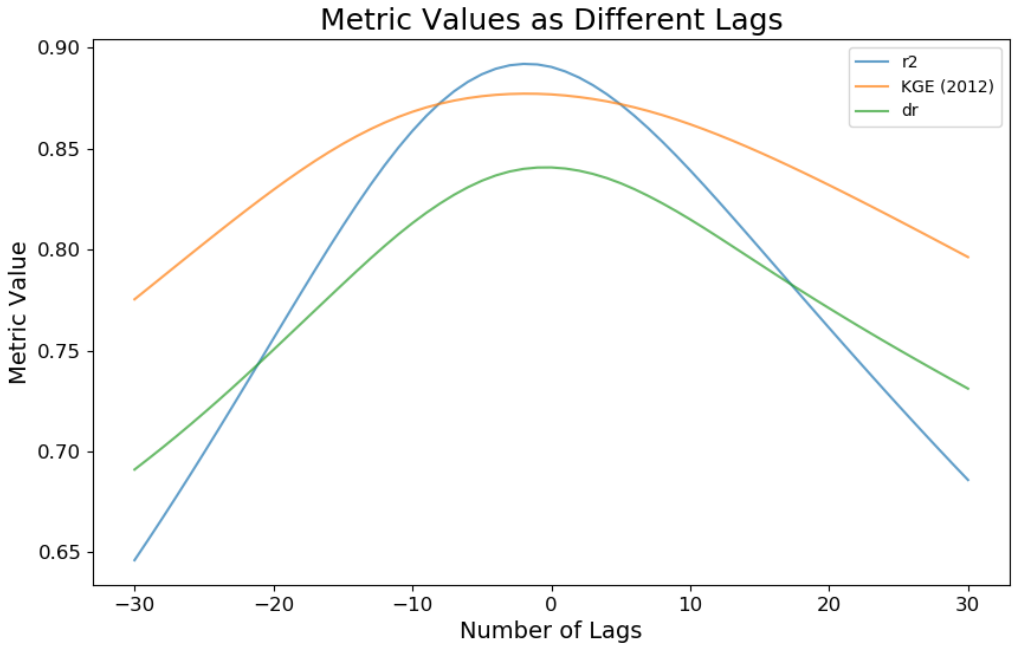
```
>>> # Merging the data
>>> merged_df = hd.merge_data(sfpt_url, glofas_url, column_names=('SFPT', 'GLOFAS
↪'))
```

There are two dataframes that are returned as part of the analysis.

```
>>> # Running the lag analysis, not that HydroErr > 1.24 must be used to access␣
↪the .abbr property
>>> time_lag_df, summary_df = ha.time_lag(merged_df, metrics=[me.abbr, r_squared.
↪abbr, rmse.abbr, kge_2012.abbr, nse.abbr])
>>> summary_df
                     Max  Max Lag Number          Min  Min Lag Number
ME            174.740510           -28.0   174.740510           -24.0
r2              0.891854            -2.0     0.646193           -30.0
RMSE         3142.795474           -30.0  1754.455598            -2.0
KGE (2012)      0.877116            -2.0     0.775328           -30.0
NSE             0.856358            -2.0     0.539076           -30.0
```

A plot can be created that visualizes the different metrics throughout the time lags. It can be saved using the savefig parameter as well if desired.

```
>>> _, _ = ha.time_lag(merged_df, metrics=[r_squared.abbr, kge_2012.abbr, dr.
↪abbr], plot=True)
```



1.7 Quick Reference Table

1.7.1 Metrics, Abbreviations, and Functions Quick Reference

This table contains a list of the metrics names, abbreviations, and the name of the functions that are associated with them. It is a good reference when creating tables or plots to be able to see what metrics are available for use and their abbreviation name.

Full Metric Name	Abbreviation	Function Name
Anomaly Correlation Coefficient	ACC	acc
Coefficient of Determination	r2	r_squared
Eclidean Distance	ED	ed
Geometric Mean Difference	GMD	g_mean_diff
Index of Agreement (d)	d	d
Index of Agreement (d1)	d1	d1
Index of Agreement Refined (dr)	dr	dr
Inertial Root Mean Square Error	IRMSE	irmse
Kling-Gupta Efficiency (2009)	KGE (2009)	kge_2009
Kling-Gupta Efficiency (2012)	KGE (2012)	kge_2012
Legate-McCabe Efficiency Index	E1'	lm_index
Legate-McCabe Index of Agreement	D1'	d1_p
Mean Absolute Error	MAE	mae
Mean Absolute H1 Error	H1 (MAHE)	h1_mahe
Mean Absolute H10 Error	H10 (MAHE)	h10_mahe
Mean Absolute H2 Error	H2 (MAHE)	h2_mahe
Mean Absolute H3 Error	H3 (MAHE)	h3_mahe
Mean Absolute H4 Error	H4 (MAHE)	h4_mahe
Mean Absolute H5 Error	H5 (MAHE)	h5_mahe
Mean Absolute H6 Error	H6 (MAHE)	h6_mahe
Mean Absolute H7 Error	H7 (MAHE)	h7_mahe
Mean Absolute H8 Error	H8 (MAHE)	h8_mahe
Mean Absolute Log Error	MALE	male
Mean Absolute Percentage Deviation	MAPD	mapd
Mean Absolute Percentage Error	MAPE	mape
Mean Absolute Scaled Error	MASE	mase
Mean Arctangent Absolute Percentage Error	MAAPE	maape
Mean Error	ME	me
Mean H1 Error	H1 (MHE)	h1_mhe
Mean H10 Error	H10 (MHE)	h10_mhe
Mean H2 Error	H2 (MHE)	h2_mhe
Mean H3 Error	H3 (MHE)	h3_mhe
Mean H4 Error	H4 (MHE)	h4_mhe
Mean H5 Error	H5 (MHE)	h5_mhe
Mean H6 Error	H6 (MHE)	h6_mhe
Mean H7 Error	H7 (MHE)	h7_mhe
Mean H8 Error	H8 (MHE)	h8_mhe
Mean Log Error	MLE	mle
Mean Squared Error	MSE	mse
Mean Squared Log Error	MSLE	msle
Mean Variance	MV	mean_var

Continued on next page

Table 5 – continued from previous page

Full Metric Name	Abbreviation	Function Name
Median Absolute Error	MdAE	mdae
Median Error	MdE	mde
Median Squared Error	MdSE	mdse
Mielke-Berry R	(MB) R	mb_r
Modified Index of Agreement	d (Mod.)	dmod
Modified Nash-Sutcliffe Efficiency	NSE (Mod.)	nse_mod
Nash-Sutcliffe Efficiency	NSE	nse
Normalized Eclidean Distance	NED	ned
Normalized Root Mean Square Error - IQR	NRMSE (IQR)	nrmse_iqr
Normalized Root Mean Square Error - Mean	NRMSE (Mean)	nrmse_mean
Normalized Root Mean Square Error - Range	NRMSE (Range)	nrmse_range
Pearson Correlation Coefficient	R (Pearson)	pearson_r
Relative Index of Agreement	d (Rel.)	drel
Relative Nash-Sutcliffe Efficiency	NSE (Rel.)	nse_rel
Root Mean Square Error	RMSE	rmse
Root Mean Square H1 Error	H1 (RMSHE)	h1_rmshe
Root Mean Square H10 Error	H10 (RMSHE)	h10_rmshe
Root Mean Square H2 Error	H2 (RMSHE)	h2_rmshe
Root Mean Square H3 Error	H3 (RMSHE)	h3_rmshe
Root Mean Square H4 Error	H4 (RMSHE)	h4_rmshe
Root Mean Square H5 Error	H5 (RMSHE)	h5_rmshe
Root Mean Square H6 Error	H6 (RMSHE)	h6_rmshe
Root Mean Square H7 Error	H7 (RMSHE)	h7_rmshe
Root Mean Square H8 Error	H8 (RMSHE)	h8_rmshe
Root Mean Squared Log Error	RMSLE	rmsle
Spearman Rank Correlation Coefficient	R (Spearman)	spearman_r
Spectral Angle	SA	sa
Spectral Correlation	SC	sc
Spectral Gradient Angle	SGA	sga
Spectral Information Divergence	SID	sid
Symmetric Mean Absolute Percentage Error (1)	SMAPE1	smape1
Symmetric Mean Absolute Percentage Error (2)	SMAPE2	smape2
Volumetric Efficiency	VE	ve
Watterson's M	M	watt_m

1.8 List of Plot File Types

These are all of different file types that the matplotlib plots created with Hydrostats can be saved as. Note that some of these types will require you to switch the backend (Read here for more on backends).

- Postscript (.ps)
- Encapsulated Postscript (.eps)
- Portable Document Format (.pdf)
- PGF code for LaTeX (.pgf)
- Portable Network Graphics (.png)
- Raw RGBA bitmap (.raw)

- Raw RGBA bitmap (.rgba)
- Scalable Vector Graphics (.svg)
- Scalable Vector Graphics (.svgz)
- Joint Photographic Experts Group (.jpg, .jpeg)
- Tagged Image File Format (.tif, .tiff)

1.9 List of Timezones

The list of timezones to use when applying timezones to your data. These timezones are baseod on Olsen timezone database, using the python package pytz.

- Africa/Abidjan (+00:00)
- Africa/Accra (+00:00)
- Africa/Addis_Ababa (+03:00)
- Africa/Algiers (+01:00)
- Africa/Asmara (+03:00)
- Africa/Asmera (+03:00)
- Africa/Bamako (+00:00)
- Africa/Bangui (+01:00)
- Africa/Banjul (+00:00)
- Africa/Bissau (+00:00)
- Africa/Blantyre (+02:00)
- Africa/Brazzaville (+01:00)
- Africa/Bujumbura (+02:00)
- Africa/Cairo (+02:00)
- Africa/Casablanca (+00:00)
- Africa/Ceuta (+01:00)
- Africa/Conakry (+00:00)
- Africa/Dakar (+00:00)
- Africa/Dar_es_Salaam (+03:00)
- Africa/Djibouti (+03:00)
- Africa/Douala (+01:00)
- Africa/El_Aaiun (+00:00)
- Africa/Freetown (+00:00)
- Africa/Gaborone (+02:00)
- Africa/Harare (+02:00)
- Africa/Johannesburg (+02:00)
- Africa/Juba (+03:00)

- Africa/Kampala (+03:00)
- Africa/Khartoum (+02:00)
- Africa/Kigali (+02:00)
- Africa/Kinshasa (+01:00)
- Africa/Lagos (+01:00)
- Africa/Libreville (+01:00)
- Africa/Lome (+00:00)
- Africa/Luanda (+01:00)
- Africa/Lubumbashi (+02:00)
- Africa/Lusaka (+02:00)
- Africa/Malabo (+01:00)
- Africa/Maputo (+02:00)
- Africa/Maseru (+02:00)
- Africa/Mbabane (+02:00)
- Africa/Mogadishu (+03:00)
- Africa/Monrovia (+00:00)
- Africa/Nairobi (+03:00)
- Africa/Ndjamena (+01:00)
- Africa/Niamey (+01:00)
- Africa/Nouakchott (+00:00)
- Africa/Ouagadougou (+00:00)
- Africa/Porto-Novo (+01:00)
- Africa/Sao_Tome (+00:00)
- Africa/Timbuktu (+00:00)
- Africa/Tripoli (+02:00)
- Africa/Tunis (+01:00)
- Africa/Windhoek (+02:00)
- America/Adak (-10:00)
- America/Anchorage (-09:00)
- America/Anguilla (-04:00)
- America/Antigua (-04:00)
- America/Araguaina (-03:00)
- America/Argentina/Buenos_Aires (-03:00)
- America/Argentina/Catamarca (-03:00)
- America/Argentina/ComodRivadavia (-03:00)
- America/Argentina/Cordoba (-03:00)

- America/Argentina/Jujuy (-03:00)
- America/Argentina/La_Rioja (-03:00)
- America/Argentina/Mendoza (-03:00)
- America/Argentina/Rio_Gallegos (-03:00)
- America/Argentina/Salta (-03:00)
- America/Argentina/San_Juan (-03:00)
- America/Argentina/San_Luis (-03:00)
- America/Argentina/Tucuman (-03:00)
- America/Argentina/Ushuaia (-03:00)
- America/Aruba (-04:00)
- America/Asuncion (-03:00)
- America/Atikokan (-05:00)
- America/Atka (-10:00)
- America/Bahia (-03:00)
- America/Bahia_Banderas (-06:00)
- America/Barbados (-04:00)
- America/Belem (-03:00)
- America/Belize (-06:00)
- America/Blanc-Sablon (-04:00)
- America/Boa_Vista (-04:00)
- America/Bogota (-05:00)
- America/Boise (-07:00)
- America/Buenos_Aires (-03:00)
- America/Cambridge_Bay (-07:00)
- America/Campo_Grande (-03:00)
- America/Cancun (-05:00)
- America/Caracas (-04:00)
- America/Catamarca (-03:00)
- America/Cayenne (-03:00)
- America/Cayman (-05:00)
- America/Chicago (-06:00)
- America/Chihuahua (-07:00)
- America/Coral_Harbour (-05:00)
- America/Cordoba (-03:00)
- America/Costa_Rica (-06:00)
- America/Creston (-07:00)

- America/Cuiaba (-03:00)
- America/Curacao (-04:00)
- America/Danmarkshavn (+00:00)
- America/Dawson (-08:00)
- America/Dawson_Creek (-07:00)
- America/Denver (-07:00)
- America/Detroit (-05:00)
- America/Dominica (-04:00)
- America/Edmonton (-07:00)
- America/Eirunepe (-05:00)
- America/El_Salvador (-06:00)
- America/Ensenada (-08:00)
- America/Fort_Nelson (-07:00)
- America/Fort_Wayne (-05:00)
- America/Fortaleza (-03:00)
- America/Glace_Bay (-04:00)
- America/Godthab (-03:00)
- America/Goose_Bay (-04:00)
- America/Grand_Turk (-04:00)
- America/Grenada (-04:00)
- America/Guadeloupe (-04:00)
- America/Guatemala (-06:00)
- America/Guayaquil (-05:00)
- America/Guyana (-04:00)
- America/Halifax (-04:00)
- America/Havana (-05:00)
- America/Hermosillo (-07:00)
- America/Indiana/Indianapolis (-05:00)
- America/Indiana/Knox (-06:00)
- America/Indiana/Marengo (-05:00)
- America/Indiana/Petersburg (-05:00)
- America/Indiana/Tell_City (-06:00)
- America/Indiana/Vevay (-05:00)
- America/Indiana/Vincennes (-05:00)
- America/Indiana/Winamac (-05:00)
- America/Indianapolis (-05:00)

- America/Inuvik (-07:00)
- America/Iqaluit (-05:00)
- America/Jamaica (-05:00)
- America/Jujuy (-03:00)
- America/Juneau (-09:00)
- America/Kentucky/Louisville (-05:00)
- America/Kentucky/Monticello (-05:00)
- America/Knox_IN (-06:00)
- America/Kralendijk (-04:00)
- America/La_Paz (-04:00)
- America/Lima (-05:00)
- America/Los_Angeles (-08:00)
- America/Louisville (-05:00)
- America/Lower_Princes (-04:00)
- America/Maceio (-03:00)
- America/Managua (-06:00)
- America/Manaus (-04:00)
- America/Marigot (-04:00)
- America/Martinique (-04:00)
- America/Matamoros (-06:00)
- America/Mazatlan (-07:00)
- America/Mendoza (-03:00)
- America/Menominee (-06:00)
- America/Merida (-06:00)
- America/Metlakatla (-09:00)
- America/Mexico_City (-06:00)
- America/Miquelon (-03:00)
- America/Moncton (-04:00)
- America/Monterrey (-06:00)
- America/Montevideo (-03:00)
- America/Montreal (-05:00)
- America/Montserrat (-04:00)
- America/Nassau (-05:00)
- America/New_York (-05:00)
- America/Nipigon (-05:00)
- America/Nome (-09:00)

- America/Noronha (-02:00)
- America/North_Dakota/Beulah (-06:00)
- America/North_Dakota/Center (-06:00)
- America/North_Dakota/New_Salem (-06:00)
- America/Ojinaga (-07:00)
- America/Panama (-05:00)
- America/Pangnirtung (-05:00)
- America/Paramaribo (-03:00)
- America/Phoenix (-07:00)
- America/Port-au-Prince (-05:00)
- America/Port_of_Spain (-04:00)
- America/Porto_Acre (-05:00)
- America/Porto_Velho (-04:00)
- America/Puerto_Rico (-04:00)
- America/Punta_Arenas (-03:00)
- America/Rainy_River (-06:00)
- America/Rankin_Inlet (-06:00)
- America/Recife (-03:00)
- America/Regina (-06:00)
- America/Resolute (-06:00)
- America/Rio_Branco (-05:00)
- America/Rosario (-03:00)
- America/Santa_Isabel (-08:00)
- America/Santarem (-03:00)
- America/Santiago (-03:00)
- America/Santo_Domingo (-04:00)
- America/Sao_Paulo (-02:00)
- America/Scoresbysund (-01:00)
- America/Shiprock (-07:00)
- America/Sitka (-09:00)
- America/St_Barthelemy (-04:00)
- America/St_Johns (-03:30)
- America/St_Kitts (-04:00)
- America/St_Lucia (-04:00)
- America/St_Thomas (-04:00)
- America/St_Vincent (-04:00)

- America/Swift_Current (-06:00)
- America/Tegucigalpa (-06:00)
- America/Thule (-04:00)
- America/Thunder_Bay (-05:00)
- America/Tijuana (-08:00)
- America/Toronto (-05:00)
- America/Tortola (-04:00)
- America/Vancouver (-08:00)
- America/Virgin (-04:00)
- America/Whitehorse (-08:00)
- America/Winnipeg (-06:00)
- America/Yakutat (-09:00)
- America/Yellowknife (-07:00)
- Antarctica/Casey (+11:00)
- Antarctica/Davis (+07:00)
- Antarctica/DumontDUrville (+10:00)
- Antarctica/Macquarie (+11:00)
- Antarctica/Mawson (+05:00)
- Antarctica/McMurdo (+13:00)
- Antarctica/Palmer (-03:00)
- Antarctica/Rothera (-03:00)
- Antarctica/South_Pole (+13:00)
- Antarctica/Syowa (+03:00)
- Antarctica/Troll (+00:00)
- Antarctica/Vostok (+06:00)
- Arctic/Longyearbyen (+01:00)
- Asia/Aden (+03:00)
- Asia/Almaty (+06:00)
- Asia/Amman (+02:00)
- Asia/Anadyr (+12:00)
- Asia/Aqtau (+05:00)
- Asia/Aqtobe (+05:00)
- Asia/Ashgabat (+05:00)
- Asia/Ashkhabad (+05:00)
- Asia/Atyrau (+05:00)
- Asia/Baghdad (+03:00)

- Asia/Bahrain (+03:00)
- Asia/Baku (+04:00)
- Asia/Bangkok (+07:00)
- Asia/Barnaul (+07:00)
- Asia/Beirut (+02:00)
- Asia/Bishkek (+06:00)
- Asia/Brunei (+08:00)
- Asia/Calcutta (+05:30)
- Asia/Chita (+09:00)
- Asia/Choibalsan (+08:00)
- Asia/Chongqing (+08:00)
- Asia/Chungking (+08:00)
- Asia/Colombo (+05:30)
- Asia/Dacca (+06:00)
- Asia/Damascus (+02:00)
- Asia/Dhaka (+06:00)
- Asia/Dili (+09:00)
- Asia/Dubai (+04:00)
- Asia/Dushanbe (+05:00)
- Asia/Famagusta (+02:00)
- Asia/Gaza (+02:00)
- Asia/Harbin (+08:00)
- Asia/Hebron (+02:00)
- Asia/Ho_Chi_Minh (+07:00)
- Asia/Hong_Kong (+08:00)
- Asia/Hovd (+07:00)
- Asia/Irkutsk (+08:00)
- Asia/Istanbul (+03:00)
- Asia/Jakarta (+07:00)
- Asia/Jayapura (+09:00)
- Asia/Jerusalem (+02:00)
- Asia/Kabul (+04:30)
- Asia/Kamchatka (+12:00)
- Asia/Karachi (+05:00)
- Asia/Kashgar (+06:00)
- Asia/Kathmandu (+05:45)

- Asia/Katmandu (+05:45)
- Asia/Khandyga (+09:00)
- Asia/Kolkata (+05:30)
- Asia/Krasnoyarsk (+07:00)
- Asia/Kuala_Lumpur (+08:00)
- Asia/Kuching (+08:00)
- Asia/Kuwait (+03:00)
- Asia/Macao (+08:00)
- Asia/Macau (+08:00)
- Asia/Magadan (+11:00)
- Asia/Makassar (+08:00)
- Asia/Manila (+08:00)
- Asia/Muscat (+04:00)
- Asia/Nicosia (+02:00)
- Asia/Novokuznetsk (+07:00)
- Asia/Novosibirsk (+07:00)
- Asia/Omsk (+06:00)
- Asia/Oral (+05:00)
- Asia/Phnom_Penh (+07:00)
- Asia/Pontianak (+07:00)
- Asia/Pyongyang (+08:30)
- Asia/Qatar (+03:00)
- Asia/Qyzylorda (+06:00)
- Asia/Rangoon (+06:30)
- Asia/Riyadh (+03:00)
- Asia/Saigon (+07:00)
- Asia/Sakhalin (+11:00)
- Asia/Samarkand (+05:00)
- Asia/Seoul (+09:00)
- Asia/Shanghai (+08:00)
- Asia/Singapore (+08:00)
- Asia/Srednekolymsk (+11:00)
- Asia/Taipei (+08:00)
- Asia/Tashkent (+05:00)
- Asia/Tbilisi (+04:00)
- Asia/Tehran (+03:30)

- Asia/Tel_Aviv (+02:00)
- Asia/Thimbu (+06:00)
- Asia/Thimphu (+06:00)
- Asia/Tokyo (+09:00)
- Asia/Tomsk (+07:00)
- Asia/Ujung_Pandang (+08:00)
- Asia/Ulaanbaatar (+08:00)
- Asia/Ulan_Bator (+08:00)
- Asia/Urumqi (+06:00)
- Asia/Ust-Nera (+10:00)
- Asia/Vientiane (+07:00)
- Asia/Vladivostok (+10:00)
- Asia/Yakutsk (+09:00)
- Asia/Yangon (+06:30)
- Asia/Yekaterinburg (+05:00)
- Asia/Yerevan (+04:00)
- Atlantic/Azores (-01:00)
- Atlantic/Bermuda (-04:00)
- Atlantic/Canary (+00:00)
- Atlantic/Cape_Verde (-01:00)
- Atlantic/Faeroe (+00:00)
- Atlantic/Faroe (+00:00)
- Atlantic/Jan_Mayen (+01:00)
- Atlantic/Madeira (+00:00)
- Atlantic/Reykjavik (+00:00)
- Atlantic/South_Georgia (-02:00)
- Atlantic/St_Helena (+00:00)
- Atlantic/Stanley (-03:00)
- Australia/ACT (+11:00)
- Australia/Adelaide (+10:30)
- Australia/Brisbane (+10:00)
- Australia/Broken_Hill (+10:30)
- Australia/Canberra (+11:00)
- Australia/Currie (+11:00)
- Australia/Darwin (+09:30)
- Australia/Eucla (+08:45)

- Australia/Hobart (+11:00)
- Australia/LHI (+11:00)
- Australia/Lindeman (+10:00)
- Australia/Lord_Howe (+11:00)
- Australia/Melbourne (+11:00)
- Australia/NSW (+11:00)
- Australia/North (+09:30)
- Australia/Perth (+08:00)
- Australia/Queensland (+10:00)
- Australia/South (+10:30)
- Australia/Sydney (+11:00)
- Australia/Tasmania (+11:00)
- Australia/Victoria (+11:00)
- Australia/West (+08:00)
- Australia/Yancowinna (+10:30)
- Brazil/Acre (-05:00)
- Brazil/DeNoronha (-02:00)
- Brazil/East (-02:00)
- Brazil/West (-04:00)
- CET (+01:00)
- CST6CDT (-06:00)
- Canada/Atlantic (-04:00)
- Canada/Central (-06:00)
- Canada/Eastern (-05:00)
- Canada/Mountain (-07:00)
- Canada/Newfoundland (-03:30)
- Canada/Pacific (-08:00)
- Canada/Saskatchewan (-06:00)
- Canada/Yukon (-08:00)
- Chile/Continental (-03:00)
- Chile/EasterIsland (-05:00)
- Cuba (-05:00)
- EET (+02:00)
- EST (-05:00)
- EST5EDT (-05:00)
- Egypt (+02:00)

- Eire (+00:00)
- Etc/GMT (+00:00)
- Etc/GMT+0 (+00:00)
- Etc/GMT+1 (-01:00)
- Etc/GMT+10 (-10:00)
- Etc/GMT+11 (-11:00)
- Etc/GMT+12 (-12:00)
- Etc/GMT+2 (-02:00)
- Etc/GMT+3 (-03:00)
- Etc/GMT+4 (-04:00)
- Etc/GMT+5 (-05:00)
- Etc/GMT+6 (-06:00)
- Etc/GMT+7 (-07:00)
- Etc/GMT+8 (-08:00)
- Etc/GMT+9 (-09:00)
- Etc/GMT-0 (+00:00)
- Etc/GMT-1 (+01:00)
- Etc/GMT-10 (+10:00)
- Etc/GMT-11 (+11:00)
- Etc/GMT-12 (+12:00)
- Etc/GMT-13 (+13:00)
- Etc/GMT-14 (+14:00)
- Etc/GMT-2 (+02:00)
- Etc/GMT-3 (+03:00)
- Etc/GMT-4 (+04:00)
- Etc/GMT-5 (+05:00)
- Etc/GMT-6 (+06:00)
- Etc/GMT-7 (+07:00)
- Etc/GMT-8 (+08:00)
- Etc/GMT-9 (+09:00)
- Etc/GMT0 (+00:00)
- Etc/Greenwich (+00:00)
- Etc/UCT (+00:00)
- Etc/UTC (+00:00)
- Etc/Universal (+00:00)
- Etc/Zulu (+00:00)

- Europe/Amsterdam (+01:00)
- Europe/Andorra (+01:00)
- Europe/Astrakhan (+04:00)
- Europe/Athens (+02:00)
- Europe/Belfast (+00:00)
- Europe/Belgrade (+01:00)
- Europe/Berlin (+01:00)
- Europe/Bratislava (+01:00)
- Europe/Brussels (+01:00)
- Europe/Bucharest (+02:00)
- Europe/Budapest (+01:00)
- Europe/Busingen (+01:00)
- Europe/Chisinau (+02:00)
- Europe/Copenhagen (+01:00)
- Europe/Dublin (+00:00)
- Europe/Gibraltar (+01:00)
- Europe/Guernsey (+00:00)
- Europe/Helsinki (+02:00)
- Europe/Isle_of_Man (+00:00)
- Europe/Istanbul (+03:00)
- Europe/Jersey (+00:00)
- Europe/Kaliningrad (+02:00)
- Europe/Kiev (+02:00)
- Europe/Kirov (+03:00)
- Europe/Lisbon (+00:00)
- Europe/Ljubljana (+01:00)
- Europe/London (+00:00)
- Europe/Luxembourg (+01:00)
- Europe/Madrid (+01:00)
- Europe/Malta (+01:00)
- Europe/Mariehamn (+02:00)
- Europe/Minsk (+03:00)
- Europe/Monaco (+01:00)
- Europe/Moscow (+03:00)
- Europe/Nicosia (+02:00)
- Europe/Oslo (+01:00)

- Europe/Paris (+01:00)
- Europe/Podgorica (+01:00)
- Europe/Prague (+01:00)
- Europe/Riga (+02:00)
- Europe/Rome (+01:00)
- Europe/Samara (+04:00)
- Europe/San_Marino (+01:00)
- Europe/Sarajevo (+01:00)
- Europe/Saratov (+04:00)
- Europe/Simferopol (+03:00)
- Europe/Skopje (+01:00)
- Europe/Sofia (+02:00)
- Europe/Stockholm (+01:00)
- Europe/Tallinn (+02:00)
- Europe/Tirane (+01:00)
- Europe/Tiraspol (+02:00)
- Europe/Ulyanovsk (+04:00)
- Europe/Uzhgorod (+02:00)
- Europe/Vaduz (+01:00)
- Europe/Vatican (+01:00)
- Europe/Vienna (+01:00)
- Europe/Vilnius (+02:00)
- Europe/Volgograd (+03:00)
- Europe/Warsaw (+01:00)
- Europe/Zagreb (+01:00)
- Europe/Zaporozhye (+02:00)
- Europe/Zurich (+01:00)
- GB (+00:00)
- GB-Eire (+00:00)
- GMT (+00:00)
- GMT+0 (+00:00)
- GMT-0 (+00:00)
- GMT0 (+00:00)
- Greenwich (+00:00)
- HST (-10:00)
- Hongkong (+08:00)

- Iceland (+00:00)
- Indian/Antananarivo (+03:00)
- Indian/Chagos (+06:00)
- Indian/Christmas (+07:00)
- Indian/Cocos (+06:30)
- Indian/Comoro (+03:00)
- Indian/Kerguelen (+05:00)
- Indian/Mahe (+04:00)
- Indian/Maldives (+05:00)
- Indian/Mauritius (+04:00)
- Indian/Mayotte (+03:00)
- Indian/Reunion (+04:00)
- Iran (+03:30)
- Israel (+02:00)
- Jamaica (-05:00)
- Japan (+09:00)
- Kwajalein (+12:00)
- Libya (+02:00)
- MET (+01:00)
- MST (-07:00)
- MST7MDT (-07:00)
- Mexico/BajaNorte (-08:00)
- Mexico/BajaSur (-07:00)
- Mexico/General (-06:00)
- NZ (+13:00)
- NZ-CHAT (+13:45)
- Navajo (-07:00)
- PRC (+08:00)
- PST8PDT (-08:00)
- Pacific/Apia (+14:00)
- Pacific/Auckland (+13:00)
- Pacific/Bougainville (+11:00)
- Pacific/Chatham (+13:45)
- Pacific/Chuuk (+10:00)
- Pacific/Easter (-05:00)
- Pacific/Efate (+11:00)

- Pacific/Enderbury (+13:00)
- Pacific/Fakaofo (+13:00)
- Pacific/Fiji (+13:00)
- Pacific/Funafuti (+12:00)
- Pacific/Galapagos (-06:00)
- Pacific/Gambier (-09:00)
- Pacific/Guadalcanal (+11:00)
- Pacific/Guam (+10:00)
- Pacific/Honolulu (-10:00)
- Pacific/Johnston (-10:00)
- Pacific/Kiritimati (+14:00)
- Pacific/Kosrae (+11:00)
- Pacific/Kwajalein (+12:00)
- Pacific/Majuro (+12:00)
- Pacific/Marquesas (-09:30)
- Pacific/Midway (-11:00)
- Pacific/Nauru (+12:00)
- Pacific/Niue (-11:00)
- Pacific/Norfolk (+11:00)
- Pacific/Noumea (+11:00)
- Pacific/Pago_Pago (-11:00)
- Pacific/Palau (+09:00)
- Pacific/Pitcairn (-08:00)
- Pacific/Pohnpei (+11:00)
- Pacific/Ponape (+11:00)
- Pacific/Port_Moresby (+10:00)
- Pacific/Rarotonga (-10:00)
- Pacific/Saipan (+10:00)
- Pacific/Samoa (-11:00)
- Pacific/Tahiti (-10:00)
- Pacific/Tarawa (+12:00)
- Pacific/Tongatapu (+13:00)
- Pacific/Truk (+10:00)
- Pacific/Wake (+12:00)
- Pacific/Wallis (+12:00)
- Pacific/Yap (+10:00)

- Poland (+01:00)
- Portugal (+00:00)
- ROC (+08:00)
- ROK (+09:00)
- Singapore (+08:00)
- Turkey (+03:00)
- UCT (+00:00)
- US/Alaska (-09:00)
- US/Aleutian (-10:00)
- US/Arizona (-07:00)
- US/Central (-06:00)
- US/East-Indiana (-05:00)
- US/Eastern (-05:00)
- US/Hawaii (-10:00)
- US/Indiana-Starke (-06:00)
- US/Michigan (-05:00)
- US/Mountain (-07:00)
- US/Pacific (-08:00)
- US/Samoa (-11:00)
- UTC (+00:00)
- Universal (+00:00)
- W-SU (+03:00)
- WET (+00:00)
- Zulu (+00:00)

1.10 Matplotlib Linestyles Help

Below are the guides to available linestyle options in matplotlib documentation. These styles can be combined (e.g. 'r-' is a red solid line, 'r^' is red triangle up markers).

- Colors
- Linestyles
- Marker Styles

1.11 Release Notes

This is the list of changes to Hydrostats between each release. For full details, see the commit logs at https://github.com/BYU-Hydroinformatics/Hydrostats.

1.11.1 Version 0.78

- Added the ability to use different thresholds for the ensemble forecast for the observed and ensemble forecast data in the hydrostats.ens_metrics.auroc() and hydrostats.ens_metrics.ens_brier() methods.
- Changes to documentation to reflect the addition of the .name and .abbr properties to metrics from the HydroErr package.

1.11.2 Version 0.77

- Added a new rolling average feature to the hydrostats.data.daily_average(). Set rolling=True as a parameter to use the defaults, or specify arguments from the pandas.DataFrame.rolling() method for a custom rolling average.
- Minor changes and more coverage.

1.11.3 Version 0.76

- Moved the documentation to new location at https://hydrostats.readthedocs.io/

1.11.4 Version 0.75

- Minor bug fixes and changes

1.11.5 Version 0.74

- Added support for parsing julian dates with the new hydrostats.data.julian_to_gregorian() function
- Added support for parsing files with julian dates in the hydrostats.data.merge_data() function.
- Added example code in the github repository, in the "Examples" directory.

1.12 Contributing to Hydrostats

1.12.1 Steps

The steps to contributing are simple:

1. Download git
2. Clone the Hydrostats repository
3. Develop features that you would like implemented, including a docstring in functions of classes in the numpydoc format.
4. Add a test case to cover the code that you created (hydrostats/tests directory, in tests.py)
5. Create a new pull request with your changes

A note to members of the BYU-Hydroinformatics group. Please make pull requests before merging changes to the master branch, as this allows continuous integration testing to take place.

CHAPTER 2

Indices and tables

- genindex
- search

Python Module Index

h

Index