PyHelpers

An open-source toolkit for facilitating Python users' data manipulation tasks

Release 2.5.0

Qian Fu

Birmingham Energy Institute, University of Birmingham

First release: **September 2019**

Last updated: **July 2026**

Table of Contents

List of Figures

Chapter 1

About PyHelpers

PyHelpers is an open-source Python package designed to streamline data (pre-)processing and manipulation tasks. It accommodates a wide range of functions and classes grounded in practical applications, making common data operations more accessible and efficient. This toolkit is particularly useful for Python learners, researchers and data scientists seeking to enhance their workflows.

The package supports handling various data types, such as geographical and textual data, allowing for flexibility for diverse data processing needs. It also simplifies data input and output operations by offering functionalities for managing many different file-like objects. In addition, PyHelpers facilitates communication with relational databases, such as PostgreSQL and Microsoft SQL Server. This capability greatly smooths data integration with database systems through efficient data storage and retrieval.

With its comprehensive suite of practical tools, PyHelpers simplifies complex data processing tasks and boosts productivity. It is ready to serve as an essential resource for effective data manipulation, management and analysis for anyone working with data in Python.

Chapter 2

Installation

2.1 Using `pip`

Note

- If you are using a virtual environment, ensure it is activated.
- Use the `--upgrade` (or `-U`) option with pip install to get the latest stable release.

To install the latest release of PyHelpers from PyPI via pip:

```
> pip install --upgrade pyhelpers
```

To install the latest development version of PyHelpers from GitHub:

```
> pip install --upgrade git+https://github.com/mikeqfu/pyhelpers.git
```

2.2 Verification

To verify the installation of PyHelpers, try importing the package in an interpreter shell:

```
>>> import pyhelpers
>>> pyhelpers.__version__  # Check the latest version
```

```
The latest version is: 2.5.0
```

Note

- Not all dependencies of PyHelpers are installed automatically to optimise installation requirements. If you encounter a ModuleNotFoundError or an ImportError, install the missing package(s) as indicated in the error message.
- When using the package, Windows users might face issues with pip failing to install some packages. In such cases, try installing the wheel (.whl) files instead. Refer to Christoph Gohlke's homepage for essential and downloadable wheel files.

- For more general instructions on installing Python packages, refer to the Installing Packages guide.

Chapter 3

Quick Start

This quick-start tutorial provides some simple examples for each of the *Subpackages*. These examples demonstrate the capabilities of pyhelpers in assisting with data manipulation tasks.

3.1 Preparation - Create a data set

< Back to Top | *Next >*

Let's start by creating an example data set using NumPy and Pandas, both of which are installed automatically with pyhelpers, as they are dependencies.

To demonstrate, we will use the numpy.random.rand() function to generate a 8×8 NumPy array of random samples drawn from a standard uniform distribution, naming the array `random_array`:

```
>>> import numpy as np  # Import NumPy and abbreviate it to 'np'
>>> np.random.seed(0)  # Ensure that the generated array data is reproducible
>>> random_array = np.random.rand(8, 8)
>>> random_array
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ,
        0.64589411, 0.43758721, 0.891773  ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492, 0.56804456,
        0.92559664, 0.07103606, 0.0871293 ],
       [0.0202184 , 0.83261985, 0.77815675, 0.87001215, 0.97861834,
        0.79915856, 0.46147936, 0.78052918],
       [0.11827443, 0.63992102, 0.14335329, 0.94466892, 0.52184832,
        0.41466194, 0.26455561, 0.77423369],
       [0.45615033, 0.56843395, 0.0187898 , 0.6176355 , 0.61209572,
        0.616934  , 0.94374808, 0.6818203 ],
       [0.3595079 , 0.43703195, 0.6976312 , 0.06022547, 0.66676672,
        0.67063787, 0.21038256, 0.1289263 ],
       [0.31542835, 0.36371077, 0.57019677, 0.43860151, 0.98837384,
        0.10204481, 0.20887676, 0.16130952],
       [0.65310833, 0.2532916 , 0.46631077, 0.24442559, 0.15896958,
        0.11037514, 0.65632959, 0.13818295]])
>>> random_array.shape  # Check the shape of the array
(8, 8)
```

Next, we will use pandas.DataFrame() to transform `random_array` into a Pandas DataFrame, naming it `data_frame`:

```
>>> import pandas as pd  # Import Pandas and abbreviate it to 'pd'
>>> data_frame = pd.DataFrame(random_array, columns=['col_' + str(x) for x in range(8)])
>>> data_frame
      col_0     col_1     col_2  ...     col_5     col_6     col_7
0  0.548814  0.715189  0.602763  ...  0.645894  0.437587  0.891773
1  0.963663  0.383442  0.791725  ...  0.925597  0.071036  0.087129
2  0.020218  0.832620  0.778157  ...  0.799159  0.461479  0.780529
3  0.118274  0.639921  0.143353  ...  0.414662  0.264556  0.774234
4  0.456150  0.568434  0.018790  ...  0.616934  0.943748  0.681820
5  0.359508  0.437032  0.697631  ...  0.670638  0.210383  0.128926
6  0.315428  0.363711  0.570197  ...  0.102045  0.208877  0.161310
7  0.653108  0.253292  0.466311  ...  0.110375  0.656330  0.138183
[8 rows x 8 columns]
```

See also

- The example of *saving data as a Pickle file*.

3.2 Altering display settings

< Previous | *Back to Top* | *Next >*

The `pyhelpers.settings` module can be used to alter frequently-used parameters (of GDAL, Matplotlib, NumPy and Pandas) to customise the working environment.

For example, we can apply the `np_preferences()` function with its default parameters to get a neater view of `random_array`:

```
>>> from pyhelpers.settings import np_preferences
>>> # To round the numbers to four decimal places
>>> np_preferences()  # By default, reset=False and precision=4
>>> random_array
array([[0.5488, 0.7152, 0.6028, 0.5449, 0.4237, 0.6459, 0.4376, 0.8918],
       [0.9637, 0.3834, 0.7917, 0.5289, 0.5680, 0.9256, 0.0710, 0.0871],
       [0.0202, 0.8326, 0.7782, 0.8700, 0.9786, 0.7992, 0.4615, 0.7805],
       [0.1183, 0.6399, 0.1434, 0.9447, 0.5218, 0.4147, 0.2646, 0.7742],
       [0.4562, 0.5684, 0.0188, 0.6176, 0.6121, 0.6169, 0.9437, 0.6818],
       [0.3595, 0.4370, 0.6976, 0.0602, 0.6668, 0.6706, 0.2104, 0.1289],
       [0.3154, 0.3637, 0.5702, 0.4386, 0.9884, 0.1020, 0.2089, 0.1613],
       [0.6531, 0.2533, 0.4663, 0.2444, 0.1590, 0.1104, 0.6563, 0.1382]])
```

To reset the display settings, set `reset=True` to revert to default values:

```
>>> np_preferences(reset=True)
>>> random_array
array([[0.54881350, 0.71518937, 0.60276338, 0.54488318, 0.42365480,
        0.64589411, 0.43758721, 0.89177300],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492, 0.56804456,
        0.92559664, 0.07103606, 0.08712930],
       [0.02021840, 0.83261985, 0.77815675, 0.87001215, 0.97861834,
        0.79915856, 0.46147936, 0.78052918],
       [0.11827443, 0.63992102, 0.14335329, 0.94466892, 0.52184832,
        0.41466194, 0.26455561, 0.77423369],
       [0.45615033, 0.56843395, 0.01878980, 0.61763550, 0.61209572,
```

(continues on next page)

(continued from previous page)

```
       0.61693400, 0.94374808, 0.68182030],
      [0.35950790, 0.43703195, 0.69763120, 0.06022547, 0.66676672,
       0.67063787, 0.21038256, 0.12892630],
      [0.31542835, 0.36371077, 0.57019677, 0.43860151, 0.98837384,
       0.10204481, 0.20887676, 0.16130952],
      [0.65310833, 0.25329160, 0.46631077, 0.24442559, 0.15896958,
       0.11037514, 0.65632959, 0.13818295]])
```

Note

- The *np_preferences()* function inherits the functionality of numpy.set_printoptions(), with some modifications.

Similarly, the *pd_preferences()* function alters a few Pandas options and settings, such as display representation and maximum number of columns when displaying a DataFrame. Applying the function with default parameters allows us to view all eight columns with precision set to four decimal places.

```
>>> from pyhelpers.settings import pd_preferences
>>> pd_preferences()  # By default, reset=False and precision=4
>>> data_frame
   col_0  col_1  col_2  col_3  col_4  col_5  col_6  col_7
0 0.5488 0.7152 0.6028 0.5449 0.4237 0.6459 0.4376 0.8918
1 0.9637 0.3834 0.7917 0.5289 0.5680 0.9256 0.0710 0.0871
2 0.0202 0.8326 0.7782 0.8700 0.9786 0.7992 0.4615 0.7805
3 0.1183 0.6399 0.1434 0.9447 0.5218 0.4147 0.2646 0.7742
4 0.4562 0.5684 0.0188 0.6176 0.6121 0.6169 0.9437 0.6818
5 0.3595 0.4370 0.6976 0.0602 0.6668 0.6706 0.2104 0.1289
6 0.3154 0.3637 0.5702 0.4386 0.9884 0.1020 0.2089 0.1613
7 0.6531 0.2533 0.4663 0.2444 0.1590 0.1104 0.6563 0.1382
```

To reset the settings, use the parameter `reset`; setting it to `True` reverts to default values, and setting it to `'all'` resets all Pandas options:

```
>>> pd_preferences(reset=True)
>>> data_frame
      col_0     col_1     col_2  ...     col_5     col_6     col_7
0  0.548814  0.715189  0.602763  ...  0.645894  0.437587  0.891773
1  0.963663  0.383442  0.791725  ...  0.925597  0.071036  0.087129
2  0.020218  0.832620  0.778157  ...  0.799159  0.461479  0.780529
3  0.118274  0.639921  0.143353  ...  0.414662  0.264556  0.774234
4  0.456150  0.568434  0.018790  ...  0.616934  0.943748  0.681820
5  0.359508  0.437032  0.697631  ...  0.670638  0.210383  0.128926
6  0.315428  0.363711  0.570197  ...  0.102045  0.208877  0.161310
7  0.653108  0.253292  0.466311  ...  0.110375  0.656330  0.138183
[8 rows x 8 columns]
```

Note

- The functions in *pyhelpers.settings* handle only a few selected parameters based on the author's preferences. Feel free to modify the source code to suit your needs.

3.3 Specify a directory or file path

< Previous | Back to Top | Next >

The *pyhelpers.dirs* module aids in manipulating directories. For instance, the *cd()* function returns the absolute path to the current working directory, or to a specified subdirectory or file within it:

```
>>> from pyhelpers.dirs import cd
>>> from pyhelpers.dirs import normalize_pathname  # Optional, for printing uniform pathnames
>>> import os
>>> cwd = cd()  # The current working directory
>>> # Relative path of `cwd` to the current working directory
>>> rel_path_cwd = os.path.relpath(cwd)
>>> print(rel_path_cwd)
.
```

To specify a path to a temporary folder named "pyhelpers_tutorial":

```
>>> # Name of a temporary folder for this tutorial
>>> dir_name = "pyhelpers_tutorial"
>>> # Path to the folder "pyhelpers_tutorial"
>>> path_to_dir = cd(dir_name)
>>> # Relative path of the directory
>>> rel_dir_path = os.path.relpath(path_to_dir)
>>> print(rel_dir_path)
pyhelpers_tutorial
```

Check whether the directory "pyhelpers_tutorial" exists:

```
>>> print(f'Does the directory "{rel_dir_path}" exist? {os.path.isdir(path_to_dir)}')
Does the directory "pyhelpers_tutorial" exist? False
```

If the directory "pyhelpers_tutorial" does not exist, set the parameter mkdir=True to create it:

```
>>> # Set `mkdir` to `True` to create the "pyhelpers_tutorial" folder
>>> path_to_dir = cd(dir_name, mkdir=True)
>>> # Check again whether the directory "pyhelpers_tutorial" exists
>>> print(f'Does the directory "{rel_dir_path}" exist? {os.path.isdir(path_to_dir)}')
Does the directory "pyhelpers_tutorial" exist? True
```

When we specify a sequence of names (in order with a filename being the last), the *cd()* function would assume that all the names prior to the filename are folder names, which specify a path to the file. For example, to specify a path to a file named "quick_start.dat" within the "pyhelpers_tutorial" folder:

```
>>> # Name of the file
>>> filename = "quick_start.dat"
>>> # Path to the file named "quick_start.dat"
>>> path_to_file = cd(dir_name, filename)  # path_to_file = cd(path_to_dir, filename)
>>> # Relative path of the file "quick_start.dat"
>>> rel_file_path = os.path.relpath(path_to_file)
>>> # [Optional]
>>> rel_file_path = normalize_pathname(os.path.relpath(path_to_file))
>>> print(rel_file_path)
pyhelpers_tutorial/quick_start.dat
```

If any directories in the specified path do not exist, setting `mkdir=True` will create them. For example, to specify a data directory named `"data"` within the `"pyhelpers_tutorial"` folder:

```
>>> # Path to the data directory
>>> data_dir = cd(dir_name, "data")  # equivalent to `cd(path_to_dir, "data")`
>>> # Relative path of the data directory
>>> rel_data_dir = os.path.relpath(data_dir)
>>> # [Optional]
>>> rel_data_dir = normalize_pathname(os.path.relpath(data_dir))
>>> print(rel_data_dir)
pyhelpers_tutorial/data
```

We can then use the `is_dir()` function to check if `data_dir` (or `rel_data_dir`) is a directory:

```
>>> from pyhelpers.dirs import is_dir
>>> # Check if `rel_data_dir` is a directory
>>> print(f'Does `rel_data_dir` specify a directory path? {is_dir(rel_data_dir)}')
Does `rel_data_dir` specify a directory path? True
>>> # Check if the data directory exists
>>> print(f'Does the directory "{rel_data_dir}" exist? {os.path.isdir(rel_data_dir)}')
Does the directory "pyhelpers_tutorial/data" exist? False
```

For another example, to specify a path to a Pickle file, named `"dat.pkl"`, in the directory `"pyhelpers_tutorial/data"`:

```
>>> # Name of the Pickle file
>>> pickle_filename = "dat.pkl"
>>> # Path to the Pickle file
>>> path_to_pickle = cd(data_dir, pickle_filename)
>>> # Relative path of the Pickle file
>>> rel_pickle_path = os.path.relpath(path_to_pickle)
>>> # [Optional]
>>> rel_pickle_path = normalize_pathname(os.path.relpath(path_to_pickle))
>>> print(rel_pickle_path)
pyhelpers_tutorial/data/dat.pkl
```

Check `rel_pickle_path` (or `path_to_pickle`):

```
>>> # Check if `rel_pickle_path` is a directory
>>> print(f'Is `rel_pickle_path` a directory? {os.path.isdir(rel_pickle_path)}')
Is `rel_pickle_path` a directory? False
>>> # Check if the file "dat.pkl" exists
>>> print(f'Does the file "{rel_pickle_path}" exist? {os.path.isfile(rel_pickle_path)}')
Does the file "pyhelpers_tutorial/data/dat.pkl" exist? False
```

Let's now set `mkdir=True` to create any missing directories:

```
>>> path_to_pickle = cd(data_dir, pickle_filename, mkdir=True)
>>> rel_data_dir = os.path.relpath(data_dir)
>>> # [Optional]
>>> rel_data_dir = normalize_pathname(os.path.relpath(data_dir))
>>> # Check again if the data directory exists
>>> print(f'Does the directory "{rel_data_dir}" exist? {os.path.isdir(rel_data_dir)}')
Does the directory "pyhelpers_tutorial/data" exist? True
>>> # Check again if the file "dat.pkl" exists
>>> print(f'Does the file "{rel_pickle_path}" exist? {os.path.isfile(rel_pickle_path)}')
Does the file "pyhelpers_tutorial/data/dat.pkl" exist? False
```

[See also the example of *saving data as a Pickle file*.]

To delete the directory "pyhelpers_tutorial" (including all its contents), we can use the *delete_dir()* function:

```
>>> from pyhelpers.dirs import delete_dir
>>> # Delete the "pyhelpers_tutorial" directory
>>> delete_dir(path_to_dir, verbose=True)
To delete the directory "./pyhelpers_tutorial/" (Not empty)
? [No]|Yes: yes
Deleting "./pyhelpers_tutorial/" ... Done.
```

3.4 Save and load data with Pickle files

< Previous | *Back to Top* | *Next >*

The *pyhelpers.store* module can facilitate tasks such as saving and loading data using file-like objects in common formats such as CSV, JSON, and Pickle.

To demonstrate, let's save the data_frame created earlier (see *Preparation*) as a Pickle file using *save_pickle()*, and later retrieve it using *load_pickle()*. We'll use path_to_pickle from the directory specified in the *Specify a directory or a file path* section:

```
>>> from pyhelpers.store import save_pickle, load_pickle
>>> # Save `data_frame` to "dat.pkl"
>>> save_pickle(data_frame, path_to_pickle, verbose=True)
Saving "dat.pkl" to "./pyhelpers_tutorial/data/" ... Done.
```

We can now retrieve/load the data from path_to_pickle and store it as df_retrieved:

```
>>> df_retrieved = load_pickle(path_to_pickle, verbose=True)
Loading "./pyhelpers_tutorial/data/dat.pkl" ... Done.
```

To verify if df_retrieved matches data_frame:

```
>>> print(f'`df_retrieved` matches `data_frame`? {df_retrieved.equals(data_frame)}')
`df_retrieved` matches `data_frame`? True
```

Before proceeding, let's delete the Pickle file (i.e. path_to_pickle) and the associated directory that's been created:

```
>>> delete_dir(path_to_dir, verbose=True)
To delete the directory "./pyhelpers_tutorial/" (Not empty)
? [No]|Yes: yes
Deleting "./pyhelpers_tutorial/" ... Done.
```

Note

- In the *pyhelpers.store* module, some functions such as *save_spreadsheet()* and *save_spreadsheets()* may require openpyxl, XlsxWriter or xlrd, which are not essential dependencies for the base installation of pyhelpers. We could install them when needed via an appropriate method such as pip install.

3.5 Convert coordinates between OSGB36 and WGS84

< Previous | Back to Top | Next >

The *pyhelpers.geom* module can assist us in manipulating geometric and geographical data. For example, we can use the *osgb36_to_wgs84()* function to convert geographical coordinates from OSGB36 (British national grid) eastings and northings to WGS84 longitudes and latitudes:

```
>>> from pyhelpers.geom import osgb36_to_wgs84
>>> # Convert coordinates (easting, northing) to (longitude, latitude)
>>> easting, northing = 530039.558844, 180371.680166  # London
>>> longitude, latitude = osgb36_to_wgs84(easting, northing)
>>> (longitude, latitude)
(-0.12764738750268856, 51.507321895400686)
```

We can also use the function for bulk conversion of an array of OSGB36 coordinates:

```
>>> from pyhelpers._cache import example_dataframe
>>> example_df = example_dataframe(osgb36=True)
>>> example_df
                Easting       Northing
City
London      530039.558844  180371.680166
Birmingham  406705.887014  286868.166642
Manchester  383830.039036  398113.055831
Leeds       430147.447354  433553.327117
>>> xy_array = example_df.to_numpy()
>>> eastings, northings = xy_array.T
>>> lonlat_array = osgb36_to_wgs84(eastings, northings, as_array=True)
>>> lonlat_array
array([[-0.12764739, 51.50732190],
       [-1.90269109, 52.47969920],
       [-2.24511479, 53.47948920],
       [-1.54379409, 53.79741850]])
```

Similarly, conversion from (longitude, latitude) back to (easting, northing) can be implemented using the function *wgs84_to_osgb36()*:

```
>>> from pyhelpers.geom import wgs84_to_osgb36
>>> longitudes, latitudes = lonlat_array.T
>>> xy_array_ = wgs84_to_osgb36(longitudes, latitudes, as_array=True)
>>> xy_array_
array([[530039.55972534, 180371.67967567],
       [406705.88783629, 286868.16621896],
       [383830.03985454, 398113.05550332],
       [430147.44820845, 433553.32682598]])
```

Note

- Conversion of coordinates between different systems may inevitably introduce minor errors, which are typically negligible.

Check if `xy_array_` is almost equal to `xy_array`:

```
>>> eq_res = np.array_equal(np.round(xy_array, 2), np.round(xy_array_, 2))
>>> print(f'Is `xy_array_` almost equal to `xy_array`? {eq_res}')
Is `xy_array_` almost equal to `xy_array`? True
```

3.6 Find similar texts

< Previous | *Back to Top* | *Next >*

The *pyhelpers.text* module can assist us in manipulating textual data. For example, suppose we have a word `'angle'`, which is stored in a str-type variable named `word`, and a list of words, which is stored in a list-type variable named `lookup_list`; if we'd like to find from the list a one that is most similar to `'angle'`, we can use the function *find_similar_str()*:

```
>>> from pyhelpers.text import find_similar_str
>>> word = 'angle'
>>> lookup_list = [
...     'Anglia',
...     'East Coast',
...     'East Midlands',
...     'North and East',
...     'London North Western',
...     'Scotland',
...     'South East',
...     'Wales',
...     'Wessex',
...     'Western']
>>> # Find the most similar word to 'angle'
>>> result_1 = find_similar_str(word, lookup_list)
>>> result_1
'Anglia'
```

By default, the function relies on difflib - a built-in Python module. Alternatively, we can make use of an open-source package, RapidFuzz, via setting the parameter `engine='rapidfuzz'` (or simply `engine='fuzz'`):

```
>>> # Find the most similar word to 'angle' by using RapidFuzz
>>> result_2 = find_similar_str(word, lookup_list, engine='fuzz')
>>> result_2
'Anglia'
```

Note

- RapidFuzz is not a required dependency for basic pyhelpers functionality. We need to install it (e.g. `pip install rapidfuzz`) to make the function run successfully with setting `engine='rapidfuzz'` (or `engine='fuzz'`).

3.7 Download an image file

< Previous | *Back to Top* | *Next >*

The *pyhelpers.ops* module provides various helper functions for operations. For example, we can use *download_file_from_url()* to download files from URLs.

Let's now try to download the Python logo image from its official page. Firstly, we need to specify the URL of the image file:

```
>>> from pyhelpers.ops import download_file_from_url
>>> # URL of a .png file of the Python logo
>>> url = 'https://www.python.org/static/community_logos/python-logo-master-v3-TM.png'
```

Then, we need to specify a directory where we'd like to save the image file, and a filename for it; let's say we want to name the file `"python-logo.png"` and save it to the directory `"pyhelpers_tutorial/images"`:

```
>>> python_logo_filename = "python-logo.png"
>>> # python_logo_file_path = cd(dir_name, "images", python_logo_filename)
>>> python_logo_file_path = cd(path_to_dir, "images", python_logo_filename)
>>> # Download the .png file of the Python logo
>>> download_file_from_url(url, python_logo_file_path, verbose=False)
```

Note

- By default, `verbose=False` prevents output during download.
- Setting `verbose=True` (or `verbose=1`) requires an the package tqdm, which is not essential for installing pyhelpers. We can install it via `pip install tqdm` if necessary.

If we set `verbose=True` (given that tqdm is available in our working environment), the function will print out relevant information about the download progress as the file is being downloaded:

```
>>> download_file_from_url(url, python_logo_file_path, if_exists='replace', verbose=True)
Downloading "python-logo.png" 100%|██████████| 83.6k/83.6k | 403kB/s | ETA: 00:00
    Updating "python-logo.png" in "./pyhelpers_tutorial/images/" ... Done.

>>> # Use a different progress bar color
>>> download_file_from_url(
...     url, python_logo_file_path, if_exists='replace', pbar_color='green', verbose=True)
Downloading "python-logo.png" 100%|██████████| 83.6k/83.6k | 760kB/s | ETA: 00:00
        Updating "python-logo.png" in "./pyhelpers_tutorial/images/" ... Done.
```

Note

- *'403kB/s'* shown at the end of the output is an estimated speed of downloading the file, which varies depending on network conditions at the time of running the function.
- Setting `if_exists='replace'` (default) allows us to replace the image file that already exists at the specified destination.

Now let's have a look at the downloaded image file using Pillow:

```
>>> from PIL import Image
>>> python_logo = Image.open(python_logo_file_path)
>>> python_logo.show()
```

Figure 1: Python Logo (for illustrative purposes in this tutorial).

Note

- In Jupyter Notebook, we can use IPython.display.Image to display the image in the notebook by running `IPython.display.Image(python_logo_file_path)`.

To delete `"pyhelpers_tutorial"` and its subdirectories (including `"pyhelpers_tutorial/images"`), we can use the *`delete_dir()`* function again:

```
>>> delete_dir(path_to_dir, confirmation_required=False, verbose=True)
Deleting "./pyhelpers_tutorial/" ... Done.
```

Setting the parameter `confirmation_required=False` can allow us to delete the directory straightaway without typing a `yes` to confirm the action. The confirmation prompt is actually implemented through the *`confirmed()`* function, which is also from the *`pyhelpers.ops`* module and can be helpful especially when we'd like to impose a manual confirmation before proceeding with certain actions. For example:

```
>>> from pyhelpers.ops import confirmed
>>> # We can specify any prompting message as to what needs to be confirmed.
>>> if confirmed(prompt="Continue? ..."):
...     print("OK! Go ahead.")
Continue? ... [No]|Yes: yes
OK! Go ahead.
```

Note

- The response to the *confirmed()* prompt is case-insensitive. It does not have to be exactly `Yes` for the function to return `True`; responses such as `yes`, `Y` or `ye` will also work. Conversely, responses such as `no` or `n` will return `False`.
- The *confirmed()* function also includes a parameter `confirmation_required`, which defaults to `True`. If we set `confirmation_required=False`, no confirmation is required, and the function will become ineffective and return `True`.

3.8 Work with a PostgreSQL server

< Previous | *Back to Top* | *Next >*

The *pyhelpers.dbms* module provides a convenient way of communicating with databases, such as PostgreSQL and Microsoft SQL Server.

For example, the *PostgreSQL* class can assist us in executing basic SQL statements on a PostgreSQL database server. To demonstrate its functionality, let's start by importing the class:

```
>>> from pyhelpers.dbms import PostgreSQL
```

3.8.1 Connect to a database

Now, we can create an instance of the class *PostgreSQL* to connect to a PostgreSQL server by specifying key parameters, including `host`, `port`, `username`, `database_name` and `password`.

Note

- If `host`, `port`, `username` and `database_name` are unspecified, their associated default attributes (namely, *DEFAULT_HOST*, *DEFAULT_PORT*, *DEFAULT_USERNAME* and *DEFAULT_DATABASE*) are used to instantiate the class, in which case we will connect to the default PostgreSQL server (as is installed on a PC).
- If the specified `database_name` does not exist, it will be automatically created during the class instantiation.
- Usually, we do not specify the parameter `password` explicitly in our code. Leaving it as `None` by default, we will be prompted to type it manually when instantiating the class.

For example, let's create an instance named `postgres` and establish a connection with a database named "*pyhelpers_tutorial*", hosted on the default PostgreSQL server:

```
>>> database_name = "pyhelpers_tutorial"
>>> postgres = PostgreSQL(database_name=database_name, verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "pyhelpers_tutorial" ... Done.
Connecting postgres:***@localhost:5432/pyhelpers_tutorial ... Successfully.
```

We can use pgAdmin, the most popular graphical management tool for PostgreSQL, to check whether the database "*pyhelpers_tutorial*" exists in the **Databases** tree of the default server, as

illustrated in Figure 2:

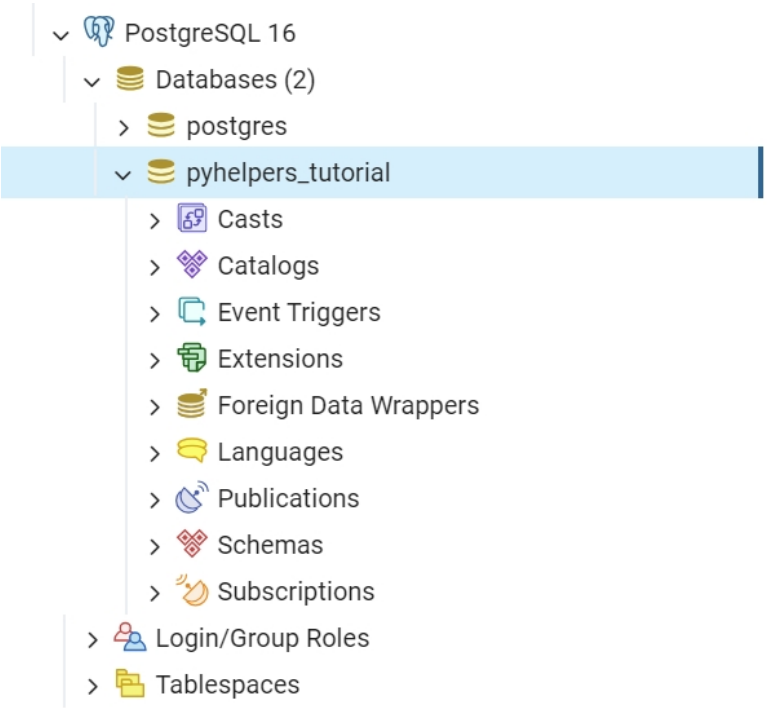

Figure 2: The database "*pyhelpers_tutorial*".

Alternatively, we can use the `database_exists()` method:

```
>>> res = postgres.database_exists(database_name)
>>> print(f'Does the database "{database_name}" exist? {res}')
Does the database "pyhelpers_tutorial" exist? True
>>> print(f'We are currently connected to the database "{postgres.database_name}".')
We are currently connected to the database "pyhelpers_tutorial".
```

On the same server, we can create multiple databases. For example, let's create another database named "*pyhelpers_tutorial_alt*" using the `create_database()` method:

```
>>> database_name_ = "pyhelpers_tutorial_alt"
>>> postgres.create_database(database_name_, verbose=True)
Creating a database: "pyhelpers_tutorial_alt" ... Done.
```

As we can see in Figure 3, the database "*pyhelpers_tutorial_alt*" has now been added to the default **Databases** tree:

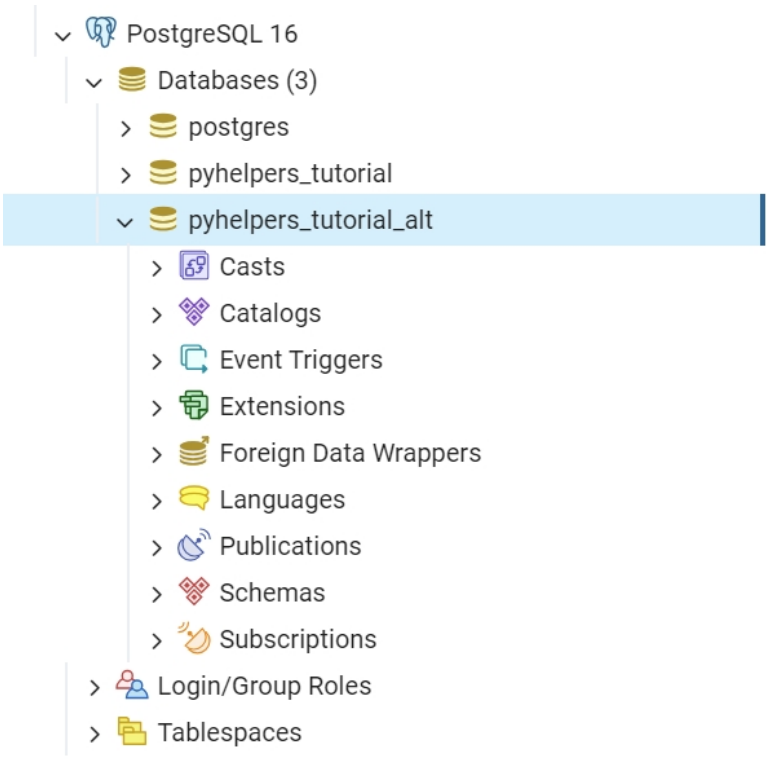

Figure 3: The database "*pyhelpers_tutorial_alt*".

Note

- When a new database is created, the instance `postgres` disconnects from the current database and connects to the new one.

Check whether "*pyhelpers_tutorial_alt*" is the currently connected database:

```
>>> res = postgres.database_exists("pyhelpers_tutorial_alt")
>>> print(f'Does the database "{database_name_}" exist? {res}')
Does the database "pyhelpers_tutorial_alt" exist? True
>>> print(f'We are currently connected to the database "{postgres.database_name}".')
We are currently connected to the database "pyhelpers_tutorial_alt".
```

To reconnect to "*pyhelpers_tutorial*" (`database_name`), we can use the *`connect_database()`* method:

```
>>> postgres.connect_database(database_name, verbose=True)
Connecting postgres:***@localhost:5432/pyhelpers_tutorial ... Successfully.
>>> print(f'We are currently connected to the database "{postgres.database_name}".')
We are now connected with the database "pyhelpers_tutorial".
```

3.8.2 Import data into a database

With the established connection to the database, we can use the *`import_data()`* method to import the `data_frame` (created in the *Preparation* section) into a table named "*df_table*" under the default schema "*public*":

```
>>> table_name = "df_table"
>>> postgres.import_data(data=data_frame, table_name=table_name, verbose=True)
To import data into the table "public"."df_table" at postgres:***@localhost:5432/...
? [No]|Yes: yes
Importing the data into "public"."df_table" ... Done.
```

We should now see the table in pgAdmin, as shown in Figure 4:

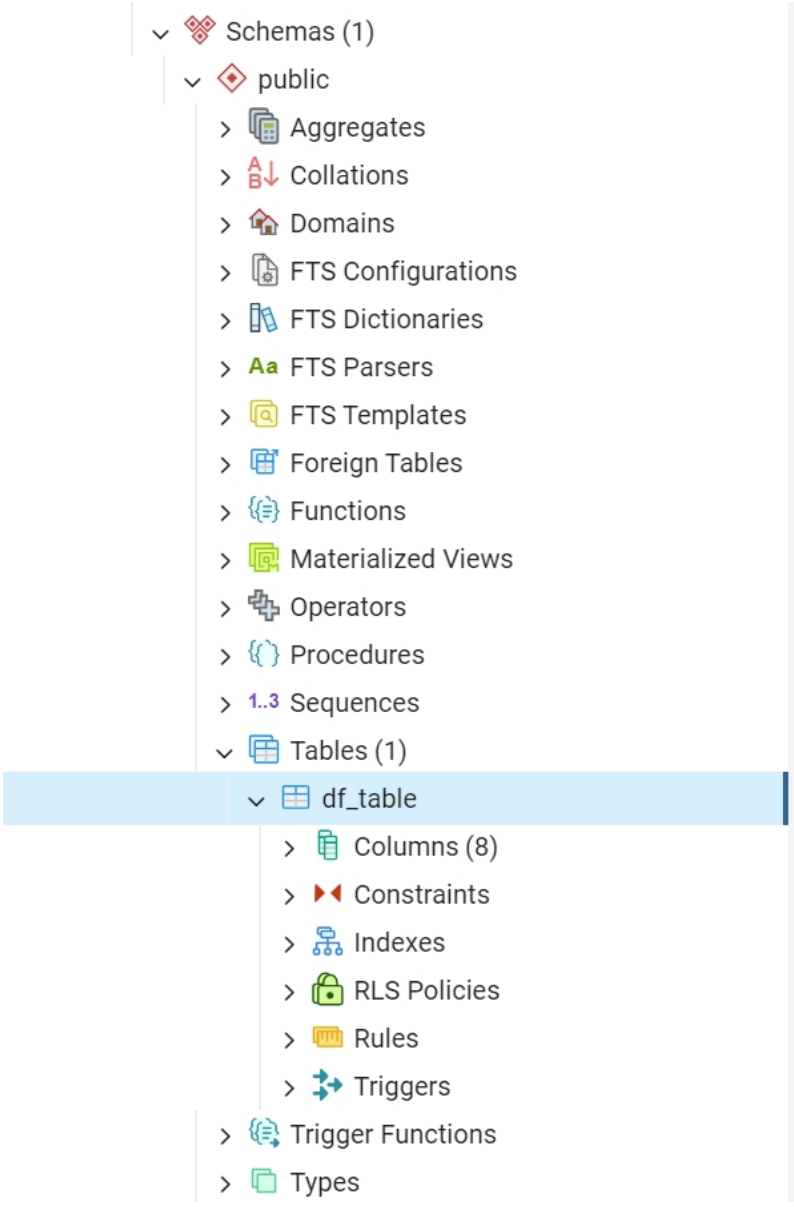

Figure 4: The table "*public*"."*df_table*".

The method *import_data()* relies on the method pandas.DataFrame.to_sql(), with the `method` parameter set to `'multi'` by default. Optionally, we can also use the method *psql_insert_copy()* to significantly speed up data import, especially for large data sets.

Let's now try to import the same data into a table named "*df_table_alt*" by setting `method=postgres.psql_insert_copy`:

```
>>> table_name_ = "df_table_alt"
>>> postgres.import_data(
...     data=data_frame, table_name=table_name_, method=postgres.psql_insert_copy,
...     verbose=True)
To import data into the table "public"."df_table_alt" at postgres:***@localhost:5432/...
? [No]|Yes: yes
Importing the data into "public"."df_table_alt" ... Done.
```

In pgAdmin, we can see the table has been added to the **Tables** list, as shown in Figure 5:

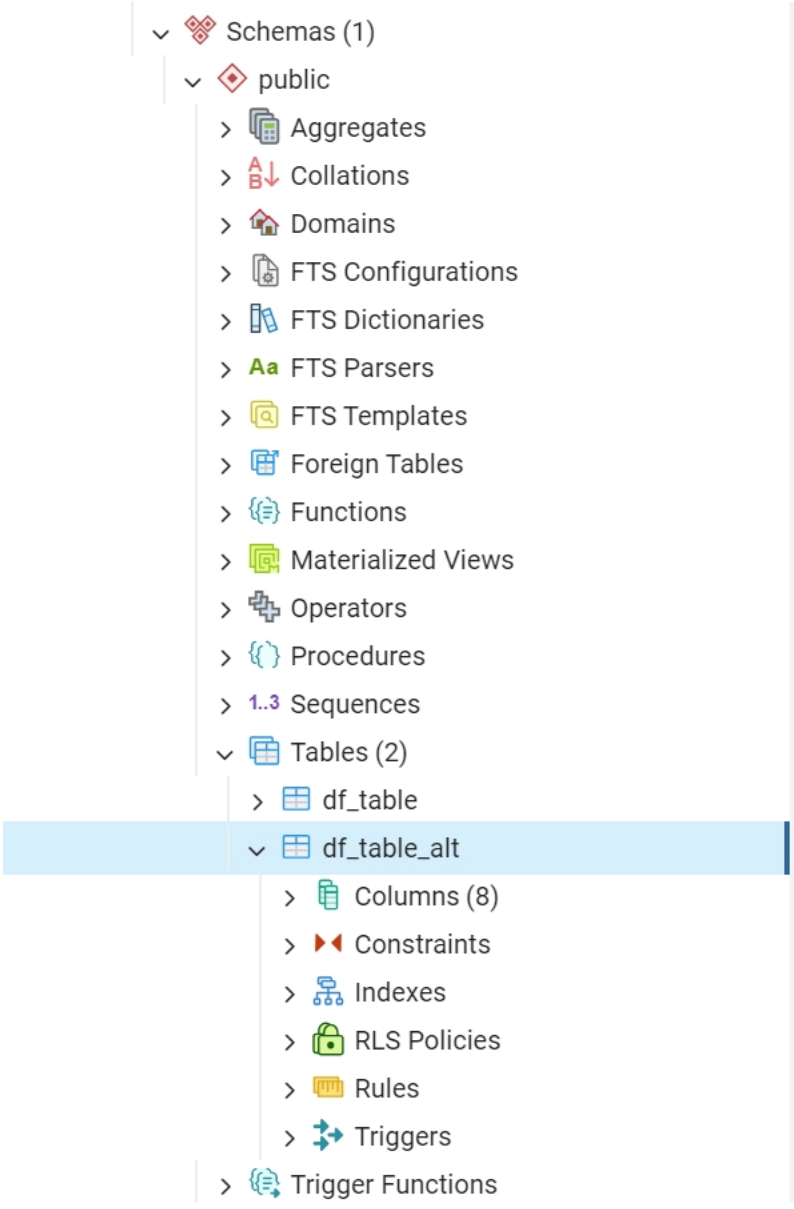

Figure 5: The table "*public*"."*df_table_alt*".

3.8.3 Fetch data from a database

To retrieve the imported data, we can use the *read_table()* method:

```
>>> df_retrieval_1 = postgres.read_table(table_name)
>>> res = df_retrieval_1.equals(data_frame)
>>> print(f"Is `df_retrieval_1` equal to `data_frame`? {res}")
Is `df_retrieval_1` equal to `data_frame`? True
```

Alternatively, we can use the *read_sql_query()* method, which serves as a more flexible way of reading/querying data. It takes PostgreSQL query statements and can be much faster for large tables.

Let's try this method to fetch the same data from the table "*df_table_alt*":

```
>>> df_retrieval_2 = postgres.read_sql_query('SELECT * FROM "public"."df_table_alt"')
>>> res = df_retrieval_2.round(8).equals(df_retrieval_1.round(8))
>>> print(f"Is `df_retrieval_2` equal to `df_retrieval_1`? {res}")
Is `df_retrieval_2` equal to `df_retrieval_1`? True
```

Note

- For the *read_sql_query()* method, any PostgreSQL query statement that is passed to `sql_query` should NOT end with `';'`.

3.8.4 Delete data

Before we leave this notebook, let's clear up the databases and tables we've created.

We can delete/drop a table (e.g. "*df_table*") using the *drop_table()* method:

```
>>> postgres.drop_table(table_name=table_name, verbose=True)
To drop the table "public"."df_table" from postgres:***@localhost:5432/pyhelpers_tutorial
? [No]|Yes: yes
Dropping "public"."df_table" ... Done.
```

To delete/drop a database, we can use the *drop_database()* method:

```
>>> # Drop "pyhelpers_tutorial" (i.e. the currently connected database)
>>> postgres.drop_database(verbose=True)
To drop the database "pyhelpers_tutorial" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "pyhelpers_tutorial" ... Done.
>>> # Drop "pyhelpers_tutorial_alt"
>>> postgres.drop_database(database_name=database_name_, verbose=True)
To drop the database "pyhelpers_tutorial_alt" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "pyhelpers_tutorial_alt" ... Done.
```

Check the currently connected database:

```
>>> print(f"We are currently connected with the database \"{postgres.database_name}\".")
We are currently connected with the database "postgres".
```

Now we have removed all the databases created in this notebook and restored the PostgreSQL server to its original status.

< *Previous* | *Back to Top*

Any issues regarding the use of pyhelpers are welcome and should be logged/reported onto Issue Tracker.

For more details and examples, check *Subpackages*.

Chapter 4

Subpackages

The current release includes the following subpackages, with each containing a number of utilities:

1anti-flashwhitewhite

settings	Settings of working environment.
dirs	Utilities for directory and file operations, navigation and management.
ops	Miscellaneous operations.
store	Performing operations on file-like objects.
geom	Manipulation of geometric/geographical data.
text	Manipulation of textual data.
dbms	Communication with databases.
viz	Visualization utilities for geospatial and map-related operations.

4.1 settings

Settings of working environment.

4.1.1 Configurations

GDAL/OGR

1anti-flashwhitewhite

gdal_configurations([reset, ...])	Alters some default configuration options of GDAL/OGR drivers.

gdal_configurations

pyhelpers.settings.**gdal_configurations**(*reset=False*, *max_tmpfile_size=None*, *interleaved_reading=True*, *custom_indexing=False*, *compress_nodes=True*)

Alters some default configuration options of GDAL/OGR drivers.

Parameters

- **reset** (*bool*) – Whether to reset to default settings; defaults to `False`.
- **max_tmpfile_size** (*int | None*) – Maximum size of the temporary file; defaults to `None`.
- **interleaved_reading** (*bool*) – Whether to enable interleaved reading; defaults to `True`.
- **custom_indexing** (*bool*) – Whether to enable custom indexing; defaults to `False`.
- **compress_nodes** (*bool*) – Whether to compress nodes in temporary database; defaults to `True`.

Examples:

```
>>> from pyhelpers.settings import gdal_configurations
>>> gdal_configurations()
```

Note

These configurations are particularly useful when working with GDAL to process large PBF files. For instance, these settings are applied by default in the pydriosm package for handling OpenStreetMap data in PBF format.

See also

- OpenStreetMap XML and PBF
- pydriosm Documentation

4.1.2 Preferences

Matplotlib

1anti-flashwhitewhite

mpl_preferences([reset, backend, font_name, ...])	Alters some Matplotlib parameters for plotting.
np_preferences([reset, precision, ...])	Alters some default parameters for displaying NumPy arrays.
pd_preferences([reset, max_columns, ...])	Alters parameters of some frequently-used Pandas options and settings for displaying pandas dataframes.

mpl_preferences

pyhelpers.settings.**mpl_preferences**(*reset=False*, *backend=None*, *font_name='Times New Roman'*, *font_size=13*, *legend_spacing=0.7*, *fig_style=None*)

Alters some Matplotlib parameters for plotting.

This function allows customizing various Matplotlib parameters such as backend, font settings, legend spacing and figure style.

Parameters

- **reset** (*bool*) – Whether to reset all parameters to their default settings; defaults to `False`.
- **backend** (*str | None*) – Specify the backend used for rendering and GUI integration; defaults to `None`.
- **font_name** (*str | None*) – Name of the font to be used; defaults to `'Times New Roman'`.
- **font_size** (*int | float*) – Font size; defaults to `13`.
- **legend_spacing** (*float | int*) – Spacing between labels in the plot legend; defaults to `0.7`.
- **fig_style** (*str | None*) – Style of the figure; defaults to `None`.

Examples:

```
>>> import numpy as np
>>> import matplotlib
>>> matplotlib.use('TkAgg')
>>> import matplotlib.pyplot as plt
>>> np.random.seed(0)
>>> random_array = np.random.rand(1000, 2)
>>> random_array
array([[0.5488135 , 0.71518937],
       [0.60276338, 0.54488318],
       [0.4236548 , 0.64589411],
       ...,
       [0.41443887, 0.79128155],
       [0.72119811, 0.48010781],
       [0.64386404, 0.50177313]])
>>> def example_plot(arr):
...     fig = plt.figure(constrained_layout=True)
...     ax = fig.add_subplot(aspect='equal', adjustable='box')
...     ax.scatter(arr[:500, 0], arr[:500, 1], label='Group0')
...     ax.scatter(arr[500:, 0], arr[500:, 1], label='Group1')
...     ax.legend(frameon=False, bbox_to_anchor=(1.0, 0.95))
...     fig.show()
>>> example_plot(random_array)
>>> # from pyhelpers.store import save_fig
>>> # save_fig("docs/source/_images/settings-mpl_preferences-demo-1.svg", verbose=True)
>>> # save_fig("docs/source/_images/settings-mpl_preferences-demo-1.pdf", verbose=True)
```

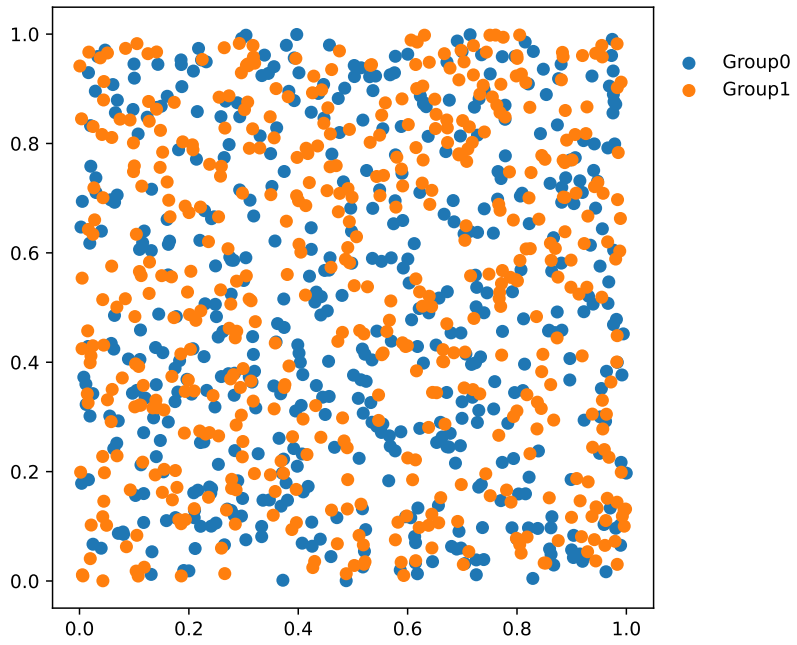

Figure 6: An example figure, before applying the function *mpl_preferences()*.

```
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(fig_style='ggplot')
>>> example_plot(random_array)
>>> # path_to_fig_ = "docs/source/_images/settings-mpl_preferences-demo-2"
>>> # save_fig(f"{path_to_fig_}.svg", verbose=True)
>>> # save_fig(f"{path_to_fig_}.pdf", verbose=True)
```

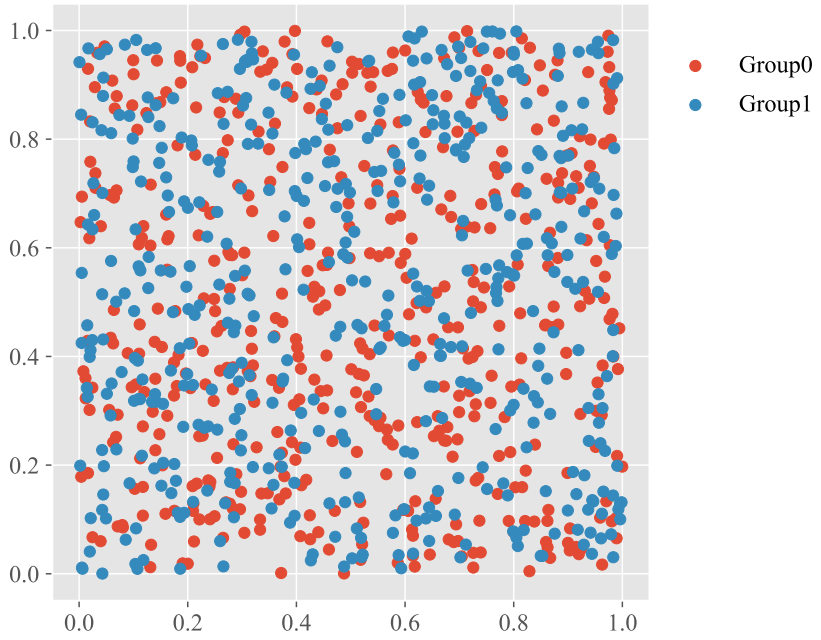

Figure 7: After applying the function *mpl_preferences()*.

Reset to default settings:

```
>>> mpl_preferences(reset=True)
>>> # The altered parameters are now set to their default values.
>>> example_plot(random_array)
```

(The above code should display the same figure as Figure 6.)

np_preferences

pyhelpers.settings.**np_preferences**(*reset=False*, *precision=4*, *head_tail=5*, *line_char=120*, *formatter=None*, ***kwargs*)

Alters some default parameters for displaying NumPy arrays.

This function allows customizing the display options for NumPy arrays, including decimal precision, summary at the beginning and end of each dimension, line width for inserting line breaks and optional custom formatting.

Parameters

- **reset** (*bool*) – Whether to reset to the default print options set by the function numpy.set_printoptions(); defaults to `False`.
- **precision** (*int*) – Number of decimal points to display, corresponding to `precision` of the function numpy.set_printoptions(); defaults to `4`.
- **line_char** (*int*) – Number of characters per line for inserting line breaks, corresponding to `linewidth` of the function numpy.set_printoptions(); defaults to `120`.
- **head_tail** (*int*) – Number of array items to summarize at the beginning (head) and end (tail) of each dimension, corresponding to `edgeitems` of the function numpy.set_printoptions(); defaults to `5`.
- **formatter** (*dict | None*) – Custom format specification, corresponding to `formatter` of the function numpy.set_printoptions(); if `formatter=None` (default), empty decimal places are filled with zeros for the specified `precision`.
- **kwargs** – [Optional] Additional parameters for the function numpy.set_printoptions().

Examples:

```
>>> import numpy as np
>>> np.random.seed(0)
>>> random_array = np.random.rand(100, 100)
>>> random_array
array([[0.5488135 , 0.71518937, 0.60276338, ..., 0.02010755, 0.82894003,
        0.00469548],
       [0.67781654, 0.27000797, 0.73519402, ..., 0.25435648, 0.05802916,
        0.43441663],
       [0.31179588, 0.69634349, 0.37775184, ..., 0.86219152, 0.97291949,
        0.96083466],
       ...,
       [0.89111234, 0.26867428, 0.84028499, ..., 0.5736796 , 0.73729114,
        0.22519844],
       [0.26969792, 0.73882539, 0.80714479, ..., 0.94836806, 0.88130699,
```

(continues on next page)

(continued from previous page)

```
       0.1419334 ],
      [0.88498232, 0.19701397, 0.56861333, ..., 0.75842952, 0.02378743,
       0.81357508]])
>>> from pyhelpers.settings import np_preferences
>>> np_preferences(precision=2)
>>> random_array
array([[0.55, 0.72, 0.60, 0.54, 0.42, ..., 0.18, 0.59, 0.02, 0.83, 0.00],
       [0.68, 0.27, 0.74, 0.96, 0.25, ..., 0.49, 0.23, 0.25, 0.06, 0.43],
       [0.31, 0.70, 0.38, 0.18, 0.02, ..., 0.22, 0.10, 0.86, 0.97, 0.96],
       [0.91, 0.77, 0.33, 0.08, 0.41, ..., 0.96, 0.36, 0.36, 0.02, 0.19],
       [0.40, 0.93, 0.10, 0.95, 0.87, ..., 0.27, 0.46, 0.40, 0.25, 0.51],
       ...,
       [0.03, 0.99, 0.09, 0.45, 0.84, ..., 0.12, 0.29, 0.37, 0.91, 0.14],
       [0.62, 0.20, 0.29, 0.45, 0.55, ..., 0.48, 0.87, 0.22, 0.14, 0.93],
       [0.89, 0.27, 0.84, 0.76, 1.00, ..., 0.98, 0.41, 0.57, 0.74, 0.23],
       [0.27, 0.74, 0.81, 0.20, 0.31, ..., 0.51, 0.23, 0.95, 0.88, 0.14],
       [0.88, 0.20, 0.57, 0.93, 0.56, ..., 0.55, 0.40, 0.76, 0.02, 0.81]])
```

Reset to default settings:

```
>>> np_preferences(reset=True)
>>> random_array
array([[0.54881350, 0.71518937, 0.60276338, ..., 0.02010755, 0.82894003,
        0.00469548],
       [0.67781654, 0.27000797, 0.73519402, ..., 0.25435648, 0.05802916,
        0.43441663],
       [0.31179588, 0.69634349, 0.37775184, ..., 0.86219152, 0.97291949,
        0.96083466],
       ...,
       [0.89111234, 0.26867428, 0.84028499, ..., 0.57367960, 0.73729114,
        0.22519844],
       [0.26969792, 0.73882539, 0.80714479, ..., 0.94836806, 0.88130699,
        0.14193340],
       [0.88498232, 0.19701397, 0.56861333, ..., 0.75842952, 0.02378743,
        0.81357508]])
```

pd_preferences

pyhelpers.settings.**pd_preferences**(*reset=False*, *max_columns=100*, *min_rows=10*, *max_rows=40*, *precision=4*, *east_asian_text=False*, ***kwargs*)

Alters parameters of some frequently-used Pandas options and settings for displaying pandas dataframes.

This function allows adjusting various display options such as maximum number of columns, maximum number of rows, decimal precision, handling of East Asian text and suppression of future warnings.

Parameters

- **reset** (*bool | str*) – Whether to reset all settings to their defaults; defaults to False.
- **max_columns** (*int*) – Maximum number of columns to display; corresponds to 'display.max_columns' in pandas.set_option(); defaults to 100.
- **min_rows** (*int*) – Minimum number of rows to display; corresponds to 'display.min_rows' in pandas.set_option(); defaults to 10.

- **max_rows** (*int*) – Maximum number of rows to display; corresponds to `'display.max_rows'` in pandas.set_option(); defaults to 40.
- **precision** (*int*) – Number of decimal places to display; corresponds to `'display.precision'` in pandas.set_option(); defaults to 4.
- **east_asian_text** (*bool*) – Whether to adjust the display for East Asian text; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the function pandas.set_option().

Examples:

```
>>> import numpy as np
>>> import pandas as pd
>>> np.random.seed(0)
>>> random_array = np.random.rand(100, 100)
>>> data_frame = pd.DataFrame(random_array)
>>> data_frame
          0         1         2   ...        97        98        99
0   0.548814  0.715189  0.602763  ...  0.020108  0.828940  0.004695
1   0.677817  0.270008  0.735194  ...  0.254356  0.058029  0.434417
2   0.311796  0.696343  0.377752  ...  0.862192  0.972919  0.960835
3   0.906555  0.774047  0.333145  ...  0.356707  0.016329  0.185232
4   0.401260  0.929291  0.099615  ...  0.401714  0.248413  0.505866
..       ...       ...       ...  ...       ...       ...       ...
95  0.029929  0.985128  0.094747  ...  0.369907  0.910011  0.142890
96  0.616935  0.202908  0.288809  ...  0.215006  0.143577  0.933162
97  0.891112  0.268674  0.840285  ...  0.573680  0.737291  0.225198
98  0.269698  0.738825  0.807145  ...  0.948368  0.881307  0.141933
99  0.884982  0.197014  0.568613  ...  0.758430  0.023787  0.813575
[100 rows x 100 columns]
```

Limit to display of at most 6 columns and round the numbers to 2 decimal places:

```
>>> from pyhelpers.settings import pd_preferences
>>> pd_preferences(max_columns=6, precision=2)
>>> data_frame
     0    1    2   ...   97   98   99
0  0.55 0.72 0.60  ... 0.02 0.83 0.00
1  0.68 0.27 0.74  ... 0.25 0.06 0.43
2  0.31 0.70 0.38  ... 0.86 0.97 0.96
3  0.91 0.77 0.33  ... 0.36 0.02 0.19
4  0.40 0.93 0.10  ... 0.40 0.25 0.51
..  ...  ...  ...  ...  ...  ...  ...
95 0.03 0.99 0.09  ... 0.37 0.91 0.14
96 0.62 0.20 0.29  ... 0.22 0.14 0.93
97 0.89 0.27 0.84  ... 0.57 0.74 0.23
98 0.27 0.74 0.81  ... 0.95 0.88 0.14
99 0.88 0.20 0.57  ... 0.76 0.02 0.81
[100 rows x 100 columns]
```

Reset to default settings:

```
>>> pd_preferences(reset=True)
>>> data_frame
```

(continues on next page)

(continued from previous page)

```
          0         1         2   ...        97        98        99
0   0.548814  0.715189  0.602763   ...  0.020108  0.828940  0.004695
1   0.677817  0.270008  0.735194   ...  0.254356  0.058029  0.434417
2   0.311796  0.696343  0.377752   ...  0.862192  0.972919  0.960835
3   0.906555  0.774047  0.333145   ...  0.356707  0.016329  0.185232
4   0.401260  0.929291  0.099615   ...  0.401714  0.248413  0.505866
..       ...       ...       ...  ...       ...       ...       ...
95  0.029929  0.985128  0.094747   ...  0.369907  0.910011  0.142890
96  0.616935  0.202908  0.288809   ...  0.215006  0.143577  0.933162
97  0.891112  0.268674  0.840285   ...  0.573680  0.737291  0.225198
98  0.269698  0.738825  0.807145   ...  0.948368  0.881307  0.141933
99  0.884982  0.197014  0.568613   ...  0.758430  0.023787  0.813575
[100 rows x 100 columns]
```

Note

Default values of all available options can be checked by running pandas.describe_option() or `pandas._config.config._registered_options`.

4.2 dirs

Utilities for directory and file operations, navigation and management.

4.2.1 Path formatting

1anti-flashwhitewhite

normalize_path(path[, sep, as_str, prepend_dot])	Convert a path into a consistent format for cross-platform compatibility.
standardize_path(path[, sep, parents])	Standardize file and directory paths.
get_relative_path(path[, as_str, ...])	Check if the pathname is relative to the current working directory.
format_display_path(path[, normalized, ...])	Format a path string for display, logging or printing purposes.

normalize_path

pyhelpers.dirs.**normalize_path**(*path*, *sep='/'*, *as_str=True*, *prepend_dot=False*)

Convert a path into a consistent format for cross-platform compatibility.

This function converts all path separators (`"\"` and `"/"`) into a single, consistent separator and collapses consecutive separators, so that the resulting path is consistent across systems (e.g. Windows and Unix-like operating systems).

Note

A run of *leading* separators (e.g. a Windows UNC prefix such as `"\\server\share"`) is also collapsed to a single separator. This function does not preserve UNC-style double-slash prefixes; avoid it for network-share paths.

Parameters

- **path** (*str | bytes | pathlib.Path | os.PathLike*) – The filesystem path to normalize.
- **sep** (*str*) – Path separator used when the result is returned as a string; this is ignored when `as_str=False`, in which case the separator is determined automatically by `pathlib.Path` based on the current platform. Defaults to `"/"`.
- **as_str** (*bool*) – Whether to return the normalized path as a string; if `False`, a `pathlib.Path` object is returned instead. Defaults to `True`.
- **prepend_dot** (*bool*) – If `True`, prepends `"./"` to a relative path that does not already begin with `"./"`, `"../"`, or a path that `pathlib` considers absolute *on the current platform*. Note that on Windows, a path with a leading separator but no drive letter (e.g. `"/pyhelpers/data"`) is drive-relative, not absolute, and will have `"./"` prepended; on POSIX systems, that same path is absolute and is left untouched. Only takes effect when `as_str=True`, since `pathlib.Path` normalizes away a leading `"./"` when constructing a `Path` object. Defaults to `False`.

Returns

Pathname formatted to a consistent standard, either as a string (default) or as a `pathlib.Path` object when `as_str=False`.

Return type

str | pathlib.Path

Examples:

```
>>> from pyhelpers.dirs import normalize_path
>>> from pathlib import Path
>>> import os

>>> path = "tests\data\dat.csv"
>>> normalize_path(path)
'tests/data/dat.csv'

>>> normalize_path(path, prepend_dot=True)
'./tests/data/dat.csv'

>>> path = "tests//data/dat.csv"
>>> normalize_path(path)
'tests/data/dat.csv'

>>> path = Path("tests\data/dat.csv")  # Assuming Windows OS
>>> normalize_path(path, sep=os.sep)
'tests\data\dat.csv'
>>> normalize_path(path, sep=os.sep, as_str=False)
WindowsPath('tests/data/dat.csv')

>>> # on POSIX: absolute, untouched
>>> normalize_path("/usr/bin", prepend_dot=True)
'/usr/bin'
>>> # on Windows: drive-relative, not absolute
```

(continues on next page)

(continued from previous page)

```
>>> normalize_path("/usr/bin", prepend_dot=True)
'./usr/bin'
```

standardize_path

pyhelpers.dirs.**standardize_path**(*path*, *sep='-'*, *parents=False*)

Standardize file and directory paths.

This function handles camelCase, spaces and special characters while preserving file extensions (`"."`) and operating system path anchors (`"/"` or `"C:\"`).

Parameters

- **path** (*str | pathlib.Path*) – The target file or directory path `str` or `pathlib.Path` object to clean.
- **sep** (*str*) – The character used to separate words. Defaults to `"-"`.
- **parents** (*bool*) – If `True`, standardizes all parent directories in the path layout. If `False` (default), only standardizes the final file or folder name.

Returns

A standardized operating system path object.

Return type

pathlib.Path

Examples:

```
>>> from pyhelpers.dirs import standardize_path
>>> from pathlib import Path

>>> standardize_path('Random Evaluation Name')  # On Windows
WindowsPath('random-evaluation-name')

>>> standardize_path("-license.txt")
WindowsPath('-license.txt')

>>> standardize_path('C:/Users/Username/ProjectData/schema-v2.json')
WindowsPath('C:/Users/Username/ProjectData/schema-v2.json')

>>> standardize_path(Path('/Archive/Old Folders/MyScript.py'), sep="_")
WindowsPath('/Archive/Old Folders/my_script.py')

>>> # Only standardize the filename, leave the directories alone
>>> standardize_path('/Archive/Old Folders/MyScript.py', parents=True)
WindowsPath('/archive/old-folders/my-script.py')
```

get_relative_path

pyhelpers.dirs.**get_relative_path**(*path*, *as_str=False*, *normalized=True*, *quoted=False*, *is_dir=None*, *prepend_dot=False*)

Check if the pathname is relative to the current working directory.

If the specified `path` resides within the current working directory, this function returns its relative path counterpart. Otherwise, it returns the original absolute path.

Parameters

- **path** (*str | bytes | pathlib.Path | os.PathLike*) – Pathname of a file or directory.
- **as_str** (*bool*) – Whether to return the path as a string; if `False`, a `pathlib.Path` object is returned instead and `normalized`, `quoted`, `is_dir` and `prepend_dot` are all ignored. Defaults to `False`.
- **normalized** (*bool*) – Whether to run the path through `_normalize_path()` (or, when `quoted=True`, `_format_display_path()`) before returning it as a string. If `False`, the string is returned using the native OS separator, unnormalized. Only takes effect when `as_str=True`. Defaults to `True`.
- **quoted** (*bool*) – Whether to format and wrap the returned string via `_format_display_path()` (double-quoted, with a trailing separator for directories) instead of `_normalize_path()`. Only takes effect when `as_str=True`. Defaults to `False`.
- **is_dir** (*bool | None*) – Explicitly treat the path as a directory; passed through to `_format_display_path()`. Only takes effect when `as_str=True` and `quoted=True`. Defaults to `None`.
- **prepend_dot** (*bool*) – Whether to prepend a dot-relative prefix to a relative path. Only takes effect when `as_str=True`. Defaults to `False`.

Returns

A location relative to the current working directory if `path` is within the working space; otherwise, a resolved copy of the absolute `path`.

Return type

pathlib.Path | str

Examples:

```
>>> from pyhelpers.dirs import get_relative_path, cd

>>> get_relative_path(path=".")  # on Windows
WindowsPath('.')

>>> get_relative_path(path=cd(), as_str=True)
'.'

>>> get_relative_path(path="C:/Windows", as_str=True)
'C:/Windows'

>>> get_relative_path(path="C:\Program Files", as_str=True, normalized=False)
'C:\Program Files'
```

format_display_path

pyhelpers.dirs.**format_display_path**(*path*, *normalized=True*, *surrounded_by='"'*, *is_dir=None*, *prepend_dot=False*)

Format a path string for display, logging or printing purposes.

This function generates a visual representation of a path. It can optionally add trailing slashes for directories, wrap the output in quotes and prepend a dot-slash for relative paths (e.g. when preparing shell commands).

Parameters

- **path** (*str | bytes | pathlib.Path | os.PathLike*) – The filesystem path to format for display.
- **normalized** (*bool*) – Whether to standardize slashes via `_normalize_path()`. Defaults to `True`.
- **surrounded_by** (*str*) – A string literal used to wrap the output. Defaults to `'"'`.
- **is_dir** (*bool | None*) – Explicitly treat the path as a directory. If `None`, the filesystem is checked first; when the path does not exist, this falls back to a heuristic that treats a path without a file extension as a directory (which can misclassify extensionless files such as `"Makefile"` or `"LICENSE"`). Defaults to `None`.
- **prepend_dot** (*bool*) – If `True`, prepends a `./` (or native OS equivalent) to paths that are not absolute. Defaults to `False`.

Returns

Formatted pathname with configured slashes and wrappers.

Return type

str

Examples:

```
>>> from pyhelpers.dirs import format_display_path

>>> format_display_path("pyhelpers\data")
'"pyhelpers/data/"'

>>> format_display_path("pyhelpers\data", normalized=False)  # on Windows
'"pyhelpers\data\"'

>>> format_display_path("pyhelpers\data\pyhelpers.dat", prepend_dot=True)
'"./pyhelpers/data/pyhelpers.dat"'

>>> format_display_path("C:\Windows", prepend_dot=True)  # on Windows
'"C:/Windows/"'
```

4.2.2 Path validation

1anti-flashwhitewhite

is_dir_path(dir_path)	Check whether a string is formatted as a directory path.
validate_filename(file_path[, suffix_num])	Validate a filename, generating a uniquely suffixed name if the file already exists.
check_files_exist(filenames, dir_path[, verbose])	Check whether specified files exist within a given directory.

is_dir_path

pyhelpers.dirs.**is_dir_path**(*dir_path*)

Check whether a string is formatted as a directory path.

This function performs a syntax-only check: it does not verify that the directory actually exists, only that the string is shaped like a directory path and does not contain characters or a length that the operating system would reject outright. See also this discussion on Stack Overflow and the Windows System Error Codes reference.

Parameters

dir_path (*str | bytes | pathlib.Path | os.PathLike*) – Pathname of a directory.

Returns

`True` if `dir_path` is formatted as a valid directory path, `False` otherwise.

Return type

bool

Examples:

```
>>> from pyhelpers.dirs import cd, is_dir_path

>>> is_dir_path("tests")
False

>>> is_dir_path("/tests")
True

>>> is_dir_path(cd("tests"))
True

>>> is_dir_path(".\tests/")
True
```

validate_filename

pyhelpers.dirs.**validate_filename**(*file_path*, *suffix_num=1*)

Validate a filename, generating a uniquely suffixed name if the file already exists.

If the file specified by `file_path` already exists, this function appends a numeric suffix such as `"(1)"`, `"(2)"`, etc. to the filename (before its extension, if any) until it finds a pathname that doesn't already exist. A pre-existing numeric suffix on `file_path` itself (e.g. from a prior call) is stripped before a new one is generated, so repeated calls don't accumulate suffixes.

Parameters

- **file_path** (*str | os.PathLike*) – Pathname of a file.
- **suffix_num** (*int*) – Starting number to use as a suffix if the filename already exists. Defaults to `1`.

Returns

Validated pathname of a file (with a unique numeric suffix if necessary).

Return type
str

Examples:

```
>>> from pyhelpers.dirs import validate_filename
>>> import os

>>> test_file_pathname = "tests/data/test.txt"
>>> os.path.exists(test_file_pathname)
False

>>> # If the file does not exist, the function returns the same filename
>>> file_pathname_0 = validate_filename(test_file_pathname)
>>> os.path.relpath(file_pathname_0)  # on Windows
'tests\data\test.txt'

>>> # Create a file named "test.txt"
>>> open(test_file_pathname, 'w').close()
>>> os.path.isfile(test_file_pathname)
True

>>> # As "test.txt" exists, the function returns a new pathname ending with "test(1).txt"
>>> file_pathname_1 = validate_filename(test_file_pathname)
>>> os.path.relpath(file_pathname_1)  # on Windows
'tests\data\test(1).txt'

>>> # When "test(1).txt" exists, it returns a pathname of a file named "test(2).txt"
>>> open(file_pathname_1, 'w').close()
>>> os.path.exists(file_pathname_1)
True

>>> file_pathname_2 = validate_filename(test_file_pathname)
>>> os.path.relpath(file_pathname_2)  # on Windows
'tests\data\test(2).txt'

>>> # Remove the created files
>>> for x in [file_pathname_0, file_pathname_1]:
...     os.remove(x)
```

check_files_exist

pyhelpers.dirs.**check_files_exist**(*filenames*, *dir_path*, *verbose=False*, ***kwargs*)

Check whether specified files exist within a given directory.

This function compares files by basename only, not by their full relative path – if `filenames` includes a path with subdirectory components, only the final filename portion is compared, so a same-named file located in a *different* subdirectory of `dir_path` (e.g. when `incl_subdir=True` is passed via `kwargs`) would count as a match.

Parameters

- **filenames** (*Iterable*) – Filenames to check for existence.
- **dir_path** (*str | os.PathLike*) – Pathname of the directory in which to check for the files.
- **verbose** (*bool | int*) – Whether to print relevant information to the console. Defaults to `False`.

- **kwargs** – [Optional] Additional parameters for `get_file_paths()` (e.g. `incl_subdir=True` to also check files within subdirectories).

Returns

`True` if all queried files exist in the directory, `False` otherwise.

Return type

bool

Examples:

```
>>> from pyhelpers.dirs import check_files_exist

>>> test_dir_name = "tests/data"

>>> # Check if all required files exist in the directory
>>> check_files_exist(["dat.csv", "dat.txt"], dir_path=test_dir_name)
True

>>> # If not all required files exist, print the missing files
>>> check_files_exist(("dat.csv", "dat.txt", "dat_0.txt"), test_dir_name, verbose=True)
Error: Required files are not satisfied, missing files are: ['dat_0.txt']
False
```

4.2.3 Directory navigation

1anti-flashwhitewhite

cd(*subdir[, mkdir, cwd, back_check, ...])	Specify and resolve the pathname of a directory or file.
cdd(*subdir[, mkdir, data_dir])	Specify and resolve the pathname of a directory (or file) under the designated `data_dir`.
cd_data(*subdir[, data_dir, mkdir, as_str])	Specify and resolve the pathname of a directory (or file) under `data_dir` of a package.
resolve_dir_path([dir_path, subdir, msg])	Resolve a directory path into an absolute pathname.
find_executable(name[, options, target, ...])	Find the pathname of an executable file for a specified application.

cd

pyhelpers.dirs.**cd**(**subdir*, *mkdir=False*, *cwd=None*, *back_check=False*, *as_str=False*, *normalized=True*, ***kwargs*)

Specify and resolve the pathname of a directory or file.

This function constructs an operating system-agnostic path object. Handles missing directories and resolves parent path traversal dynamically.

Parameters

- **subdir** (*str | bytes | pathlib.Path | os.PathLike | None*) – Name of a directory or directories (and/or a filename). `None` values are safely ignored.

- **mkdir** (*bool*) – Whether to create the directory if it does not exist. Defaults to False.
- **cwd** (*str | bytes | pathlib.Path | os.PathLike | None*) – Current working directory fallback. Defaults to None.
- **back_check** (*bool*) – Whether to check if a parent directory exists. Defaults to False.
- **as_str** (*bool*) – If True, forces the return type to be a standard string instead of a path object. Defaults to False.
- **normalized** (*bool*) – Whether to normalize the returned pathname structurally. Defaults to True.
- **kwargs** – Optional parameters (e.g. mode=0o777) passed to pathlib.Path.mkdir.

Returns

A standardized path object representing the target directory or file.

Return type

Path | str

Examples:

```
>>> from pyhelpers.dirs import cd
>>> from pathlib import Path

>>> cur_dir = cd()  # Current working directory
>>> cur_dir.name
'pyhelpers'

>>> cd(None).relative_to(cur_dir)  # On Windows OS
WindowsPath('.')

>>> # The directory will be created if it does not exist
>>> test_path = cd("tests")
>>> test_path.relative_to(cur_dir)
WindowsPath('tests')

>>> test_path = cd("tests\folder1/folder2")
>>> test_path.relative_to(cur_dir)
WindowsPath('tests/folder1/folder2')

>>> test_path = cd(Path("tests\folder1"), as_str=True, normalized=False)
>>> test_path  # On Windows OS
'X:\pyhelpers\tests\folder1'
```

cdd

pyhelpers.dirs.**cdd**(**subdir*, *mkdir=False*, *data_dir='data'*, ***kwargs*)

Specify and resolve the pathname of a directory (or file) under the designated data_dir.

Parameters

- **subdir** (*str | bytes | pathlib.Path | os.PathLike | None*) – Name of a directory or directories (and/or a filename).

- **mkdir** (*bool*) – Whether to create the directory if it does not exist. Defaults to `False`.
- **data_dir** (*str | bytes | pathlib.Path | os.PathLike*) – Name of the directory where data is stored. Defaults to `"data"`.
- **kwargs** – Optional parameters passed to *cd()*.

Returns

Pathname of a directory or file under `data_dir`.

Return type

pathlib.Path | str

Examples:

```
>>> from pyhelpers.dirs import cd, cdd, delete_dir
>>> from pathlib import Path

>>> cur_dir = cd()
>>> cur_dir == Path.cwd()
True

>>> test_path = cdd()
>>> # As `mkdir=False`, `test_path` will NOT be created if it doesn't exist
>>> test_path.relative_to(cur_dir)
WindowsPath('data')

>>> test_path = cdd(data_dir="test_cdd", mkdir=True)
>>> # As `mkdir=True`, `test_path` will be created if it doesn't exist
>>> test_path.is_dir()
True
>>> test_path.relative_to(cur_dir)
WindowsPath('test_cdd')

>>> # Delete the "test_cdd" folder
>>> delete_dir(test_path, verbose=True)
To delete the directory "./test_cdd/"
? [No]|Yes: yes
Deleting "./test_cdd/" ... Done.

>>> # Set `data_dir` to be `"tests"`
>>> test_path = cdd("data", data_dir="test_cdd", mkdir=True)
>>> test_path.relative_to(cur_dir)
WindowsPath('test_cdd/data')

>>> # Delete the "test_cdd" folder and the sub-folder "data"
>>> test_cdd = test_path.parent
>>> delete_dir(test_cdd, verbose=True)
To delete the directory "./test_cdd/" (Not empty)
? [No]|Yes: yes
Deleting "./test_cdd/" ... Done.
>>> # # Alternatively,
>>> # import shutil
>>> # shutil.rmtree(test_cdd)
```

cd_data

pyhelpers.dirs.**cd_data**(**subdir*, *data_dir='data'*, *mkdir=False*, *as_str=False*, ***kwargs*)

Specify and resolve the pathname of a directory (or file) under `data_dir` of a package.

Parameters

- **subdir** (*str | os.PathLike | bytes*) – Name of a directory or directories (and/or a filename).
- **data_dir** (*str | os.PathLike | bytes*) – Name of the directory to store data; defaults to `"data"`.
- **mkdir** (*bool*) – Whether to create the directory if it does not exist; defaults to `False`.
- **as_str** (*bool*) – If `True`, forces the return type to be a standard string instead of a path object. Defaults to `False`.
- **kwargs** – [Optional] Additional parameters (e.g. `mode=0o777`) for the method pathlib.Path.mkdir.

Returns

A normalized `pathlib.Path` object (or `str`) of the requested target.

Return type

pathlib.Path | str

Examples:

```
>>> from pyhelpers.dirs import cd_data
>>> from pathlib import Path
>>> test_path = cd_data("tests", mkdir=False)
>>> test_path.relative_to(Path.cwd())  # on Windows
WindowsPath('pyhelpers/data/tests')
```

resolve_dir_path

pyhelpers.dirs.**resolve_dir_path**(*dir_path=None*, *subdir=''*, *msg='Invalid input!'*, ***kwargs*)

Resolve a directory path into an absolute pathname.

This function accepts a variety of input types and converts them into a standardized directory pathname via *cd()*. If `dir_path` is not given, `subdir` is used instead (or the current directory, if `subdir` is also not given).

Note

Both a relative and an absolute `dir_path` are routed through the same call to *cd()*, on the assumption that `cd` (like `os.path.join`) treats an absolute argument as replacing any preceding base directory, so `**kwargs` (e.g. `mkdir=True`) is applied consistently either way. If `cd` does not behave this way for absolute paths, this needs revisiting.

Parameters

- **dir_path** (*str | bytes | os.PathLike | None*) – Pathname of a data directory; if `dir_path=None` (default), `subdir` is examined instead to

construct a valid directory path.

- **subdir** (*str | bytes | os.PathLike*) – Name of a subdirectory to fall back on when `dir_path=None`. Defaults to `""`.
- **msg** (*str*) – Error message used when `dir_path` cannot be decoded into a valid pathname. Defaults to `"Invalid input!"`.
- **kwargs** – [Optional] Additional parameters for *cd()* (e.g. `mkdir=True` to create the directory if it does not already exist).

Returns
Valid pathname of a directory.

Return type
pathlib.Path

Examples:

```
>>> from pyhelpers.dirs import resolve_dir_path, get_relative_path
>>> from pathlib import Path

>>> data_dir = resolve_dir_path()
>>> get_relative_path(data_dir)
'.'

>>> data_dir = resolve_dir_path("tests")
>>> get_relative_path(data_dir)
'tests'

>>> data_dir = resolve_dir_path(subdir=Path("data"))
>>> get_relative_path(data_dir)
'data'
```

find_executable

pyhelpers.dirs.**find_executable**(*name*, *options=None*, *target=None*, *normalized=True*, *as_str=True*)

Find the pathname of an executable file for a specified application.

This function relies on `_find_file_path()` to check a known `target` pathname, search through `options`, and fall back to the system's PATH, in that order.

Parameters

- **name** (*str*) – Name or filename of the application that is to be called.
- **options** (*list | set | None*) – Possible pathnames or directories to search for the executable; defaults to `None`.
- **target** (*str | None*) – Specific pathname of the executable file (if already known); this is checked first and, if it does not resolve to a valid match, the function stops and does not search `options` or the system PATH. Defaults to `None`.
- **normalized** (*bool*) – Whether to format the returned pathname for display via `_format_display_path()`. If `True`, the pathname is returned as a formatted `str`; if `False`, it is returned unformatted as a `pathlib.Path`. Defaults to `True`.

- **as_str** (*bool*) – Whether to return the path as a string; if `False`, a `pathlib.Path` object is returned instead. Defaults to `True`.

Returns

Whether the specified executable file exists (i.e. a boolean indicating existence), together with its pathname, or `None` if it was not found.

Return type

tuple[bool, pathlib.Path | str | None]

Examples:

```
>>> from pyhelpers.dirs import find_executable
>>> import os
>>> import sys

>>> python_exe = "python.exe"
>>> python_exe_exists, path_to_python_exe = find_executable(python_exe)
>>> python_exe_exists
True

>>> possible_paths = [os.path.dirname(sys.executable), sys.executable]
>>> target = possible_paths[0]  # a directory, not a file - not a valid target
>>> python_exe_exists, path_to_python_exe = find_executable(python_exe, target=target)
>>> python_exe_exists
False

>>> target = possible_paths[1]
>>> python_exe_exists, path_to_python_exe = find_executable(python_exe, target=target)
>>> python_exe_exists
True

>>> python_exe_exists, path_to_python_exe = find_executable(possible_paths[1])
>>> python_exe_exists
True

>>> text_exe = "pyhelpers.exe"  # This file does not actually exist
>>> test_exe_exists, path_to_test_exe = find_executable(text_exe, possible_paths)
>>> test_exe_exists
False
>>> path_to_test_exe is None
True
```

4.2.4 Directory management

1anti-flashwhitewhite

delete_dir(dir_path[, ...])	Delete a directory or multiple directories.
get_file_paths(dir_path[, file_ext, ...])	Get the paths of files in a directory, optionally filtered by file extension.

delete_dir

pyhelpers.dirs.**delete_dir**(*dir_path*, *confirmation_required=True*, *verbose=False*, *indent=2*, *raise_error=False*, ***kwargs*)

Delete a directory or multiple directories.

Parameters

- **dir_path** (*str | bytes | os.PathLike | collections.abc.Sequence*) – Pathname(s) of the directory (or directories) to be deleted.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding. Defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console. Defaults to `False`.
- **indent** (*int | str*) – Indentation level; if an integer, represents the number of spaces; if a string, used as the indentation character (e.g. `'\t'`). Defaults to 2 (two spaces).
- **raise_error** (*bool*) – Whether to raise an exception if deletion fails; if `raise_error=False` (default), the error will be suppressed. Defaults to `False`.
- **kwargs** – Optional parameters passed directly to shutil.rmtree (the underlying deletion function).

Examples:

```
>>> from pyhelpers.dirs import cd, delete_dir, get_relative_path

>>> test_dirs = []
>>> for x in range(3):
...     test_dirs.append(cd("tests", f"test_dir{x}", mkdir=True))
...     if x == 0:
...         cd("tests", f"test_dir{x}", "a_folder", mkdir=True)
...     elif x == 1:
...         open(cd("tests", f"test_dir{x}", "file"), 'w').close()

>>> for x in test_dirs:
...     print(get_relative_path(x))
tests   est_dir0
tests   est_dir1
tests   est_dir2

>>> delete_dir(test_dirs, verbose=True)
Confirm deletion of the following directories:
  "tests/test_dir0/" (Not empty)
  "tests/test_dir1/" (Not empty)
  "tests/test_dir2/"
? [No]|Yes: yes
Deleting:
  "tests/test_dir0/" ... Done.
  "tests/test_dir1/" ... Done.
  "tests/test_dir2/" ... Done.
```

get_file_paths

pyhelpers.dirs.**get_file_paths**(*dir_path*, *file_ext=None*, *incl_subdir=False*, *abs_path=False*, *normalized=True*, *prepend_dot=False*)

Get the paths of files in a directory, optionally filtered by file extension.

This function retrieves paths of files within the directory specified by `dir_path`. Files are optionally filtered by the exact `file_ext` given, matched against each file's actual extension (via `os.path.splitext`) rather than a trailing-substring comparison. If `file_ext` is `None`, `"*"` or `"all"`, all files are returned. If `incl_subdir=True`, subdirectories are traversed recursively.

Parameters

- **dir_path** (*str | os.PathLike*) – Pathname of the directory to search.
- **file_ext** (*str | None*) – Exact file extension to filter files by, with or without the leading dot (e.g. `".txt"` or `"txt"`). Defaults to `None`, in which case all files are returned regardless of extension.
- **incl_subdir** (*bool*) – Whether to include files from subdirectories; when `incl_subdir=True`, files from all subdirectories are included recursively. Defaults to `False`.
- **abs_path** (*bool*) – Whether to return absolute pathname(s). Defaults to `False`.
- **normalized** (*bool*) – Whether to normalize the returned pathname(s) via `_normalize_path()`. Defaults to `True`.
- **prepend_dot** (*bool*) – If `True`, prepends `"./"` to a relative pathname that doesn't already have a dot-relative or absolute prefix. Only takes effect when `normalized=True`. Defaults to `False`.

Returns

List of file pathnames matching the criteria.

Return type

list[str]

Examples:

```
>>> from pyhelpers.dirs import get_file_paths, delete_dir
>>> from pyhelpers.store import unzip
>>> import os

>>> test_dir_name = "tests/data"

>>> # Get all files in the directory (without subdirectories) on Windows
>>> get_file_paths(test_dir_name, prepend_dot=True)
['./tests/data/csr_mat.npz',
 './tests/data/dat.csv',
 './tests/data/dat.feather',
 './tests/data/dat.joblib',
 './tests/data/dat.json',
 './tests/data/dat.ods',
 './tests/data/dat.pickle',
 './tests/data/dat.pickle.bz2',
```

(continues on next page)

(continued from previous page)

```
 './tests/data/dat.pickle.gz',
 './tests/data/dat.pickle.xz',
 './tests/data/dat.txt',
 './tests/data/dat.xlsx',
 './tests/data/zipped.7z',
 './tests/data/zipped.txt',
 './tests/data/zipped.zip']

>>> get_file_paths(test_dir_name, file_ext=".txt")
['tests/data/dat.txt', 'tests/data/zipped.txt']

>>> output_dir = unzip('tests/data/zipped.zip', ret_output_dir=True)
>>> os.listdir(output_dir)
['zipped.txt']

>>> # Get absolute paths of all files contained in the folder (incl. all subdirectories)
>>> get_file_paths(test_dir_name, file_ext="txt", incl_subdir=True, abs_path=True)
['<Parent directories>/tests/data/dat.txt',
 '<Parent directories>/tests/data/zipped.txt',
 '<Parent directories>/tests/data/zipped/zipped.txt']

>>> delete_dir(output_dir, confirmation_required=False, verbose=True)
Deleting "tests/data/zipped/" ... Done.
```

4.3 ops

Miscellaneous operations.

4.3.1 Basic computation / conversion

1anti-flashwhitewhite

get_utc_tai_offset([verbose, raise_error, url])	Retrieves the difference between UTC (Coordinated Universal Time) and TAI (International Atomic Time).
gps_time_to_utc(gps_time[, as_datetime, ...])	Converts GPS time to UTC time.
find_closest_date(date, lookup_dates[, ...])	Finds the closest date to a given date from a list of dates.
parse_size(size[, binary, precision])	Parses size into human-readable format or vice versa.
get_number_of_chunks(file_or_obj[, ...])	Gets the total number of chunks of a data file, given a minimum chunk size limit.
interquartile_range(num_dat[, axis, rng, ...])	Calculates the interquartile range (IQR) of numerical data.
get_extreme_outlier_bounds(num_dat[, k])	Gets the upper and lower bounds for extreme outliers using the interquartile range method.

get_utc_tai_offset

pyhelpers.ops.**get_utc_tai_offset**(*verbose=False*, *raise_error=False*, *url=None*)

Retrieves the difference between UTC (Coordinated Universal Time) and TAI (International Atomic Time).

This difference increases as leap seconds are added to UTC.

Parameters

- **verbose** (*bool | int*) – Whether to print additional information to the console; defaults to `True`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **url** (*str | None*) – URL to the IERS (International Earth Rotation and Reference Systems Service)'s Bulletin C, which announces leap seconds; defaults to `None`.

Returns

The difference between UTC and TAI, i.e. UTC - TAI.

Return type

int | float | None

Examples:

```
>>> from pyhelpers.ops import get_utc_tai_offset
>>> get_utc_tai_offset()
-37
```

gps_time_to_utc

pyhelpers.ops.**gps_time_to_utc**(*gps_time*, *as_datetime=True*, *utc_tai_offset=None*)

Converts GPS time to UTC time.

Parameters

- **gps_time** (*float*) – Standard GPS time in seconds since GPS epoch (6 January 1980).
- **as_datetime** (*bool*) – If `True`, the function returns the UTC time as datetime type; otherwise string type; defaults to `True`.
- **utc_tai_offset** (*float | int | None*) – The difference between UTC (Coordinated Universal Time) and the TAI (International Atomic Time), i.e. UTC - TAI; defaults to `None`.

Returns

UTC datetime corresponding to the GPS time.

Return type

datetime.datetime | str

Examples:

```
>>> from pyhelpers.ops import gps_time_to_utc
>>> gps_time = 1271398985.7822514
>>> utc_time = gps_time_to_utc(gps_time)
>>> utc_time
datetime.datetime(2020, 4, 20, 6, 22, 47, 782251)
>>> utc_time = gps_time_to_utc(gps_time, as_datetime=False)
>>> utc_time
>>> '2020-04-20T06:22:47.782251'
```

find_closest_date

pyhelpers.ops.**find_closest_date**(*date*, *lookup_dates*, *as_datetime=False*, *fmt='%Y-%m-%d %H:%M:%S.%f'*)

Finds the closest date to a given date from a list of dates.

Parameters

- **date** (*str | datetime.datetime*) – Date to find the closest match for.
- **lookup_dates** (*Iterable*) – Iterable of dates to search within.
- **as_datetime** (*bool*) – Whether to return the closest date as a datetime.datetime object; defaults to `False` which returns a string representation.
- **fmt** (*str*) – Format string for datetime parsing and formatting; defaults to `'%Y-%m-%d %H:%M:%S.%f'`.

Returns

Closest date to the given *date*.

Return type

str | datetime.datetime

Examples:

```
>>> from pyhelpers.ops import find_closest_date
>>> import pandas
>>> lookup_dates = pandas.date_range('2019-01-02', '2019-12-31')
>>> lookup_dates
DatetimeIndex(['2019-01-02', '2019-01-03', '2019-01-04', '2019-01-05',
               '2019-01-06', '2019-01-07', '2019-01-08', '2019-01-09',
               '2019-01-10', '2019-01-11',
               ...
               '2019-12-22', '2019-12-23', '2019-12-24', '2019-12-25',
               '2019-12-26', '2019-12-27', '2019-12-28', '2019-12-29',
               '2019-12-30', '2019-12-31'],
              dtype='datetime64[ns]', length=364, freq='D')
>>> date = '2019-01-01'
>>> closest_example_date = find_closest_date(date, lookup_dates)
>>> closest_example_date
'2019-01-02 00:00:00.000000'
>>> date = pandas.to_datetime('2019-01-01')
>>> closest_date = find_closest_date(date, lookup_dates, as_datetime=True)
>>> closest_date
Timestamp('2019-01-02 00:00:00', freq='D')
```

parse_size

pyhelpers.ops.**parse_size**(*size*, *binary=True*, *precision=1*)

Parses size into human-readable format or vice versa.

Parameters

- **size** (*str | int | float*) – Size to be parsed, either in human-readable format (e.g. `'10 MB'`) or as an integer.
- **binary** (*bool*) – Whether to use binary (factorized by 1024) or decimal (factorized by 10 ** 3) representation; defaults to `True` for binary representation.
- **precision** (*int*) – Number of decimal places when converting `size` to human-readable format; defaults to `1`.

Returns

Parsed size, either as an integer (for machine-readable) or a formatted string (for human-readable).

Return type

int | str

Examples:

```
>>> from pyhelpers.ops import parse_size
>>> parse_size(size='123.45 MB')
129446707
>>> parse_size(size='123.45 MB', binary=False)
123450000
>>> parse_size(size='123.45 MiB', binary=True)
129446707
>>> # If a metric unit (e.g. 'MiB') is specified in the input,
>>> # the function returns a result accordingly, regardless of `binary`
>>> parse_size(size='123.45 MiB', binary=False)
129446707
>>> parse_size(size=129446707, precision=2)
'123.45 MiB'
>>> parse_size(size=129446707, binary=False, precision=2)
'129.45 MB'
```

get_number_of_chunks

pyhelpers.ops.**get_number_of_chunks**(*file_or_obj*, *chunk_size_limit=50*, *binary=True*)

Gets the total number of chunks of a data file, given a minimum chunk size limit.

Parameters

- **file_or_obj** (*Any*) – Path to a file or an object representing the data.
- **chunk_size_limit** (*int | float | None*) – Minimum limit of chunk size in megabytes (MB) or mebibytes (MiB) above which the function counts the number of chunks; defaults to `50`.
- **binary** (*bool*) – Whether to use binary (factorized by 1024) or decimal (factorized by 10 ** 3) representation for size calculations; defaults to `True` for binary representation.

Returns
Number of chunks, or `None` if `file_or_obj` is invalid or chunk calculation is not applicable.

Return type
int | None

Examples:

```
>>> from pyhelpers.ops import get_number_of_chunks
>>> import numpy
>>> example_obj = numpy.zeros((1000, 1000))
>>> get_number_of_chunks(example_obj, chunk_size_limit=5)
2
>>> file_path = "C:\Program Files\Python310\python310.pdb"
>>> get_number_of_chunks(file_path, chunk_size_limit=2)
8
```

interquartile_range

pyhelpers.ops.**interquartile_range**(*num_dat*, *axis=None*, *rng=(25, 75)*, *scale=1.0*, *outlier_fences=False*, *k=1.5*, *ignore_nan=True*, ***kwargs*)

Calculates the interquartile range (IQR) of numerical data.

This function may serve as an alternative to scipy.stats.iqr when `scipy` is not available.

Parameters

- **num_dat** (*numpy.ndarray | list | tuple*) – Numerical data.
- **axis** (*int | None*) – Axis along which to calculate IQR; defaults to `None`.
- **rng** (*tuple*) – Percentile range for calculating IQR; defaults to `(25, 75)`.
- **scale** (*int | float*) – Scaling factor for the IQR; defaults to `1.0`.
- **outlier_fences** (*bool*) – Whether to calculate and return outlier fences; defaults to `False`.
- **k** (*int | float*) – Multiplier for IQR for calculating outlier fences; defaults to `1.5`
- **ignore_nan** (*bool*) – Whether to ignore NaN values; defaults to `True`.
- **kwargs** – [Optional] Additional parameters passed to numpy.percentile.

Returns
Scaled interquartile range.

Return type
float

Raises
ValueError – If input has insufficient non-NaN values.

Examples:

```
>>> from pyhelpers.ops import interquartile_range
>>> num_dat = [1, 2, 'nan', 3, 4]
>>> iqr = interquartile_range(num_dat, ignore_nan=True)
>>> iqr
1.5
>>> num_dat = list(range(100))
>>> iqr = interquartile_range(num_dat)
>>> iqr
49.5
>>> iqr, lower_fence, upper_fence = interquartile_range(num_dat, outlier_fences=True)
>>> iqr, lower_fence, upper_fence
(49.5, -49.5, 148.5)
```

get_extreme_outlier_bounds

pyhelpers.ops.**get_extreme_outlier_bounds**(*num_dat*, *k=1.5*)

Gets the upper and lower bounds for extreme outliers using the interquartile range method.

Parameters

- **num_dat** (*array-like*) – Array-like object containing numerical data.
- **k** (*float | int*) – Scale coefficient associated with the interquartile range; defaults to 1.5.

Returns

Tuple containing the lower and upper bounds for extreme outliers.

Return type

tuple

Examples:

```
>>> from pyhelpers.ops import get_extreme_outlier_bounds
>>> import pandas as pd
>>> data = pd.DataFrame(range(100), columns=['col'])
>>> data
    col
0     0
1     1
2     2
3     3
4     4
..  ...
95   95
96   96
97   97
98   98
99   99
[100 rows x 1 columns]
>>> data.describe()
              col
count  100.000000
mean    49.500000
std     29.011492
min      0.000000
25%     24.750000
50%     49.500000
```

(continues on next page)

(continued from previous page)

```
75%     74.250000
max     99.000000
>>> lo_bound, up_bound = get_extreme_outlier_bounds(data, k=1.5)
>>> lo_bound, up_bound
(0.0, 148.5)
```

4.3.2 Basic data manipulation

Iterable

1anti-flashwhitewhite

shift_decimal_to_int(value[, precision])	Converts a float value to an integer by simply removing its decimal point.
loop_in_pairs(iterable)	Generates pairs of consecutive elements from the given iterable.
split_list_by_size(lst, sub_len)	Splits a list into evenly sized sub-lists.
split_list(lst, num_of_sub)	Splits a list into a specified number of equally-sized sub-lists.
split_iterable(iterable, chunk_size)	Splits an iterable into evenly sized chunks.
update_dict(dictionary, updates[, inplace])	Updates a (nested) dictionary with another dictionary.
update_dict_keys(dictionary[, replacements])	Updates keys in a (nested) dictionary based on a given replacements dictionary.
get_dict_values(key, dictionary)	Retrieves all values in a (nested) dictionary for a given key.
remove_dict_keys(dictionary, *keys)	Removes multiple keys from a dictionary.
compare_dicts(dict1, dict2)	Compares the differences between two dictionaries.
merge_dicts(*dicts)	Merges multiple dictionaries into a single dictionary.

shift_decimal_to_int

pyhelpers.ops.**shift_decimal_to_int**(*value*, *precision=20*)

Converts a float value to an integer by simply removing its decimal point.

The function shifts the decimal point to the right until it disappears, effectively multiplying by the appropriate power of 10 to eliminate the fractional part. For example, 12.345 becomes 12345 (×1000).

Parameters

- **value** (*float | int*) – The float value to convert. Can be positive, negative, or zero.
- **precision** (*int*) – The maximum number of digits after the decimal point. Defaults to 20.

Returns

Integer with the decimal point removed.

Return type
int

Raises
TypeError – If `value` is not a number.

Examples:

```
>>> from pyhelpers.ops import shift_decimal_to_int
>>> shift_decimal_to_int(12.345)
12345
>>> shift_decimal_to_int(0.056)
56
>>> shift_decimal_to_int(-3.14)
-314
>>> shift_decimal_to_int(1000.0)
10000
>>> shift_decimal_to_int(1.2300)
123
>>> shift_decimal_to_int(0.0005)
5
>>> shift_decimal_to_int(1.2e-15)
12
```

Note

- Trailing zeros are preserved (1.2300 → 12300)
- The conversion uses string manipulation for accuracy and speed

loop_in_pairs

pyhelpers.ops.**loop_in_pairs**(*iterable*)

Generates pairs of consecutive elements from the given iterable.

Parameters
iterable (*Iterable*) – Iterable object from which to generate pairs.

Returns
Zip object containing pairs of consecutive elements.

Return type
zip

Examples:

```
>>> from pyhelpers.ops import loop_in_pairs
>>> res = loop_in_pairs(iterable=[1])
>>> list(res)
[]
>>> res = loop_in_pairs(iterable=range(0, 10))
>>> list(res)
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]
```

split_list_by_size

pyhelpers.ops.**split_list_by_size**(*lst*, *sub_len*)

Splits a list into evenly sized sub-lists.

See also [OPS-SLBS-1].

Parameters

- **lst** (*list*) – List to be split.
- **sub_len** (*int*) – Length of each sub-list.

Returns

A generator yielding sub-lists of length `sub_len` from `lst`.

Return type

Generator[list]

Examples:

```
>>> from pyhelpers.ops import split_list_by_size
>>> lst_ = list(range(0, 10))
>>> lst_
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lists = split_list_by_size(lst_, sub_len=3)
>>> list(lists)
[[0, 1, 2], [3, 4, 5], [6, 7, 8], [9]]
```

split_list

pyhelpers.ops.**split_list**(*lst*, *num_of_sub*)

Splits a list into a specified number of equally-sized sub-lists.

See also [OPS-SL-1].

Parameters

- **lst** (*list*) – List to be split.
- **num_of_sub** (*int*) – Number of sub-lists to create.

Returns

A generator yielding a total of `num_of_sub` sub-lists from `lst`.

Return type

Generator[list]

Examples:

```
>>> from pyhelpers.ops import split_list
>>> lst_ = list(range(0, 10))
>>> lst_
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lists = split_list(lst_, num_of_sub=3)
>>> list(lists)
[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9]]
```

split_iterable

pyhelpers.ops.**split_iterable**(*iterable*, *chunk_size*)

Splits an iterable into evenly sized chunks.

See also [OPS-SI-1].

Parameters

- **iterable** (*Iterable*) – Iterable object to be split.
- **chunk_size** (*int*) – Size of each chunk.

Returns

A generator yielding chunks of size `chunk_size` from `iterable`.

Return type

Generator[*Iterable*]

Examples:

```
>>> from pyhelpers.ops import split_iterable
>>> import pandas
>>> iterable_1 = list(range(0, 10))
>>> iterable_1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> iterable_1_ = split_iterable(iterable_1, chunk_size=3)
>>> type(iterable_1_)
generator
>>> for dat in iterable_1_:
...     print(list(dat))
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]
[9]
>>> iterable_2 = pandas.Series(range(0, 20))
>>> iterable_2
0      0
1      1
2      2
3      3
4      4
5      5
6      6
7      7
8      8
9      9
10    10
11    11
12    12
13    13
14    14
15    15
16    16
17    17
18    18
19    19
dtype: int64
>>> iterable_2_ = split_iterable(iterable_2, chunk_size=5)
```

(continues on next page)

(continued from previous page)

```
>>> for dat in iterable_2_:
...     print(list(dat))
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[10, 11, 12, 13, 14]
[15, 16, 17, 18, 19]
```

update_dict

pyhelpers.ops.**update_dict**(*dictionary*, *updates*, *inplace=False*)

Updates a (nested) dictionary with another dictionary.

See also [OPS-UD-1].

Parameters

- **dictionary** (*dict*) – The (nested) dictionary to be updated.
- **updates** (*dict*) – The dictionary containing updates.
- **inplace** (*bool*) – Whether to update the original `dictionary` in place; defaults to `False`.

Returns

The updated dictionary.

Return type

dict

Examples:

```
>>> from pyhelpers.ops import update_dict
>>> source_dict = {'key_1': 1}
>>> update_data = {'key_2': 2}
>>> upd_dict = update_dict(source_dict, updates=update_data)
>>> upd_dict
{'key_1': 1, 'key_2': 2}
>>> source_dict
{'key_1': 1}
>>> update_dict(source_dict, updates=update_data, inplace=True)
>>> source_dict
{'key_1': 1, 'key_2': 2}
>>> source_dict = {'key': 'val_old'}
>>> update_data = {'key': 'val_new'}
>>> upd_dict = update_dict(source_dict, updates=update_data)
>>> upd_dict
{'key': 'val_new'}
>>> source_dict = {'key': {'k1': 'v1_old', 'k2': 'v2'}}
>>> update_data = {'key': {'k1': 'v1_new'}}
>>> upd_dict = update_dict(source_dict, updates=update_data)
>>> upd_dict
{'key': {'k1': 'v1_new', 'k2': 'v2'}}
>>> source_dict = {'key': {'k1': {}, 'k2': 'v2'}}
>>> update_data = {'key': {'k1': 'v1'}}
>>> upd_dict = update_dict(source_dict, updates=update_data)
>>> upd_dict
{'key': {'k1': 'v1', 'k2': 'v2'}}
```

(continues on next page)

(continued from previous page)

```
>>> source_dict = {'key': {'k1': 'v1', 'k2': 'v2'}}
>>> update_data = {'key': {'k1': {}}}
>>> upd_dict = update_dict(source_dict, updates=update_data)
>>> upd_dict
{'key': {'k1': 'v1', 'k2': 'v2'}}
```

update_dict_keys

pyhelpers.ops.**update_dict_keys**(*dictionary*, *replacements=None*)

Updates keys in a (nested) dictionary based on a given replacements dictionary.

See also [OPS-UDK-1] and [OPS-UDK-2].

Parameters

- **dictionary** (*dict*) – The (nested) dictionary in which certain keys are to be updated.
- **replacements** (*dict | None*) – A dictionary in the form of {<current_key>: <new_key>} describing which keys are to be updated; defaults to None.

Returns

The dictionary with updated keys.

Return type

dict

Examples:

```
>>> from pyhelpers.ops import update_dict_keys
>>> source_dict = {'a': 1, 'b': 2, 'c': 3}
>>> upd_dict = update_dict_keys(source_dict, replacements=None)
>>> upd_dict  # remain unchanged
{'a': 1, 'b': 2, 'c': 3}
>>> repl_keys = {'a': 'd', 'c': 'e'}
>>> upd_dict = update_dict_keys(source_dict, replacements=repl_keys)
>>> upd_dict
{'d': 1, 'b': 2, 'e': 3}
>>> source_dict = {'a': 1, 'b': 2, 'c': {'d': 3, 'e': {'f': 4, 'g': 5}}}
>>> repl_keys = {'d': 3, 'f': 4}
>>> upd_dict = update_dict_keys(source_dict, replacements=repl_keys)
>>> upd_dict
{'a': 1, 'b': 2, 'c': {3: 3, 'e': {4: 4, 'g': 5}}}
```

get_dict_values

pyhelpers.ops.**get_dict_values**(*key*, *dictionary*)

Retrieves all values in a (nested) dictionary for a given key.

See also [OPS-GDV-1] and [OPS-GDV-2].

Parameters

- **key** (*any*) – The key to search for in the dictionary.
- **dictionary** (*dict*) – The (nested) dictionary to search within.

Returns

A generator yielding all values associated with the given `key` within the `dictionary`.

Return type

Generator[*Iterable*]

Examples:

```
>>> from pyhelpers.ops import get_dict_values
>>> key_ = 'key'
>>> target_dict_ = {'key': 'val'}
>>> val = get_dict_values(key_, target_dict_)
>>> list(val)
[['val']]
>>> key_ = 'k1'
>>> target_dict_ = {'key': {'k1': 'v1', 'k2': 'v2'}}
>>> val = get_dict_values(key_, target_dict_)
>>> list(val)
[['v1']]
>>> key_ = 'k1'
>>> target_dict_ = {'key': {'k1': ['v1', 'v1_1']}}
>>> val = get_dict_values(key_, target_dict_)
>>> list(val)
[['v1', 'v1_1']]
>>> key_ = 'k2'
>>> target_dict_ = {'key': {'k1': 'v1', 'k2': ['v2', 'v2_1']}}
>>> val = get_dict_values(key_, target_dict_)
>>> list(val)
[['v2', 'v2_1']]
```

remove_dict_keys

pyhelpers.ops.**remove_dict_keys**(*dictionary*, **keys*)

Removes multiple keys from a dictionary.

Parameters

- **dictionary** (*dict*) – The dictionary from which keys should be removed.
- **keys** – The keys to be removed from the dictionary.

Examples:

```
>>> from pyhelpers.ops import remove_dict_keys
>>> target_dict_ = {'k1': 'v1', 'k2': 'v2', 'k3': 'v3', 'k4': 'v4', 'k5': 'v5'}
>>> remove_dict_keys(target_dict_, 'k1', 'k3', 'k4')
>>> target_dict_
{'k2': 'v2', 'k5': 'v5'}
```

compare_dicts

pyhelpers.ops.**compare_dicts**(*dict1*, *dict2*)

Compares the differences between two dictionaries.

See also [OPS-CD-1].

Parameters

- **dict1** (*dict*) – The first dictionary for comparison.

- **dict2** (*dict*) – The second dictionary for comparison.

Returns

A tuple containing the main differences between `dict1` and `dict2`: modified items, common keys with different values, unchanged keys (same value in both dictionaries), new keys added in `dict2` and keys removed from `dict1`.

Return type

tuple

Examples:

```
>>> from pyhelpers.ops import compare_dicts
>>> d1 = {'a': 1, 'b': 2, 'c': 3}
>>> d2 = {'b': 2, 'c': 4, 'd': [5, 6]}
>>> items_modified, k_shared, k_unchanged, k_new, k_removed = compare_dicts(d1, d2)
>>> items_modified
{'c': [3, 4]}
>>> k_shared
['b', 'c']
>>> k_unchanged
['b']
>>> k_new
['d']
>>> k_removed
['a']
```

merge_dicts

pyhelpers.ops.**merge_dicts**(**dicts*)

Merges multiple dictionaries into a single dictionary.

Parameters

dicts (*dict*) – One or multiple dictionaries to merge.

Returns

A dictionary containing all elements from the input dictionaries.

Return type

dict

Examples:

```
>>> from pyhelpers.ops import merge_dicts
>>> dict_a = {'a': 1}
>>> dict_b = {'b': 2}
>>> dict_c = {'c': 3}
>>> merged_dict = merge_dicts(dict_a, dict_b, dict_c)
>>> merged_dict
{'a': 1, 'b': 2, 'c': 3}
>>> dict_c_ = {'c': 4}
>>> merged_dict = merge_dicts(merged_dict, dict_c_)
>>> merged_dict
{'a': 1, 'b': 2, 'c': [3, 4]}
>>> dict_1 = merged_dict
>>> dict_2 = {'b': 2, 'c': 4, 'd': [5, 6]}
>>> merged_dict = merge_dicts(dict_1, dict_2)
```

(continues on next page)

(continued from previous page)

```
>>> merged_dict
{'a': 1, 'b': 2, 'c': [[3, 4], 4], 'd': [5, 6]}
```

Tabular data

1anti-flashwhitewhite

detect_nan_for_str_column(data_frame[, ...])	Detects if a column with string type contains NaN values for a given dataframe.
create_rotation_matrix(theta)	Creates a 2D rotation matrix for counterclockwise rotation.
dict_to_dataframe(input_dict[, k, v])	Converts a dictionary to a dataframe.
swap_cols(array, c1, c2[, as_list])	Swaps positions of two columns in an array.
swap_rows(array, r1, r2[, as_list])	Swaps positions of two rows in an array.
np_shift(array, step[, fill_value])	Shifts an array by a desired number of rows.
downcast_numeric_columns(*data[, return_copy])	Downcast numeric types in pandas or polars DataFrames and Series to their optimal sizes.
flatten_columns(columns[, ignore_linear_level])	Flattens multi-level column names into single-level strings.

detect_nan_for_str_column

pyhelpers.ops.**detect_nan_for_str_column**(*data_frame*, *column_names=None*)

Detects if a column with string type contains NaN values for a given dataframe.

Parameters

- **data_frame** (*pandas.DataFrame*) – A dataframe to be examined.
- **column_names** (*collections.abc.Iterable | None*) – A sequence of column names to check; if column_names=None (default), all columns are checked.

Returns

Generator yielding position index of columns that contain NaN.

Return type

Generator

Examples:

```
>>> from pyhelpers.ops import detect_nan_for_str_column
>>> from pyhelpers._cache import example_dataframe
>>> dat = example_dataframe()
>>> dat
            Easting  Northing
City
London       530034    180381
Birmingham   406689    286822
Manchester   383819    398052
Leeds        582044    152953
>>> dat.loc['Leeds', 'Latitude'] = None
```

(continues on next page)

(continued from previous page)

```
>>> dat
            Easting  Northing
City
London       530034  180381.0
Birmingham   406689  286822.0
Manchester   383819  398052.0
Leeds        582044       NaN
>>> nan_col_pos = detect_nan_for_str_column(data_frame=dat, column_names=None)
>>> list(nan_col_pos)
[1]
```

create_rotation_matrix

pyhelpers.ops.**create_rotation_matrix**(*theta*)

Creates a 2D rotation matrix for counterclockwise rotation.

Parameters

theta (*float | int*) – Rotation angle in radians.

Returns

Rotation matrix of shape (2, 2).

Return type

numpy.ndarray

Note

- The rotation matrix is defined as:

```
[[cos(theta), -sin(theta)],
 [sin(theta), cos(theta)]]
```

- For counterclockwise rotation, the matrix rotates points in the positive direction (i.e. the positive x-axis rotates towards the positive y-axis).

Examples:

```
>>> from pyhelpers.ops import create_rotation_matrix
>>> rot_mat = create_rotation_matrix(theta=30)
>>> rot_mat
array([[-0.98803162,  0.15425145],
       [-0.15425145, -0.98803162]])
```

dict_to_dataframe

pyhelpers.ops.**dict_to_dataframe**(*input_dict*, *k='key'*, *v='value'*)

Converts a dictionary to a dataframe.

Parameters

- **input_dict** (*dict*) – Dictionary to be converted to a dataframe.
- **k** (*str*) – Column name for keys; defaults to `'key'`.
- **v** (*str*) – Column name for values; defaults to `'value'`.

Returns
Dataframe converted from the input dictionary.

Return type
pandas.DataFrame

Examples:

```
>>> from pyhelpers.ops import dict_to_dataframe
>>> test_dict = {'a': 1, 'b': 2}
>>> dat = dict_to_dataframe(input_dict=test_dict)
>>> dat
  key  value
0   a      1
1   b      2
```

swap_cols

pyhelpers.ops.**swap_cols**(*array*, *c1*, *c2*, *as_list=False*)

Swaps positions of two columns in an array.

Parameters

- **array** (*numpy.ndarray*) – An array.
- **c1** (*int*) – Index of the first (i.e. the c1-th) column to swap.
- **c2** (*int*) – Index of the second (i.e. the c2-th) column to swap.
- **as_list** (*bool*) – Whether to return a list instead of an array; defaults to False.

Returns
A new array or list where the positions of the c1-th and c2-th columns are swapped.

Return type
numpy.ndarray | list

Examples:

```
>>> from pyhelpers.ops import swap_cols
>>> from pyhelpers._cache import example_dataframe
>>> example_arr = example_dataframe(osgb36=True).to_numpy(dtype=int)
>>> example_arr
array([[530039, 180371],
       [406705, 286868],
       [383830, 398113],
       [430147, 433553]])
>>> # Swap the 0th and 1st columns
>>> new_arr = swap_cols(example_arr, c1=0, c2=1)
>>> new_arr
array([[180371, 530039],
       [286868, 406705],
       [398113, 383830],
       [433553, 430147]])
>>> new_list = swap_cols(example_arr, c1=0, c2=1, as_list=True)
>>> new_list
[[180371, 530039], [286868, 406705], [398113, 383830], [433553, 430147]]
```

swap_rows

pyhelpers.ops.**swap_rows**(*array*, *r1*, *r2*, *as_list=False*)

Swaps positions of two rows in an array.

Parameters

- **array** (*numpy.ndarray*) – An array.
- **r1** (*int*) – Index of the first (i.e. the r1-th) row to swap.
- **r2** (*int*) – Index of the second (i.e. the r2-th) row to swap.
- **as_list** (*bool*) – Whether to return a list instead of an array; defaults to False.

Returns

A new array or list where the positions of the r1-th and r2-th rows are swapped.

Return type

numpy.ndarray | list

Examples:

```
>>> from pyhelpers.ops import swap_rows
>>> from pyhelpers._cache import example_dataframe
>>> example_arr = example_dataframe(osgb36=True).to_numpy(dtype=int)
>>> example_arr
array([[406705, 286868],
       [530039, 180371],
       [383830, 398113],
       [430147, 433553]])
>>> # Swap the 0th and 1st rows
>>> new_arr = swap_rows(example_arr, r1=0, r2=1)
>>> new_arr
array([[406705, 286868],
       [530039, 180371],
       [383830, 398113],
       [430147, 433553]])
>>> new_list = swap_rows(example_arr, r1=0, r2=1, as_list=True)
>>> new_list
[[406705, 286868], [530039, 180371], [383830, 398113], [430147, 433553]]
```

np_shift

pyhelpers.ops.**np_shift**(*array*, *step*, *fill_value=nan*)

Shifts an array by a desired number of rows.

See also [OPS-NS-1].

Parameters

- **array** (*numpy.ndarray*) – An array of numbers.
- **step** (*int*) – Number of rows to shift. Positive value shifts downwards; negative shifts upwards.
- **fill_value** (*float | int*) – Value to fill missing rows due to the shift; defaults to NaN.

Returns
Shifted array.

Return type
numpy.ndarray

Examples:

```
>>> from pyhelpers.ops import np_shift
>>> from pyhelpers._cache import example_dataframe
>>> arr = example_dataframe(osgb36=True).to_numpy()
>>> arr
array([[530039.5588445, 180371.6801655],
       [406705.8870136, 286868.1666422],
       [383830.0390357, 398113.0558309],
       [430147.4473539, 433553.3271173]])
>>> np_shift(arr, step=-1)
array([[406705.8870136, 286868.1666422],
       [383830.0390357, 398113.0558309],
       [430147.4473539, 433553.3271173],
       [           nan,            nan]])
>>> np_shift(arr, step=1, fill_value=0)
array([[     0,      0],
       [530039, 180371],
       [406705, 286868],
       [383830, 398113]])
```

downcast_numeric_columns

pyhelpers.ops.**downcast_numeric_columns**(**data*, *return_copy=True*)

Downcast numeric types in pandas or polars DataFrames and Series to their optimal sizes.

This function processes multiple objects in one pass, converting integer columns to the smallest signed integer dtype (e.g. `int8`, `int16`) and floating-point columns to the smallest floating dtype (e.g. `float16`, `float32`) that can safely represent their values without data loss.

Parameters

- **data** (*pandas.DataFrame | pandas.Series | polars.DataFrame | polars.Series*) – One or more pandas or polars DataFrames/Series to optimize.
- **return_copy** (*bool*) – Whether to return a copy or modify the input in-place (where possible). Defaults to `True`.

Returns
New DataFrame(s) or Series with downcasted numeric columns.

Return type
None | pandas.DataFrame | pandas.Series | polars.DataFrame | polars.Series | tuple

Note

- Non-numeric and timedelta columns are automatically skipped.

- For polars, operations are inherently out-of-place, so `return_copy=False` primarily avoids an explicit early clone, but structural changes still yield new objects.

Examples:

```
>>> from pyhelpers.ops import downcast_numeric_columns
>>> from pyhelpers._cache import example_dataframe
>>> import polars as pl

>>> df1 = example_dataframe().copy()
>>> df1.dtypes
Longitude    float64
Latitude     float64
dtype: object

>>> df2 = example_dataframe().T.copy()
>>> df2.dtypes
City
London        float64
Birmingham    float64
Manchester    float64
Leeds         float64
dtype: object

>>> df11, df21 = downcast_numeric_columns(df1, df2)

>>> df11.dtypes
Longitude    float32
Latitude     float32
dtype: object

>>> df21.dtypes
City
London        float32
Birmingham    float32
Manchester    float32
Leeds         float32
dtype: object

>>> df1, df2 = map(pl.from_pandas, (df1, df2))

>>> df1.dtypes
[Float64, Float64]
>>> df2.dtypes
[Float64, Float64, Float64, Float64]

>>> df21, df22 = downcast_numeric_columns(df1, df2)
>>> df21.dtypes
[Float32, Float32]
>>> df22.dtypes
[Float32, Float32, Float32, Float32]
```

flatten_columns

pyhelpers.ops.**flatten_columns**(*columns*, *ignore_linear_level=False*)

Flattens multi-level column names into single-level strings.

This function processes column names (either as tuples or lists of strings) by joining them with underscores; optionally, it simplifies column names when they represent linear hierarchies.

Parameters

- **columns** (*Iterable[Sequence[str]]*) – Iterable of multi-level column names (each column is a sequence of strings).
- **ignore_linear_level** (*bool*) – If `ignore_linear_level=True`, the function simplifies linear hierarchies by omitting single-child levels; defaults to `False`.

Returns

List of flattened column names.

Return type

List[str]

Note

- For MultiIndex columns, passes the result of `df.columns.tolist()`.
- When `ignore_linear_level=True`, parent levels with only one child are collapsed.
- Preserves original order of columns.

Examples:

```
>>> from pyhelpers.ops import flatten_columns
>>> flatten_columns([('A', 'B'), ('A', 'C')])
['A_B', 'A_C']
>>> flatten_columns([('A', 'B', 'C'), ('A', 'B', 'D')], ignore_linear_level=True)
['A_B_C', 'A_B_D']
>>> flatten_columns([('A', 'C'), ('B', 'C')], ignore_linear_level=True)
['A', 'B']
>>> flatten_columns([('A',), ('B', 'C')], ignore_linear_level=True)
['A', 'B']
```

4.3.3 Web data manipulation

Internet-related utilities

1anti-flashwhitewhite

is_network_connected()	Checks whether the current machine is connected to the Internet.
is_url(url[, partially])	Checks if `url` is a valid URL.
is_url_connectable(url)	Checks if the current machine can connect to the given URL.

continues on next page

Table 11 – continued from previous page

init_requests_session(url[, max_retries, ...])	Instantiates a requests session with configurable retry behavior.
load_user_agent_strings([shuffled, ...])	Loads user-agent strings for popular web browsers.
get_user_agent_string([fancy])	Gets a random user-agent string for a specified browser.
fake_requests_headers([randomized])	Generates fake HTTP headers.
get_dynamic_url(page_url, pattern, base_url)	Gets a dynamic URL matching a specific regex pattern.

is_network_connected

pyhelpers.ops.**is_network_connected**()

Checks whether the current machine is connected to the Internet.

Returns

`True` if the Internet connection is currently working, `False` otherwise.

Return type

bool

Examples:

```
>>> from pyhelpers.ops import is_network_connected
>>> is_network_connected()  # Assuming we're currently connected to the Internet
True
```

is_url

pyhelpers.ops.**is_url**(*url*, *partially=False*)

Checks if `url` is a valid URL.

See also [OPS-IU-1].

Parameters

- **url** (*str*) – A string representing the URL to be checked.
- **partially** (*bool*) – Whether to consider the input as partially valid; defaults to `False`.

Returns

`True` if the given URL is a valid URL; `False` otherwise.

Return type

bool

Examples:

```
>>> from pyhelpers.ops import is_url
>>> is_url(url='https://github.com/mikeqfu/pyhelpers')
True
>>> is_url(url='github.com/mikeqfu/pyhelpers')
False
>>> is_url(url='github.com/mikeqfu/pyhelpers', partially=True)
```

(continues on next page)

(continued from previous page)

```
True
>>> is_url(url='github.com')
False
>>> is_url(url='github.com', partially=True)
True
>>> is_url(url='github', partially=True)
False
```

is_url_connectable

pyhelpers.ops.**is_url_connectable**(*url*)

Checks if the current machine can connect to the given URL.

Parameters

url (*str*) – A valid URL.

Returns

`True` if the machine can currently connect to the given URL, `False` otherwise.

Return type

bool

Examples:

```
>>> from pyhelpers.ops import is_url_connectable
>>> url_0 = 'https://www.python.org/'
>>> is_url_connectable(url_0)
True
>>> url_1 = 'https://www.python.org1/'
>>> is_url_connectable(url_1)
False
```

init_requests_session

pyhelpers.ops.**init_requests_session**(*url*, *max_retries=5*, *backoff_factor=0.1*, *retry_status='default'*, ***kwargs*)

Instantiates a requests session with configurable retry behavior.

Parameters

- **url** (*str*) – A valid URL to establish the session.
- **max_retries** (*int*) – Maximum number of retry attempts; defaults to `5`.
- **backoff_factor** (*float*) – Backoff factor for exponential backoff in retries; defaults to `0.1`.
- **retry_status** – HTTP status codes that trigger retries, derived from urllib3.util.Retry(); defaults to `[429, 500, 502, 503, 504]` when `retry_status='default'`.
- **kwargs** – [Optional] Additional parameters for the class urllib3.util.Retry().

Returns

A requests.Session() instance configured with the specified retry settings.

Return type

requests.Session

Examples:

```
>>> from pyhelpers.ops import init_requests_session
>>> url = 'https://www.python.org/static/community_logos/python-logo-master-v3-TM.png'
>>> s = init_requests_session(url)
>>> type(s)
requests.sessions.Session
```

load_user_agent_strings

pyhelpers.ops.**load_user_agent_strings**(*shuffled=False*, *flattened=False*, *update=False*, *verbose=False*)

Loads user-agent strings for popular web browsers.

This function retrieves a partially comprehensive list of user-agent strings for Chrome, Firefox, Safari, Edge, Internet Explorer, and Opera.

Parameters

- **shuffled** (*bool*) – Whether to randomly shuffle the user-agent strings; defaults to `False`.
- **flattened** (*bool*) – Whether to return a flattened list of all user-agent strings; defaults to `False`.
- **update** (*bool*) – Whether to update the backup data of user-agent strings; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to `False`.

Returns

Dictionary or list of user-agent strings, depending on the *flattened* parameter.

Return type

dict | list

Examples:

```
>>> from pyhelpers.ops import load_user_agent_strings
>>> uas = load_user_agent_strings()
>>> type(uas)
dict
>>> list(uas.keys())
['Chrome', 'Firefox', 'Safari', 'Edge', 'Internet Explorer', 'Opera']
>>> type(uas['Chrome'])
list
>>> uas['Chrome'][0]
'Chrome/15.0.860.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.20.25 (KHTML, li...
>>> uas_list = load_user_agent_strings(shuffled=True, flattened=True)
>>> type(uas_list)
list
>>> uas_list[0]  # a random one
'Mozilla/5.0 (X11; U; Linux x86_64; en-US) AppleWebKit/532.2 (KHTML, like Gecko) Chrome...
```

> **Note**
>
> The order of the elements in `uas_list` may be different every time we run the example as `shuffled=True`.

get_user_agent_string

`pyhelpers.ops.`**`get_user_agent_string`**(*fancy=None*, ***kwargs*)

Gets a random user-agent string for a specified browser.

Parameters

- **`fancy`** (*`None` | `str`*) – Name of the preferred browser; options include `'Chrome'`, `'Firefox'`, `'Safari'`, `'Edge'`, `'Internet Explorer'` and `'Opera'`. If `fancy=None` (default), the function returns a user-agent string from a randomly-selected browser among all available options.
- **`kwargs`** – [Optional] Additional parameters for the function `get_user_agent_strings()`.

Returns

A user-agent string for the specified browser.

Return type

str

Examples:

```
>>> from pyhelpers.ops import get_user_agent_string
>>> # Get a random user-agent string
>>> uas_0 = get_user_agent_string()
>>> uas_0
'Mozilla/5.0 (X11; U; Linux i686; de; rv:1.8.0.5) Gecko/20060731 Ubuntu/dapper-security...
>>> # Get a random Chrome user-agent string
>>> uas_1 = get_user_agent_string(fancy='Chrome')
>>> uas_1
'Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.13 (KHTML, like Gecko) Chrome/...
```

> **Note**
>
> In the above examples, the returned user-agent string is random and may be different every time of running the function.

fake_requests_headers

`pyhelpers.ops.`**`fake_requests_headers`**(*randomized=True*, ***kwargs*)

Generates fake HTTP headers.

This function creates HTTP headers suitable for use with requests.get. By default, it includes a randomly selected user-agent string from various popular browsers.

Parameters

- **`randomized`** (*`bool`*) – Whether to use a randomly selected user-agent string

from available browser data; defaults to `True`; if `randomized=False`, a random Chrome user-agent string is used.

- **kwargs** – [Optional] Additional parameters for the function *get_user_agent_string()*.

Returns
Fake HTTP headers.

Return type
dict

Examples:

```
>>> from pyhelpers.ops import fake_requests_headers
>>> fake_headers_1 = fake_requests_headers()
>>> fake_headers_1
{'User-Agent': 'Mozilla/5.0 (Windows; U; Windows NT 5.1; it-IT) AppleWebKit/525.1...
>>> fake_headers_2 = fake_requests_headers(randomized=False)
>>> fake_headers_2  # using a random Chrome user-agent string
{'User-Agent': 'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/532.1...
```

Note

- `fake_headers_1` may also be different every time we run the example. This is because the returned result is randomly chosen from a limited set of candidate user-agent strings, even though `randomized` is (by default) set to be `False`.
- By setting `randomized=True`, the function returns a random result from among all available user-agent strings of several popular browsers.

get_dynamic_url

`pyhelpers.ops.`**get_dynamic_url**(*page_url*, *pattern*, *base_url*)

Gets a dynamic URL matching a specific regex pattern.

This function is primarily used to retrieve direct download links for datasets (e.g. timestamped files) whose filenames may change. It parses the HTML of the `page_url` and looks for the first `<a>` tag with an `href` attribute that matches the provided `pattern`.

Parameters

- **page_url** (*str*) – The URL of the webpage to scrape for links.
- **pattern** (*str*) – A regular expression pattern to search for within the `href` attributes.
- **base_url** (*str*) – The base URL used to resolve relative links found on the page.

Returns
The fully qualified URL if a match is found; `None` otherwise.

Return type
str | None

Examples:

```
>>> from pyhelpers.ops import get_dynamic_url
>>> page_url = "https://northerngasopendataportal.co.uk/datasets/"
>>> pattern = ".*\.pdf"
>>> base_url = "https://northerngasopendataportal.co.uk"
>>> download_url = get_dynamic_url(page_url, pattern, base_url)
>>> print(download_url)
https://www.northerngasnetworks.co.uk/wp-content/uploads/2018/05/GDPR-Privacy-Statement...
```

File downloads utilities

1anti-flashwhitewhite

is_downloadable(url[, request_field])	Checks if a URL leads to a webpage where downloadable content is available.
download_file_from_url(url, path_to_file[, ...])	Downloads a file from a valid URL with optional progress tracking and validation.

is_downloadable

pyhelpers.ops.**is_downloadable**(*url*, *request_field='content-type'*, ***kwargs*)

Checks if a URL leads to a webpage where downloadable content is available.

Parameters

- **url** (*str*) – A valid URL.
- **request_field** (*str*) – Name of the field/header indicating the original media type of the resource; defaults to `'content-type'`.
- **kwargs** – [Optional] Additional parameters for the function requests.head().

Returns

`True` if the given URL leads to downloadable content, `False` otherwise.

Return type

bool

Examples:

```
>>> from pyhelpers.ops import is_downloadable
>>> url = 'https://www.python.org/static/community_logos/python-logo-master-v3-TM.png'
>>> is_downloadable(url)
True
>>> url = 'https://www.google.co.uk/'
>>> is_downloadable(url)
False
```

download_file_from_url

pyhelpers.ops.**download_file_from_url**(*url*, *path_to_file*, *if_exists='replace'*, *max_retries=5*, *requests_session_args=None*, *requests_headers=None*, *verbose=False*, *print_wrap_limit=None*, *total_records=None*, *chunk_multiplier=1*, *pbar_desc=None*, *pbar_format=None*, *pbar_color='green'*, *validate=True*, *stream_download=False*, *indent=None*, ***kwargs*)

Downloads a file from a valid URL with optional progress tracking and validation.

The function uses the `requests` library for the download and `tqdm` for progress. It supports memory-efficient streaming and handles HTTP retries internally.

See also [OPS-DFFU-1] and [OPS-DFFU-2].

Parameters

- **`url`** (*`str`*) – Valid URL pointing to a web resource.
- **`path_to_file`** (*`str | os.PathLike`*) – Path where the downloaded file will be saved; if it is a filename only, data will be saved in the current working directory.
- **`if_exists`** (*`str`*) – Action if the file already exists; options include `'replace'` (default - downloads and replaces the existing file) and `'pass'` (cancels the download and returns `None`).
- **`max_retries`** (*`int`*) – Maximum number of retries in case of download failures. Defaults to `5`.
- **`requests_session_args`** (*`dict | None`*) – [Optional] Additional parameters for initializing the requests session (e.g. *proxies*, *verify*). Defaults to `None`.
- **`requests_headers`** (*`dict | None`*) – [Optional] Custom headers to be included in the HTTP request. A default 'User-Agent' is automatically generated unless overridden.
- **`verbose`** (*`bool | int`*) – Whether to print progress and relevant information to the console; defaults to `False`. If `True`, a *tqdm* progress bar is displayed.
- **`print_wrap_limit`** (*`int | None`*) – Maximum length of the string before splitting into two lines; defaults to `None`, which disables splitting. If the string exceeds this value, e.g. `100`, it will be split at (before) `state_prep` to improve readability when printed.
- **`total_records`** (*`int | None`*) – The expected number of records (rows) in the dataset, used for progress tracking when the response's `Content-Length` header is unavailable; defaults to `None`.
- **`chunk_multiplier`** (*`int | float`*) – A factor by which the default chunk size (1MB) is multiplied; this can be adjusted to optimize download performance based on file size. Defaults to `1`.
- **`pbar_desc`** (*`str | None`*) – Custom description for the progress bar; when `desc=None`, it defaults to the filename.
- **`pbar_format`** (*`str | None`*) – Custom format for the progress bar.
- **`pbar_color`** (*`str | None`*) – Custom color of the progress bar (e.g. 'green', 'yellow'). Defaults to `'green'`.

- **validate** (*bool*) – Whether to validate the download integrity (check whether the downloaded file size matches the expected content length); defaults to `True`.
- **stream_download** (*bool*) – When *stream_download=True*, use streaming download (memory-efficient, preferred for large files or when *verbose=False*); When *stream_download=False*, the entire file content is loaded into memory first (simpler/faster for small files). Defaults to `False`.
- **indent** – Indentation level for the progress bar. Status messages (`"Saving..."`, `"Done"`) will be automatically indented further (2 spaces) to indicate hierarchy. Defaults to `None`.
- **kwargs** – [Optional] Additional parameters passed to the method tqdm.tqdm().

Returns

`None` upon successful completion or if the download is skipped (i.e. when `if_exists='pass'` and the file already exists).

Raises

ValueError – If `validate=True` and the downloaded file size is zero.

Examples:

```
>>> from pyhelpers.ops import download_file_from_url
>>> from pyhelpers.dirs import cd
>>> from PIL import Image
>>> import os
>>> url = 'https://www.python.org/static/community_logos/python-logo-master-v3-TM.png'
>>> path_to_img = cd("tests", "images", "ops-download_file_from_url-demo.png")
>>> # Check if "python-logo.png" exists at the specified path
>>> os.path.exists(path_to_img)
False
>>> # Download the .png file
>>> download_file_from_url(url, path_to_img, verbose=True, pbar_color='green')
Downloading "ops-download_file_from_url-demo.png" 100%|████████████| 83.6k/83.6k | ...
  Saving "ops-download_file_from_url-demo.png" to "./tests/images/" ... Done.
>>> # If download is successful, check again:
>>> os.path.exists(path_to_img)
True
>>> img = Image.open(path_to_img)
>>> img.show()  # as illustrated below
```

Figure 8: The Python Logo.

Note

- When `verbose=True`, the function requires tqdm.
- The function handles HTTP retries internally using a *requests* session adapter.

1anti-flashwhitewhite

GitHubFileDownloader(repo_url[, ...])	Downloads files from GitHub repositories.

GitHubFileDownloader

class pyhelpers.ops.**GitHubFileDownloader**(*repo_url*, *flatten_files=False*, *output_dir=None*)

Downloads files from GitHub repositories.

This class facilitates downloading files from a specified GitHub repository URL.

Parameters

- **repo_url** (*str*) – URL of the GitHub repository to download from; it can be a path to a specific *blob* or *tree* location.
- **flatten_files** (*bool*) – Whether to flatten the directory structure by pulling all files into the root folder; defaults to `False`.
- **output_dir** (*str | os.PathLike | None*) – Output directory where downloaded files will be saved; defaults to `None`, meaning files will be saved in the current directory.

Variables

- **repo_url** (*str*) – URL of the GitHub repository.
- **flatten_files** (*bool*) – Whether to flatten the directory structure (i.e. pull the contents of all subdirectories into the root folder); defaults to `False`.
- **output_dir** (*str | None*) – Output directory path; defaults to `None`.
- **api_url** (*str*) – URL of the GitHub repository compatible with GitHub's REST API.
- **download_path** (*str*) – Pathname for downloading files.
- **total_files** (*int*) – Total number of files under the given directory.

Examples:

```
>>> from pyhelpers.ops import GitHubFileDownloader
>>> from pyhelpers.dirs import delete_dir
>>> output_dir = "tests/temp"
>>> # Download a single file
>>> repo_url = 'https://github.com/mikeqfu/pyhelpers/blob/master/tests/data/dat.csv'
>>> downloader = GitHubFileDownloader(repo_url, output_dir=output_dir)
>>> downloader.download()
Downloaded to: "./tests/temp/dat.csv"
```

(continues on next page)

(continued from previous page)

```
1
>>> # Download a directory
>>> repo_url = 'https://github.com/mikeqfu/pyhelpers/blob/master/tests/data'
>>> downloader = GitHubFileDownloader(repo_url, output_dir=output_dir)
>>> downloader.download()
Downloaded to: "./tests/temp/tests/data/csr_mat.npz"
Downloaded to: "./tests/temp/tests/data/dat.csv"
Downloaded to: "./tests/temp/tests/data/dat.feather"
Downloaded to: "./tests/temp/tests/data/dat.joblib"
Downloaded to: "./tests/temp/tests/data/dat.json"
Downloaded to: "./tests/temp/tests/data/dat.ods"
Downloaded to: "./tests/temp/tests/data/dat.pickle"
Downloaded to: "./tests/temp/tests/data/dat.pickle.bz2"
Downloaded to: "./tests/temp/tests/data/dat.pickle.gz"
Downloaded to: "./tests/temp/tests/data/dat.pickle.xz"
Downloaded to: "./tests/temp/tests/data/dat.txt"
Downloaded to: "./tests/temp/tests/data/dat.xlsx"
Downloaded to: "./tests/temp/tests/data/zipped.7z"
Downloaded to: "./tests/temp/tests/data/zipped.txt"
Downloaded to: "./tests/temp/tests/data/zipped.zip"
16
>>> downloader = GitHubFileDownloader(
...     repo_url, flatten_files=True, output_dir=output_dir)
>>> downloader.download()
Downloaded to: "./tests/temp/csr_mat.npz"
Downloaded to: "./tests/temp/dat.csv"
Downloaded to: "./tests/temp/dat.feather"
Downloaded to: "./tests/temp/dat.joblib"
Downloaded to: "./tests/temp/dat.json"
Downloaded to: "./tests/temp/dat.ods"
Downloaded to: "./tests/temp/dat.pickle"
Downloaded to: "./tests/temp/dat.pickle.bz2"
Downloaded to: "./tests/temp/dat.pickle.gz"
Downloaded to: "./tests/temp/dat.pickle.xz"
Downloaded to: "./tests/temp/dat.txt"
Downloaded to: "./tests/temp/dat.xlsx"
Downloaded to: "./tests/temp/zipped.7z"
Downloaded to: "./tests/temp/zipped.txt"
Downloaded to: "./tests/temp/zipped.zip"
16
>>> delete_dir(output_dir)
To delete the directory "./tests/temp/" (Not empty)
? [No]|Yes: yes
```

Methods

1anti-flashwhitewhite

check_url(url)	Checks if the scheme of the provided `url` is valid.
create_url(url)	Creates a URL compatible with GitHub's REST API from the given URL.
download([api_url])	Downloads files from the specified GitHub `api_url`.

continues on next page

Table 14 – continued from previous page

download_single_file(file_url, dir_out)	Downloads a single file from the specified `file_url` to the `dir_out` directory.

GitHubFileDownloader.check_url

classmethod GitHubFileDownloader.**check_url**(*url*)

Checks if the scheme of the provided `url` is valid.

Parameters

url (*str*) – The target URL.

See also

- Examples for *GitHubFileDownloader.download()*.

GitHubFileDownloader.create_url

classmethod GitHubFileDownloader.**create_url**(*url*)

Creates a URL compatible with GitHub's REST API from the given URL.

Handles *blob* or *tree* paths.

Parameters

url (*str*) – URL.

Returns

Tuple containing the URL of the GitHub repository and the pathname for downloading a file.

Return type

tuple

Examples:

```
>>> from pyhelpers.ops import GitHubFileDownloader
>>> main_page = 'https://github.com/mikeqfu'
>>> output_dir = "tests/temp"
>>> url = f'{main_page}/pyhelpers/blob/master/tests/data/dat.csv'
>>> downloader = GitHubFileDownloader(url, output_dir=output_dir)
>>> api_url, download_path = downloader.create_url(url)
>>> api_url
'https://api.github.com/repos/mikeqfu/pyhelpers/contents/tests/data/dat.csv?ref=mas...
>>> download_path
'tests/data/dat.csv'
>>> url = f'{main_page}/smart-home-product-reviews-analysis/tree/master/demos'
>>> downloader = GitHubFileDownloader(url, output_dir=output_dir)
>>> api_url, download_path = downloader.create_url(url)
>>> api_url
'https://api.github.com/repos/mikeqfu/smart-home-product-reviews-analysis/contents/...
>>> download_path
'demos'
```

GitHubFileDownloader.download

GitHubFileDownloader.**download**(*api_url=None*)

Downloads files from the specified GitHub `api_url`.

Parameters

api_url (*str | None*) – GitHub API URL for downloading files; defaults to `None`.

Returns

Total number of files downloaded under the given directory.

Return type

int

Examples:

```
>>> from pyhelpers.ops import GitHubFileDownloader
>>> from pyhelpers.dirs import delete_dir
>>> import tempfile
>>> test_output_dir = tempfile.mkdtemp()
>>> test_url = "https://github.com/mikeqfu/pyhelpers/blob/master/tests/data"
>>> downloader = GitHubFileDownloader(test_url, output_dir=test_output_dir)
>>> downloader.download()
Downloaded to: "<TEMP_DIR>/tests/data/csr_mat.npz"
Downloaded to: "<TEMP_DIR>/tests/data/dat.csv"
Downloaded to: "<TEMP_DIR>/tests/data/dat.feather"
Downloaded to: "<TEMP_DIR>/tests/data/dat.joblib"
Downloaded to: "<TEMP_DIR>/tests/data/dat.json"
Downloaded to: "<TEMP_DIR>/tests/data/dat.ods"
Downloaded to: "<TEMP_DIR>/tests/data/dat.pickle"
Downloaded to: "<TEMP_DIR>/tests/data/dat.pickle.bz2"
Downloaded to: "<TEMP_DIR>/tests/data/dat.pickle.gz"
Downloaded to: "<TEMP_DIR>/tests/data/dat.pickle.xz"
Downloaded to: "<TEMP_DIR>/tests/data/dat.txt"
Downloaded to: "<TEMP_DIR>/tests/data/dat.xlsx"
Downloaded to: "<TEMP_DIR>/tests/data/zipped.7z"
Downloaded to: "<TEMP_DIR>/tests/data/zipped.txt"
Downloaded to: "<TEMP_DIR>/tests/data/zipped.zip"
>>> delete_dir(test_output_dir, confirmation_required=False, verbose=True)
Deleting "<TEMP_DIR>/" ... Done.
```

GitHubFileDownloader.download_single_file

GitHubFileDownloader.**download_single_file**(*file_url*, *dir_out*)

Downloads a single file from the specified `file_url` to the `dir_out` directory.

Parameters

- **file_url** (*str*) – URL of the file to be downloaded.
- **dir_out** (*str*) – Directory path where the downloaded file will be saved.

See also

- Examples for *GitHubFileDownloader.download()*.

API-related utilities

1anti-flashwhitewhite

CrossRefOrcid([my_name, requests_headers])	A class to interact with the ORCID Public API and CrossRef API to retrieve metadata about academic publications.

CrossRefOrcid

class pyhelpers.ops.**CrossRefOrcid**(*my_name=''*, *requests_headers=None*)

A class to interact with the ORCID Public API and CrossRef API to retrieve metadata about academic publications.

Variables

- **my_name** (*str*) – My name to be bolded in the author list; defaults to `"Fu, Qian"`.
- **requests_headers** (*dict*) – HTTP headers used for making requests to external APIs; defaults to `{"Accept": "application/json"}`.

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> co.ORCID_PUBLIC_API_ENDPOINT
'https://pub.orcid.org/v3.0'
>>> co.CROSSREF_REST_API_ENDPOINT
'https://api.crossref.org/works'
>>> list(co.CITATION_STYLES)
['APA', 'MLA', 'Chicago', 'Harvard', 'IEEE', 'Vancouver']
```

Attributes

1anti-flashwhitewhite

CITATION_STYLES	Templates of various styles of citations.
CROSSREF_REST_API_ENDPOINT	The CrossRef REST API endpoint for querying metadata about published works.
ORCID_PUBLIC_API_ENDPOINT	The ORCID environment for the Public API on the production ORCID Registry.
ZENODO_REST_API_ENDPOINT	The Zenodo REST API endpoint for access to records stored in Zenodo.

CrossRefOrcid.CITATION_STYLES

CrossRefOrcid.**CITATION_STYLES: dict = {'APA': '{author} ({year}). {title}. {publication}, {pages}. https://doi.org/{doi}', 'Chicago': '{author} {year}. "{title}." {publication}, {pages}. https://doi.org/{doi}', 'Harvard': "{author} ({year}) '{title}', {publication}, pp. {pages}. Available at: https://doi.org/{doi} [Accessed {accessed}].", 'IEEE': '[{index}] {author}, "{title}," {publication}, pp. {pages}, {year}, doi:{doi}.', 'MLA': '{author} "{title}." {publication}, {year}, pp. {pages}.', 'Vancouver': '{author} {title}. {publication}. {year};{pages}. doi:{doi}.'}**

Templates of various styles of citations.

CrossRefOrcid.CROSSREF_REST_API_ENDPOINT

CrossRefOrcid.**CROSSREF_REST_API_ENDPOINT: str = 'https://api.crossref.org/works'**

The CrossRef REST API endpoint for querying metadata about published works.

CrossRefOrcid.ORCID_PUBLIC_API_ENDPOINT

CrossRefOrcid.**ORCID_PUBLIC_API_ENDPOINT: str = 'https://pub.orcid.org/v3.0'**

The ORCID environment for the Public API on the production ORCID Registry.

CrossRefOrcid.ZENODO_REST_API_ENDPOINT

CrossRefOrcid.**ZENODO_REST_API_ENDPOINT: str = 'https://zenodo.org/api/records'**

The Zenodo REST API endpoint for access to records stored in Zenodo.

Methods

1anti-flashwhitewhite

fetch_orcid_works(orcid_id[, work_types, ...])	Fetch recent works from ORCID and enrich with metadata from DOI.
fetch_references(orcid_id[, work_types, ...])	Fetch and format references from an ORCID profile.
format_references(ref_data[, style])	Format multiple references in a given citation style.
get_list_of_works(orcid_id)	Get a list of works from an ORCID profile.
get_metadata_from_doi(doi)	Get full metadata from CrossRef using DOI.
get_orcid_profile(orcid_id[, section, verbose])	Get the ORCID profile for a given ORCID ID.
update_references(orcid_id[, work_types, ...])	Orchestrate the fetching and writing of references to a Markdown file.

CrossRefOrcid.fetch_orcid_works

CrossRefOrcid.**fetch_orcid_works**(*orcid_id*, *work_types=None*, *recent_years=2*)

Fetch recent works from ORCID and enrich with metadata from DOI.

Parameters

- **orcid_id** (*str*) – ORCID iD of the researcher.
- **work_types** (*str | list | None*) – Work type(s) to include (e.g.

'journal-article' or ['book', 'conference-paper']); defaults to None.

- **recent_years** (*int*) – Number of recent years to include; defaults to 2.

Returns
A list of extracted reference dictionaries.

Return type
list[dict]

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> orcid_id = '0000-0002-6502-9934'
>>> ref_data = co.fetch_orcid_works(orcid_id)  # Past two years
>>> type(ref_data)
list
>>> # ref_data = co.fetch_orcid_works(orcid_id, recent_years=5)  # Past five years
>>> # for ref_dat in ref_data:
... #     print(ref_dat)
```

CrossRefOrcid.fetch_references

CrossRefOrcid.**fetch_references**(*orcid_id*, *work_types=None*, *recent_years=2*, *style='APA'*)
Fetch and format references from an ORCID profile.

Parameters

- **orcid_id** (*str*) – ORCID iD of the researcher.
- **work_types** (*str | list | None*) – Work type(s) to include; defaults to None.
- **recent_years** (*int*) – Number of recent years to include; defaults to 2.
- **style** (*str*) – The citation style to use; defaults to 'APA'.

Returns
A list of formatted citation strings.

Return type
list[str]

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> orcid_id = '0000-0002-6502-9934'
>>> references = co.fetch_references(orcid_id)
>>> references[-1]
'Fu, Q., Easton, J.M., Burrow, M.P.N. (2024). Development of an Integrated Computing...
```

CrossRefOrcid.format_references

CrossRefOrcid.**format_references**(*ref_data*, *style='APA'*)
Format multiple references in a given citation style.

Parameters

- **ref_data** (*list | dict | Iterable*) – A collection of reference dictionaries or a single reference.
- **style** (*str*) – The citation style to use; defaults to `'APA'`.

Returns
A list of formatted citation strings.

Return type
list[str]

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> orcid_id = '0000-0002-6502-9934'
>>> ref_data = co.fetch_orcid_works(orcid_id)  # Past two years
>>> references = co.format_references(ref_data, style='APA')
>>> references[-1]
'Fu, Q., Easton, J.M., Burrow, M.P.N. (2024). Development of an Integrated Computing...
```

CrossRefOrcid.get_list_of_works

CrossRefOrcid.**get_list_of_works**(*orcid_id*)

Get a list of works from an ORCID profile.

Parameters
orcid_id (*str*) – ORCID ID of the researcher.

Returns
A list of works with titles and years.

Return type
list[str]

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> orcid_id = '0000-0002-6502-9934'
>>> list_of_works = co.get_list_of_works(orcid_id)
>>> list_of_works[-1]
'A Review on Transit Assignment Modelling Approaches to Congested Networks: A New...
```

CrossRefOrcid.get_metadata_from_doi

CrossRefOrcid.**get_metadata_from_doi**(*doi*)

Get full metadata from CrossRef using DOI.

Parameters
doi (*str*) – The DOI of the publication.

Returns
A dictionary containing metadata.

Return type
dict

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> doi = 'https://doi.org/10.1016/j.jii.2024.100729'
>>> co.get_metadata_from_doi(doi)
{'journal': 'Journal of Industrial Information Integration',
 'conference': '',
 'volume': '42',
 'issue': '',
 'pages': '100729',
 'authors': 'Fu, Qian, Nicholson, Gemma L., Easton, John M.'}
```

CrossRefOrcid.get_orcid_profile

CrossRefOrcid.**get_orcid_profile**(*orcid_id*, *section=None*, *verbose=False*)

Get the ORCID profile for a given ORCID ID.

Parameters

- **orcid_id** (*str*) – ORCID iD of the researcher.
- **section** (*str | None*) – Specific section of the profile (e.g. 'works'); defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to False.

Returns

A dictionary containing profile data, or None if an error occurs.

Return type

dict | None

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> co = CrossRefOrcid()
>>> orcid_id = '0000-0002-6502-9934'
>>> profile_data = co.get_orcid_profile(orcid_id)
>>> list(profile_data.keys())
['orcid-identifier',
 'preferences',
 'history',
 'person',
 'activities-summary',
 'path']
```

CrossRefOrcid.update_references

CrossRefOrcid.**update_references**(*orcid_id*, *work_types=None*, *recent_years=2*, *style='APA'*, *file_path='tests/README.md'*, *heading='Recent publications'*, *heading_level=3*, *heading_suffix=':'*, *max_entries=100*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Orchestrate the fetching and writing of references to a Markdown file.

Parameters

- **orcid_id** (*str*) – ORCID iD of the researcher.

- **work_types** (*str | list | None*) – Work type(s) to include; defaults to None.
- **recent_years** (*int*) – Number of recent years to include; defaults to 2.
- **style** (*str*) – The citation style to use; defaults to 'APA'.
- **file_path** (*str | os.PathLike*) – Path to the Markdown file; defaults to "README.md".
- **heading_level** (*int*) – The level of the heading under which the contents are to be updated; defaults to 3.
- **heading** (*str*) – The Markdown heading for the references section; defaults to "Recent publications".
- **heading_suffix** (*str | None*) – Suffix to the heading; defaults to ":".
- **max_entries** (*int | None*) – The maximum number of references to be included; defaults to 100.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to True.
- **verbose** (*bool*) – If True, prints status messages; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.ops import CrossRefOrcid
>>> import os
>>> co = CrossRefOrcid()
>>> orcid_id = '0000-0002-6502-9934'
>>> co.update_references(orcid_id, verbose=True)
To write references in "./tests/README.md"
? [No]|Yes: yes
Writing "Recent publications" in "./tests/README.md" ... Done.
>>> co.update_references(orcid_id, recent_years=5, max_entries=5, verbose=True)
To update references in "./tests/README.md"
? [No]|Yes: yes
Updating "Recent publications" in "./tests/README.md" ... Done.
>>> os.remove("./tests/README.md")
```

4.3.4 Misc general utilities

1anti-flashwhitewhite

confirmed([prompt, confirmation_required, resp])	Prompts user for confirmation to proceed.
get_obj_attr(obj[, col_names, as_dataframe])	Retrieves main attributes of an object.
eval_dtype(str_val)	Converts a string representation to its intrinsic data type.

continues on next page

Table 18 – continued from previous page

is_visual_object(obj)	Determines if an object is a known figure or image type.
hash_password(password[, salt, salt_size, ...])	Hashes a password using hashlib.pbkdf2_hmac (PBKDF2 algorithm with HMAC-SHA256).
verify_password(password, salt, key[, ...])	Verifies if a password matches the provided salt and key.
func_running_time(func)	Decorator to measure the execution time of a function or class method.
get_git_branch([verbose])	Gets the current Git branch name.
get_ansi_color_code(colors[, ...])	Returns the ANSI escape code(s) for the given color name(s) and/or style(s).
get_project_structure(start_path[, ...])	Prints and/or writes the directory and file structure of a given project folder starting from `start_path`.

confirmed

pyhelpers.ops.**confirmed**(*prompt=None*, *confirmation_required=True*, *resp=False*)

Prompts user for confirmation to proceed.

See also [OPS-C-1].

Parameters

- **prompt** (*str | None*) – Message prompting a response (Yes/No); defaults to `None`.
- **confirmation_required** (*bool*) – Whether to require user confirmation to proceed; defaults to `True`.
- **resp** (*bool*) – Default response if no user input; defaults to `False` (No).

Returns

User response indicating confirmation (`True`) or denial (`False`).

Return type

bool

Examples:

```
>>> from pyhelpers.ops import confirmed
>>> if confirmed(prompt="Testing if the function works?", resp=True):
...     print("Passed.")
Testing if the function works? [Yes]|No: yes
Passed.
```

get_obj_attr

pyhelpers.ops.**get_obj_attr**(*obj*, *col_names=None*, *as_dataframe=False*)

Retrieves main attributes of an object.

Parameters

- **obj** (*object*) – Object from which to retrieve attributes, e.g. an instance of a class.

- **col_names** (*list | None*) – List of column names for renaming the returned dataframe when `as_dataframe=True`; defaults to `None`.
- **as_dataframe** (*bool*) – Whether to return the data in tabular format (dataframe); defaults to `False`.

Returns

The main attributes of the given `obj`.

Return type

list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.ops import get_obj_attr
>>> from pyhelpers.dbms import PostgreSQL
>>> postgres = PostgreSQL()
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> obj_attr = get_obj_attr(postgres, as_dataframe=True)
>>> obj_attr.head()
          attribute       value
0  DEFAULT_DATABASE    postgres
1   DEFAULT_DIALECT  postgresql
2    DEFAULT_DRIVER    psycopg2
3      DEFAULT_HOST   localhost
4      DEFAULT_PORT        5432
>>> obj_attr.attribute.to_list()
['DEFAULT_DATABASE',
 'DEFAULT_DIALECT',
 'DEFAULT_DRIVER',
 'DEFAULT_HOST',
 'DEFAULT_PORT',
 'DEFAULT_SCHEMA',
 'DEFAULT_USERNAME',
 'address',
 'database_info',
 'database_name',
 'engine',
 'host',
 'port',
 'url',
 'username']
```

eval_dtype

`pyhelpers.ops.`**eval_dtype**(*str_val*)

Converts a string representation to its intrinsic data type.

Parameters

str_val (*str*) – String representation of a value.

Returns

Value converted to its intrinsic data type.

Return type

Any

Examples:

```
>>> from pyhelpers.ops import eval_dtype
>>> val_1 = '1'
>>> origin_val = eval_dtype(val_1)
>>> origin_val
1
>>> val_2 = '1.1.1'
>>> origin_val = eval_dtype(val_2)
>>> origin_val
'1.1.1'
```

is_visual_object

pyhelpers.ops.**is_visual_object**(*obj*)

Determines if an object is a known figure or image type.

This function checks for visual objects post-conversion, supporting Matplotlib Figures/Axes, Seaborn grids, Plotly figures, and PIL Images. It is designed to be library-agnostic by checking for common methods ('.show' or '.savefig') and using string-based type inspection to avoid explicit imports of heavy libraries.

Parameters

obj (*Any*) – The object to inspect.

Returns

True if the object is identified as a figure or image, else False.

Return type

bool

Examples:

```
>>> from pyhelpers.ops import is_visual_object
>>> import matplotlib.pyplot as plt
>>> from PIL import Image
>>> import numpy as np

>>> # Case 1: Matplotlib Figure
>>> fig, ax = plt.subplots()
>>> is_visual_object(fig)
True

>>> # Case 2: PIL Image converted from array
>>> img = Image.fromarray(np.zeros((10, 10), dtype=np.uint8))
>>> is_visual_object(img)
True

>>> # Case 3: Raw data (not converted)
>>> is_visual_object([1, 2, 3])
False
```

hash_password

pyhelpers.ops.**hash_password**(*password*, *salt=None*, *salt_size=None*, *iterations=None*, *ret_hash=True*, ***kwargs*)

Hashes a password using hashlib.pbkdf2_hmac (PBKDF2 algorithm with HMAC-SHA256).

See also [OPS-HP-1].

Parameters

- **password** (*str | int | float | bytes*) – Password to be hashed.
- **salt** (*bytes | str | None*) – Optional salt data for hashing; if `salt=None` (default), it is generated by os.urandom(), which depends on `salt_size`; see also [OPS-HP-2].
- **salt_size** (*int | None*) – Size of the `salt` in bytes, which is equivalent to `size` of the function os.urandom(); defaults to `128` if not specified.
- **iterations** (*int | None*) – Number of iterations for PBKDF2, which is equivalent to `size` of the function hashlib.pbkdf2_hmac(); defaults to `100000` if not specified.
- **ret_hash** (*bool*) – Whether to return the salt and key; defaults to `True`.
- **kwargs** – [Optional] Additional parameters for the function hashlib.pbkdf2_hmac().

Returns

Hashed password and salt as bytes (returned only if `ret_hash=True`).

Return type

bytes | None

Examples:

```
>>> from pyhelpers.ops import hash_password, verify_password
>>> test_pwd = 'test%123'
>>> salt_size_ = 16
>>> salt_and_key = hash_password(password=test_pwd, salt_size=salt_size_)  # salt and key
>>> salt_data, key_data = salt_and_key[:salt_size_].hex(), salt_and_key[salt_size_:].hex()
>>> verify_password(password=test_pwd, salt=salt_data, key=key_data)
True
>>> test_pwd = b'test%123'
>>> sk = hash_password(password=test_pwd)
>>> salt_data, key_data = sk[:128].hex(), sk[128:].hex()
>>> verify_password(password=test_pwd, salt=salt_data, key=key_data)
True
```

verify_password

pyhelpers.ops.**verify_password**(*password*, *salt*, *key*, *iterations=None*)

Verifies if a password matches the provided salt and key.

Parameters

- **password** (*str | int | float | bytes*) – Password to be verified.
- **salt** (*bytes | str*) – Salt used during hashing to enhance security; see also [OPS-HP-1].
- **key** (*bytes | str*) – Hashed key generated using PBKDF2 algorithm (produced by hashlib.pbkdf2_hmac()).
- **iterations** (*int | None*) – Number of iterations used in PBKDF2; defaults to `100000` if not specified.

Returns
`True` if the input password matches the hashed key and salt, `False` otherwise.

Return type
bool

See also

- Examples of the function *pyhelpers.ops.hash_password()*.

func_running_time

pyhelpers.ops.**func_running_time**(*func*)

Decorator to measure the execution time of a function or class method.

Parameters
func (*Callable*) – Function or class method to be decorated.

Returns
Decorated function or class method that measures its own execution time.

Return type
Callable

Examples:

```
>>> from pyhelpers.ops import func_running_time
>>> import time
>>> @func_running_time
>>> def test_func():
...     print("Testing if the function works.")
...     time.sleep(3)
>>> test_func()
INFO Begin to run function: test_func ...
Testing if the function works.
INFO Finished running function: test_func, total: 3s
```

get_git_branch

pyhelpers.ops.**get_git_branch**(*verbose=False*)

Gets the current Git branch name.

Parameters
verbose (*bool | int*) – Whether to print relevant information in console; defaults to `False`.

Returns
The name of the currently checked-out Git branch.

Return type
str

Examples:

```
>>> from pyhelpers.ops import get_git_branch
>>> get_git_branch()
'master'
```

get_ansi_color_code

pyhelpers.ops.**get_ansi_color_code**(*colors*, *show_valid_colors=False*, *concatenated=True*)

Returns the ANSI escape code(s) for the given color name(s) and/or style(s).

The function handles both single attribute requests and compound sequences (e.g. `['red', 'blue']`) by concatenating the codes into a single escape sequence string when appropriate.

Parameters

- **colors** (*str | list[str] | tuple[str]*) – A single color/style name (str) or a sequence of names (e.g. `'red'`, `'bold'`, `['red', 'bg_blue']`).
- **show_valid_colors** (*bool*) – If `True`, returns a tuple containing the final output (string or list) and a set of all valid color/style names.
- **concatenated** (*bool*) – If `True` (default), multiple requested codes are concatenated into a single string (e.g. `'\u001b[31m\u001b[1m'`). If `False`, a list of individual escape code strings is returned (e.g. `['\u001b[31m', '\u001b[1m']`).

Returns

The ANSI escape code(s). This is a single string if `concatenated=True`, a list of strings if `concatenated=False` and multiple items were requested, or a tuple if `show_valid_colors=True`.

Return type

str | list[str] | tuple[Union[str, list[str]], set[str]]

Raises

ValueError – If an invalid color or style name is provided.

Examples:

```
>>> from pyhelpers.ops import get_ansi_color_code
>>> get_ansi_color_code('red')  # \u001b[31m
'\033[31m'
>>> get_ansi_color_code(['red', 'blue'])  # \u001b[31m\u001b[34m
['\033[31m', '\033[34m']
>>> get_ansi_color_code(['red', 'blue'], concatenated=False)
'\033[31m\033[34m'
>>> get_ansi_color_code('invalid_color')
Traceback (most recent call last):
    ...
ValueError: 'invalid_color' is not a valid color name.
>>> get_ansi_color_code('red', show_valid_colors=True)  # ('\u001b[31m', ...
('\033[31m', {'bg_black', 'bg_blue', 'bg_bright_black', 'bg_bright_blue', ...
```

get_project_structure

pyhelpers.ops.**get_project_structure**(*start_path*, *ignore_dirs=None*, *out_file=None*, *encoding='utf-8'*, *print_in_console=True*)

Prints and/or writes the directory and file structure of a given project folder starting from `start_path`.

The output shows a tree-like hierarchy with branch symbols for better readability.

Parameters

- **start_path** (*str | pathlib.Path*) – Path to the root directory whose structure to visualize; can be absolute or relative to the current working directory.
- **ignore_dirs** (*None | typying.Iterable*) – Optional set of directory names to ignore during traversal; defaults to `{'__pycache__'}`.
- **out_file** (*str | None*) – Optional file path to write the structure output. If `None` (default), no file will be written. If specified, the structure will be saved to the specified file path.
- **encoding** (*str*) – The encoding to use when writing to the output file; defaults to `'utf-8'`.
- **print_in_console** (*bool*) – Whether to print the structure to the console; defaults to `True`.

Examples:

```
>>> from pyhelpers.ops import get_project_structure
>>> get_project_structure(start_path="pyhelpers")
>>> get_project_structure("my_project", print_in_console=False, out_file="structure.txt")
```

4.4 store

Performing operations on file-like objects.

These operations include saving and loading data, as well as other relevant tasks.

Utilities

1anti-flashwhitewhite

_check_saving_path(path[, verbose, ...])	Verify a file path before saving, creates directories, and manages console output.
_autofit_column_width(excel_writer, ...)	Adjust the column widths in an Excel spreadsheet based on the content length.
_check_loading_path(path[, verbose, ...])	Verify a file path for loading and prints status to the console.
_set_index(data[, index_col])	Set the index of a dataframe using column names or integer positions.

4.4.1 _check_saving_path

pyhelpers.store.**_check_saving_path**(*path*, *verbose=False*, *msg_prefix=''*, *state_verb='Saving'*, *state_prep='to'*, *msg_suffix=''*, *end=' ... '*, *skip_updating_state=False*, *indent=None*, *msg_wrap_limit=None*, *return_info=False*, ***kwargs*)

Verify a file path before saving, creates directories, and manages console output.

Parameters

- **path** (*str | pathlib.Path | os.PathLike*) – Destination file path.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; `2` enables CWD boundary warnings; defaults to `False`.
- **msg_prefix** (*str*) – Text prefixed to the default print message; defaults to `""`.
- **state_verb** (*str*) – Verb indicating the action being performed, e.g. *saving* or *updating*; defaults to `"Saving"`.
- **state_prep** (*str*) – Preposition associated with `state_verb`; defaults to `"to"`.
- **msg_suffix** (*str*) – Text suffixed to the default print message; defaults to `""`.
- **end** (*str*) – String passed to the `end` parameter of `print`; defaults to `" ... "`.
- **skip_updating_state** (*bool*) – If `True`, `state_verb` does not flip to `"Updating"`; defaults to `False`.
- **indent** (*int | str | None*) – Indentation level; defaults to `None`.
- **msg_wrap_limit** (*int | None*) – Threshold for multi-line wrapping; If the string exceeds this value (e.g. `100`), it will be split at (before) `state_prep` to improve readability when being printed. If `None` (default), the printed string is in a single line.
- **return_info** (*bool*) – Whether to return file path information; defaults to `False`.

Returns

A tuple containing the absolute path, the relative directory path, and the extension.

Return type

tuple

Tests:

```
>>> from pyhelpers.store import _check_saving_path
>>> from pyhelpers.dirs import cd
>>> path = cd()
>>> _check_saving_path(path, verbose=True)
Traceback (most recent call last):
  ...
    raise ValueError(f"The input '{path}' appears to be a directory, not a file path.")
```

(continues on next page)

(continued from previous page)

```
ValueError: The input '<path>' appears to be a directory, not a file path.
>>> path = "pyhelpers.txt"
>>> _check_saving_path(path, verbose=True); print("Passed.")
Saving "pyhelpers.txt" ... Passed.
>>> path = cd("tests", "documents", "pyhelpers.txt")
>>> _check_saving_path(path, verbose=True); print("Passed.")
Saving "pyhelpers.txt" to "./tests/documents/" ... Passed.
>>> _check_saving_path(path, verbose=True, msg_wrap_limit=40); print("Passed.")
Saving "pyhelpers.txt" ...
  to "./tests/documents/" ... Passed.
>>> path = "C:\Windows\pyhelpers.txt"
>>> _check_saving_path(path, verbose=2); print("Passed.")
Saving "pyhelpers.txt" to "C:/Windows/" ... Passed.
  Warning: "C:/Windows/pyhelpers.txt" is outside the current working directory.
>>> path = "C:\pyhelpers.txt"
>>> _check_saving_path(path, verbose=2); print("Passed.")
Saving "pyhelpers.txt" to "C:/" ... Passed.
  Warning: "C:/pyhelpers.txt" is outside the current working directory.
>>> _check_saving_path(path, verbose=2, indent=4, msg_wrap_limit=30); print("Passed.")
    Saving "pyhelpers.txt" to "C:/" ... Passed.
      Warning: "C:/pyhelpers.txt" is outside the current working directory.
```

4.4.2 _autofit_column_width

pyhelpers.store.**_autofit_column_width**(*excel_writer*, *writer_kwargs*, *sheet_name*)

Adjust the column widths in an Excel spreadsheet based on the content length.

This function is specifically designed for *openpyxl* engine when working with pandas.ExcelWriter. It iterates through each column of the specified sheet and calculates the maximum length of the content. It then adjusts the column width to accommodate the longest content plus some padding.

Parameters

- **excel_writer** (*pandas.ExcelWriter*) – pandas.ExcelWriter object used to write data into Excel file.
- **writer_kwargs** (*dict*) – A dict of keyword arguments passed to the pandas.ExcelWriter, including the `'engine'`.
- **sheet_name** (*str*) – The name of the sheet to adjust.

Note

- This function assumes that *openpyxl* engine is used (i.e. `writer_kwargs['engine'] == 'openpyxl'`).
- It modifies the column dimensions directly in the pandas.ExcelWriter object.

See also

- For more information on *openpyxl*, refer to the official documentation: https://openpyxl.readthedocs.io/en/stable/

- Reference: [STORE-U-ACW-1]

4.4.3 _check_loading_path

pyhelpers.store.**_check_loading_path**(*path*, *verbose=False*, *msg_prefix=''*, *state_verb='Loading'*, *msg_suffix=''*, *end=' ... '*, *indent=None*, *return_info=False*, ***kwargs*)

Verify a file path for loading and prints status to the console.

Parameters

- **path_to_file** (*str | bytes | pathlib.Path*) – Path to the target file.
- **verbose** (*bool | int*) – Whether to print status; defaults to `False`.
- **msg_prefix** (*str*) – Text prepended to the message; defaults to `""`.
- **state_verb** (*str*) – Action verb; defaults to `"Loading"`.
- **msg_suffix** (*str*) – Text appended to the message; defaults to `""`.
- **end** (*str*) – Print end character; defaults to `" ... "`.
- **indent** (*int | str | None*) – Indentation level; defaults to `None`.
- **return_info** (*bool*) – If `True`, returns path metadata; defaults to `False`.

Returns

(Absolute path, Parent directory, Extension) if `return_info` else `None`.

Return type

tuple | None

Tests:

```
>>> from pyhelpers.store import _check_loading_path
>>> from pyhelpers.dirs import cd, get_relative_path
>>> from pathlib import Path

>>> path = cd("test_func.py")
>>> _check_loading_path(path, verbose=True); print("Passed.")
Loading "./test_func.py" ... Passed.

>>> file_path, rel_dir, file_ext = _check_loading_path(path, return_info=True)
>>> get_relative_path(file_path)
Path('test_func.py')
>>> get_relative_path(rel_dir)
Path('.')
>>> file_ext
'.py'

>>> path = Path("C:\Windows\pyhelpers.pkg")
>>> _check_loading_path(path, verbose=True); print("Passed.")
Loading "C:/Windows/pyhelpers.pkg" ... Passed.

>>> file_path, rel_dir, file_ext = _check_loading_path(path, return_info=True)
>>> get_relative_path(file_path)
Path('C:/Windows/pyhelpers.pkg')
```

(continues on next page)

(continued from previous page)

```
>>> get_relative_path(rel_dir)
Path('C:/Windows')
>>> file_ext
'.pkg'
```

4.4.4 _set_index

pyhelpers.store.**_set_index**(*data*, *index_col=None*)

Set the index of a dataframe using column names or integer positions.

Parameters

- **data** (*pandas.DataFrame*) – The dataframe to update.
- **index_col** (*str | int | list | None*) – Column name(s) or integer index/indices to set as the index. If `None` (default), the function sets the first column as the index only if its name is an empty string (`''`) or starts with `'Unnamed:'`.

Returns

The dataframe with the updated index.

Return type

pandas.DataFrame

Tests:

```
>>> from pyhelpers.store import _set_index
>>> from pyhelpers._cache import example_dataframe
>>> import numpy as np
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> example_df.equals(_set_index(example_df))
True
>>> example_df_1 = _set_index(example_df, index_col=0)
>>> example_df_1
           Latitude
Longitude
-0.127647  51.507322
-1.902691  52.479699
-2.245115  53.479489
-1.543794  53.797418
>>> example_df.iloc[:, 0].to_list() == example_df_1.index.to_list()
True
>>> example_df_2 = example_df.copy()
>>> example_df_2.index.name = ''
>>> example_df_2.reset_index(inplace=True)
>>> example_df_2 = _set_index(example_df_2, index_col=None)
>>> np.array_equal(example_df_2.values, example_df.values)
True
```

4.4.5 Saving data

1anti-flashwhitewhite

save_pickle(data, path_to_file[, verbose, ...])	Save data to a pickle file.
save_spreadsheet(data, path_to_file[, ...])	Save data to a spreadsheet file format (e.g. CSV, Microsoft Excel or OpenDocument).
save_spreadsheets(data, path_to_file, ...[, ...])	Save multiple dataframes to a multi-sheet Microsoft Excel (.xlsx, .xls) or OpenDocument (.ods) format file.
save_json(data, path_to_file[, engine, ...])	Save data to a JSON file.
save_joblib(data, path_to_file[, verbose, ...])	Save data to a Joblib file.
save_feather(data, path_to_file[, index, ...])	Save a dataframe to a Feather file.
save_svg_as_emf(path_to_svg, path_to_emf[, ...])	Save a SVG file (.svg) as a EMF file (.emf) using Inkscape.
save_fig(path_to_file[, dpi, ...])	Save a figure object to a file in a supported format.
save_figure(data, path_to_file[, ...])	Save a figure object to a file in a supported format (with the figure object specified).
save_html_as_pdf(data, path_to_file[, ...])	Save a web page as a PDF file using wkhtmltopdf.
save_data(data, path_to_file[, verbose, ...])	Save data to a file in a specific format.

save_pickle

pyhelpers.store.**save_pickle**(*data*, *path_to_file*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save data to a pickle file.

Parameters

- **data** (*Any*) – Data to be saved, compatible with the built-in pickle.dump() function.
- **path_to_file** (*str | os.PathLike*) – Path where the Pickle file will be saved.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to None.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for pickle.dump().

Examples:

```
>>> from pyhelpers.store import save_pickle
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
```

(continues on next page)

(continued from previous page)

```
>>> pickle_dat = 1
>>> pickle_pathname = cd("tests", "data", "dat.pickle")
>>> save_pickle(pickle_dat, pickle_pathname, verbose=True)
Saving "dat.pickle" to "./tests/data/" ... Done.
>>> # Get an example dataframe
>>> pickle_dat = example_dataframe()
>>> pickle_dat
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> save_pickle(pickle_dat, pickle_pathname, verbose=True)
Updating "dat.pickle" in "./tests/data/" ... Done.
```

Tip

- The file path is validated before saving. Ensure the directory exists and is writable.
- Supported compression formats: `.gz` (gzip), `.xz` (LZMA compression) and `.bz2` (bzip2).
- Other extensions are saved as uncompressed files.
- Compression format is determined by the file extension. Ensure the extension matches the desired format.

See also

- Examples for the function *load_pickle()*.

save_spreadsheet

pyhelpers.store.**save_spreadsheet**(*data*, *path_to_file*, *sheet_name='Sheet1'*, *index=False*, *engine=None*, *delimiter=','*, *autofit_column_width=True*, *writer_kwargs=None*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save data to a spreadsheet file format (e.g. CSV, Microsoft Excel or OpenDocument).

The file format is determined by the extension of `path_to_file`, which can be `".txt"`, `".csv"`, `".xlsx"` or `".xls"`. The saving engine may use xlsxwriter, openpyxl or odfpy.

Parameters

- **data** (*pandas.DataFrame*) – Data to be saved as a spreadsheet.
- **path_to_file** (*str | os.PathLike[str] | None*) – File path where the spreadsheet will be saved.
- **sheet_name** (*str*) – Name of the sheet where the data will be saved; defaults to `"Sheet1"`.

- **index** (*bool*) – Whether to include the dataframe index as a column; defaults to `False`.
- **engine** (*str | None*) – Engine to use for saving:
  - `'openpyxl'` or `'xlsxwriter'` for Microsoft Excel formats such as `.xlsx` or `.xls`.
  - `'odf'` for OpenDocument format `.ods`.
- **delimiter** (*str*) – Separator for `".csv"`, `".txt"` or `".odt"` file formats; defaults to `','`.
- **autofit_column_width** (*bool*) – Whether to autofit column width; defaults to `True`.
- **writer_kwargs** (*dict | None*) – [Optional] Additional parameters for the class pandas.ExcelWriter().
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to `None`.
- **kwargs** – [Optional] Additional parameters for the method pandas.DataFrame.to_excel() or pandas.DataFrame.to_csv().

Examples:

```
>>> from pyhelpers.store import save_spreadsheet
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> # Get an example dataframe
>>> spreadsheet_dat = example_dataframe()
>>> spreadsheet_dat
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> spreadsheet_pathname = cd("tests", "data", "dat.csv")
>>> save_spreadsheet(spreadsheet_dat, spreadsheet_pathname, index=True, verbose=True)
Saving "dat.csv" to "./tests/data/" ... Done.
>>> spreadsheet_pathname = cd("tests", "data", "dat.xlsx")
>>> save_spreadsheet(spreadsheet_dat, spreadsheet_pathname, index=True, verbose=True)
Saving "dat.xlsx" to "./tests/data/" ... Done.
>>> spreadsheet_pathname = cd("tests", "data", "dat.ods")
>>> save_spreadsheet(spreadsheet_dat, spreadsheet_pathname, index=True, verbose=True)
Saving "dat.ods" to "./tests/data/" ... Done.
```

save_spreadsheets

pyhelpers.store.**save_spreadsheets**(*data*, *path_to_file*, *sheet_names*, *mode='w'*, *if_sheet_exists=None*, *autofit_column_width=True*, *writer_kwargs=None*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save multiple dataframes to a multi-sheet Microsoft Excel (.xlsx, .xls) or OpenDocument (.ods) format file.

The function wraps `pandas.ExcelWriter` and `pandas.DataFrame.to_excel`, adding features like error suppression, progress output, column autofit and interactive handling of existing sheets in append mode.

Parameters

- **data** (*list[pandas.DataFrame] | tuple[pandas.DataFrame] | iterable[pandas.DataFrame]*) – Sequence of pandas DataFrames to be saved as sheets in the workbook.
- **path_to_file** (*str | os.PathLike*) – File path where the spreadsheet will be saved. Must end with `.xlsx`, `.xls` or `.ods`.
- **sheet_names** (*list[str] | tuple[str] | iterable[str]*) – Names of all sheets in the workbook. Must match the length of *data*.
- **mode** (*str*) – Mode for writing to the spreadsheet file:
  - `'w'` (default): Write mode. Creates a new file or overwrites existing.
  - `'a'`: Append mode. Adds sheets to an existing file. **Note:** Not supported for OpenDocument (`.ods`) files; a write operation will be performed instead.
- **if_sheet_exists** (*None | str*) – Behavior when trying to write a sheet that already exists:
  - `None` (default): Prompts the user for action: `[pass]|new|replace`.
  - `'error'`: Raises a ValueError (pandas default).
  - `'new'`: Creates a new sheet with an incremented name (e.g. 'Sheet11').
  - `'replace'`: Overwrites the existing sheet.
  - `'overlay'`: Writes data on top of existing data (only available with `openpyxl`).
  - `'pass'`: Skips saving the sheet.
- **autofit_column_width** (*bool*) – Whether to adjust column width to fit content automatically; defaults to `True`. Requires the `openpyxl` or `odfpy` engine.
- **writer_kwargs** (*dict | None*) – [Optional] Additional parameters for the class pandas.ExcelWriter(), such as *date_format* or *datetime_format*; defaults to `None`.

- **verbose** (*bool | int*) – Whether to print relevant information and sheet saving progress to the console; defaults to `False`.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to `None`.
- **raise_error** (*bool*) – Whether to raise the exception if saving a specific sheet fails; if `raise_error=False` (default), the error will be suppressed, and the process will continue with the next sheet.
- **kwargs** – [Optional] Additional parameters for the method pandas.DataFrame.to_excel(), e.g. `index=False`, `header=True`.

Returns

`None`. The function's main effect is the side effect of saving the file.

Return type

None

Raises

- **AssertionError** – If *path_to_file* does not end with a supported file extension (`.xlsx`, `.xls`, `.ods`).
- **ValueError** – If an invalid option is provided for `if_sheet_exists` when not `None`.
- **Exception** – Any exception raised during saving if `raise_error=True`.

Examples:

```
>>> from pyhelpers.store import save_spreadsheets
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> data1 = example_dataframe()  # Get an example dataframe
>>> data1
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> data2 = data1.T
>>> data2
City          London  Birmingham  Manchester      Leeds
Longitude  -0.127647   -1.902691   -2.245115  -1.543794
Latitude   51.507322   52.479699   53.479489  53.797418
>>> data = [data1, data2]
>>> sheet_names = ['TestSheet1', 'TestSheet2']
>>> # Save to ODS format (write mode)
>>> path_to_file = cd("tests", "data", "dat.ods")
>>> save_spreadsheets(
...     data, path_to_file=path_to_file, sheet_names=sheet_names, verbose=True)
Saving "dat.ods" to "./tests/data/" ...
  'TestSheet1' ... Done.
  'TestSheet2' ... Done.
>>> # Save to XLSX format (append mode with interactive prompt)
>>> path_to_file = cd("tests", "data", "dat.xlsx")
```

(continues on next page)

(continued from previous page)

```
>>> save_spreadsheets(
...     data, path_to_file=path_to_file, sheet_names=sheet_names, verbose=True)
Saving "dat.xlsx" to "./tests/data/" ...
  'TestSheet1' ... Done.
  'TestSheet2' ... Done.
>>> save_spreadsheets(
...     data, path_to_file=path_to_file, sheet_names=sheet_names, mode='a', verbose=True)
Updating "dat.xlsx" at "./tests/data/" ...
  'TestSheet1' ... This sheet already exists; [pass]|new|replace: new
    saved as 'TestSheet11' ... Done.
  'TestSheet2' ... This sheet already exists; [pass]|new|replace: new
    saved as 'TestSheet21' ... Done.
>>> # Save with automatic replacement
>>> save_spreadsheets(
...     data, path_to_file=path_to_file, sheet_names=sheet_names, mode='a',
...     if_sheet_exists='replace', verbose=True)
Updating "dat.xlsx" at "./tests/data/" ...
  'TestSheet1' ... Done.
  'TestSheet2' ... Done.
>>> save_spreadsheets(
...     data, path_to_file=path_to_file, sheet_names=sheet_names, mode='a',
...     if_sheet_exists='new', verbose=True)
Updating "dat.xlsx" at "./tests/data/" ...
  'TestSheet1' ... saved as 'TestSheet12' ... Done.
  'TestSheet2' ... saved as 'TestSheet22' ... Done.
```

save_json

pyhelpers.store.**save_json**(*data*, *path_to_file*, *engine=None*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save data to a JSON file.

Parameters

- **data** (*Any*) – Data to be serialized and saved as a JSON file.
- **path_to_file** (*str | os.PathLike*) – File path where the JSON file will be saved.
- **engine** (*str | None*) – Serialisation engine:
  - None (default): Use the built-in json module.
  - 'ujson': Use UltraJSON for faster serialization.
  - 'orjson': Use orjson for faster and more efficient serialization.
  - 'rapidjson': Use python-rapidjson for fast and efficient serialization.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to None.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

- **kwargs** – [Optional] Additional parameters for one of the following functions:
  - json.dump() (if engine=None)
  - orjson.dumps() (if engine='orjson')
  - ujson.dump() (if engine='ujson')
  - rapidjson.dump() (if engine='rapidjson')

Examples:

```
>>> from pyhelpers.store import save_json
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> import json
>>> json_pathname = cd("tests", "data", "dat.json")
>>> json_dat = {'a': 1, 'b': 2, 'c': 3, 'd': ['a', 'b', 'c']}
>>> save_json(json_dat, json_pathname, indent=4, verbose=True)
Saving "dat.json" to "./tests/data/" ... Done.
>>> # Get an example dataframe
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> # Convert the dataframe to JSON format
>>> json_dat = json.loads(example_df.to_json(orient='index'))
>>> json_dat
{'London': {'Longitude': -0.1276474, 'Latitude': 51.5073219},
 'Birmingham': {'Longitude': -1.9026911, 'Latitude': 52.4796992},
 'Manchester': {'Longitude': -2.2451148, 'Latitude': 53.4794892},
 'Leeds': {'Longitude': -1.5437941, 'Latitude': 53.7974185}}
>>> # Use built-in json module
>>> save_json(json_dat, json_pathname, indent=4, verbose=True)
Updating "dat.json" in "./tests/data/" ... Done.
>>> save_json(json_dat, json_pathname, engine='orjson', verbose=True)
Updating "dat.json" in "./tests/data/" ... Done.
>>> save_json(json_dat, json_pathname, engine='ujson', indent=4, verbose=True)
Updating "dat.json" in "./tests/data/" ... Done.
>>> save_json(json_dat, json_pathname, engine='rapidjson', indent=4, verbose=True)
Updating "dat.json" in "./tests/data/" ... Done.
```

See also

- Examples for the function *load_json()*.

save_joblib

pyhelpers.store.**save_joblib**(*data*, *path_to_file*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save data to a Joblib file.

Parameters

- **data** (*Any*) – The data to be serialized and saved using joblib.dump().
- **path_to_file** (*str | os.PathLike*) – The file path where the Joblib file will be saved.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to None.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the joblib.dump() function.

Examples:

```
>>> from pyhelpers.store import save_joblib
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> joblib_pathname = cd("tests", "data", "dat.joblib")
>>> # Example 1:
>>> joblib_dat = example_dataframe().to_numpy()
>>> joblib_dat
array([[-0.1276474, 51.5073219],
       [-1.9026911, 52.4796992],
       [-2.2451148, 53.4794892],
       [-1.5437941, 53.7974185]])
>>> save_joblib(joblib_dat, joblib_pathname, verbose=True)
Saving "dat.joblib" to "./tests/data/" ... Done.
>>> # Example 2:
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> np.random.seed(0)
>>> x = example_dataframe().to_numpy()
>>> y = np.random.rand(*x.shape)
>>> reg = LinearRegression().fit(x, y)
>>> reg
LinearRegression()
>>> save_joblib(reg, joblib_pathname, verbose=True)
Updating "dat.joblib" in "./tests/data/" ... Done.
```

See also

- Examples for the function *pyhelpers.store.load_joblib()*.

save_feather

pyhelpers.store.**save_feather**(*data*, *path_to_file*, *index=True*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save a dataframe to a Feather file.

Note

The Feather format may require the index to be the default integer index. If the index is not a standard range, or if `index=True`, it will be automatically reset and saved as a column.

Parameters

- **data** (*pandas.DataFrame*) – The dataframe to be saved.
- **path_to_file** (*str | pathlib.Path*) – The path where the Feather file will be saved.
- **index** (*bool | None*) – Whether to include the index as a column. If `None`, the index is included only if it is not the default range; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to `None`.
- **raise_error** (*bool*) – Whether to raise an exception if saving fails; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for pandas.DataFrame.to_feather().

Examples:

```
>>> from pyhelpers.store import save_feather
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> feather_dat = example_dataframe()  # Get an example dataframe
>>> feather_dat
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> feather_pathname = cd("tests/data", "dat.feather")
>>> save_feather(feather_dat, feather_pathname, verbose=True)
Saving "dat.feather" to "./tests/data/" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> save_feather(feather_dat, feather_pathname, index=True, verbose=True)
Updating "dat.feather" in "./tests/data/" ... Done.
```

See also

- Examples for the function *pyhelpers.store.load_feather()*.

save_svg_as_emf

pyhelpers.store.**save_svg_as_emf**(*path_to_svg*, *path_to_emf*, *inkscape_exe=None*, *verbose=False*, *print_kwargs=None*, *raise_error=False*)

Save a SVG file (.svg) as a EMF file (.emf) using Inkscape.

Parameters

- **path_to_svg** (*str | os.PathLike*) – The path to the source SVG file.
- **path_to_emf** (*str | os.PathLike*) – The path where the EMF file will be saved.
- **inkscape_exe** (*str | None*) – The path to the Inkscape executable. If `inkscape_exe=None` (default), uses the standard installation path.
- **verbose** (*bool | int*) – Whether to print progress to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise FileNotFoundError if the Inkscape executable is not found; defaults to `False`.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to `None`.

Examples:

```
>>> from pyhelpers.store import save_svg_as_emf
>>> from pyhelpers.dirs import cd
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(backend='TkAgg')
>>> import matplotlib.pyplot as plt
>>> x, y = (1, 1), (2, 2)
>>> fig = plt.figure()
>>> ax = fig.add_subplot()
>>> ax.plot([x[0], y[0]], [x[1], y[1]])
>>> # from pyhelpers.store import save_figure
>>> # save_figure(fig, "docs/source/_images/store-save_fig-demo.pdf", verbose=True)
>>> fig.show()
```

The above example is illustrated in Figure 9:

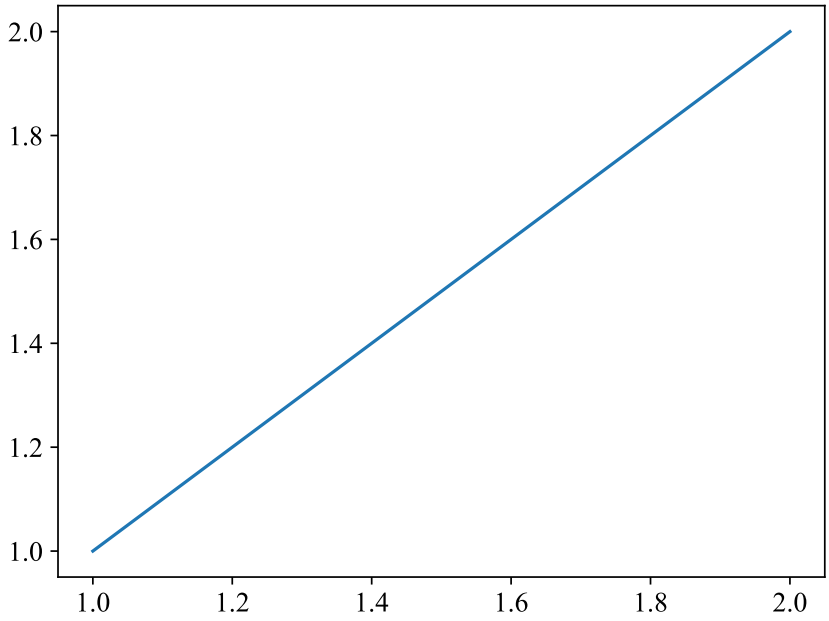

Figure 9: An example figure created for the function *save_svg_as_emf()*.

```
>>> img_dir = cd("tests", "images")
>>> path_to_svg = cd(img_dir, "store-save_fig-demo.svg")
>>> fig.savefig(path_to_svg)  # Save the figure as a .svg file
>>> path_to_emf = cd(img_dir, "store-save_fig-demo.emf")
>>> save_svg_as_emf(path_to_svg, path_to_emf, verbose=True)
Saving "store-save_fig-demo.emf" to "./tests/images/" ... Done.
>>> plt.close()
```

save_fig

pyhelpers.store.**save_fig**(*path_to_file*, *dpi=None*, *conv_svg_to_emf=False*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save a figure object to a file in a supported format.

This function utilizes the matplotlib.pyplot.savefig() function and optionally Inkscape for SVG to EMF conversion.

Parameters

- **path_to_file** (*str | os.PathLike*) – The path where the figure file will be saved.
- **dpi** (*int | None*) – Resolution in dots per inch; when `dpi=None` (default), it takes the value of `rcParams['savefig.dpi']`.
- **conv_svg_to_emf** (*bool*) – Whether to convert a .svg file to a .emf file; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to None.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters passed to matplotlib.pyplot.savefig().

Examples:

```
>>> from pyhelpers.store import save_fig
>>> from pyhelpers.dirs import cd
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(backend='TkAgg')
>>> import matplotlib.pyplot as plt
>>> x, y = (1, 1), (2, 2)
>>> fig = plt.figure()
>>> ax = fig.add_subplot()
>>> ax.plot([x[0], y[0]], [x[1], y[1]])
>>> fig.show()
```

The above example is illustrated in Figure 10:

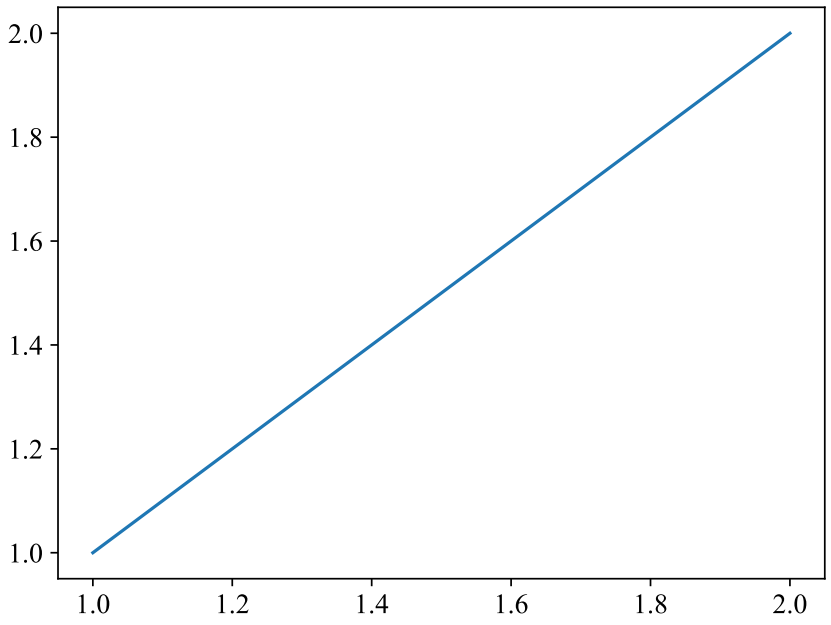

Figure 10: An example figure created for the function *save_fig()*.

```
>>> img_dir = cd("tests", "images")
>>> svg_file_pathname = cd(img_dir, "store-save_fig-demo.svg")
>>> save_fig(svg_file_pathname, verbose=True)
[Figure 1] Saving "store-save_fig-demo.png" in "./tests/images/" ... Done.
>>> save_fig(svg_file_pathname, verbose=True, conv_svg_to_emf=True)
```

(continues on next page)

(continued from previous page)

```
[Figure 1] Updating "store-save_fig-demo.svg" in "./tests/images/" ... Done.
[Figure 1] Saving "store-save_fig-demo.emf" to "./tests/images/" ... Done.
>>> plt.close()
```

save_figure

pyhelpers.store.**save_figure**(*data*, *path_to_file*, *conv_svg_to_emf=False*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save a figure object to a file in a supported format (with the figure object specified).

This function serves an alternative to the *save_fig()* function.

Parameters

- **data** (*matplotlib.Figure | seaborn.FacetGrid*) – The figure object to be saved.
- **path_to_file** (*str | os.PathLike*) – The path where the figure file will be saved.
- **conv_svg_to_emf** (*bool*) – Whether to convert a .svg file to a .emf file; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to None.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters passed to matplotlib.pyplot.savefig().

Examples:

```
>>> from pyhelpers.store import save_figure
>>> from pyhelpers.dirs import cd
>>> import numpy as np
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(backend='TkAgg')
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-5, 5)
>>> y = 1 / (1 + np.exp(-x))
>>> fig = plt.figure()
>>> ax = fig.add_subplot()
>>> ax.plot(x, y)
>>> fig.show()
```

The above example is illustrated in Figure 11:

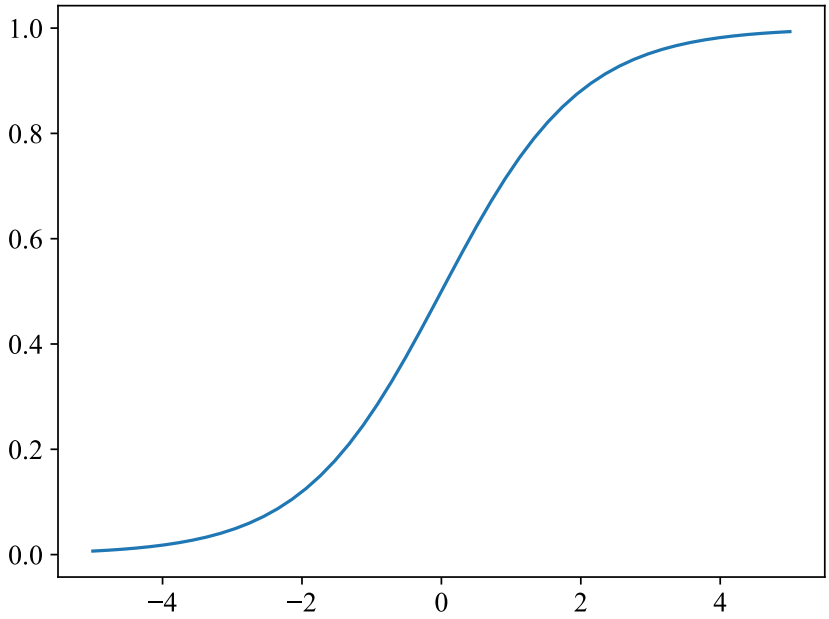

Figure 11: An example figure created for the function *save_figure()*.

```
>>> img_dir = cd("tests", "images")
>>> svg_file_pathname = cd(img_dir, "store-save_figure-demo.svg")
>>> save_figure(fig, svg_file_pathname, verbose=True)
[Figure 1] Saving "store-save_figure-demo.png" in "./tests/images/" ... Done.
>>> # save_figure(fig, "docs/source/_images/store-save_figure-demo.svg", verbose=True)
>>> # save_figure(fig, "docs/source/_images/store-save_figure-demo.pdf", verbose=True)
>>> save_figure(fig, svg_file_pathname, verbose=True, conv_svg_to_emf=True)
[Figure 1] Updating "store-save_figure-demo.svg" in "./tests/images/" ... Done.
[Figure 1] Saving "store-save_figure-demo.emf" to "./tests/images/" ... Done.
>>> plt.close()
```

save_html_as_pdf

pyhelpers.store.**save_html_as_pdf**(*data*, *path_to_file*, *if_exists='replace'*, *page_size='A4'*, *zoom=1.0*, *encoding='UTF-8'*, *wkhtmltopdf_options=None*, *wkhtmltopdf_path=None*, *verbose=False*, *print_kwargs=None*, *raise_error=False*, ***kwargs*)

Save a web page as a PDF file using wkhtmltopdf.

Parameters

- **data** (*str | os.PathLike*) – The URL of a web page or the pathname of an HTML file.
- **path_to_file** (*str | os.PathLike*) – The path where the PDF file will be saved.
- **if_exists** (*str*) – Action to take if the .pdf file already exists; options are `'replace'` (default) and `'pass'`.

- **page_size** (*str*) – The page size; defaults to `'A4'`.
- **zoom** (*float*) – Magnification for zooming in/out; defaults to `1.0`.
- **encoding** (*str*) – The encoding format; defaults to `'UTF-8'`.
- **wkhtmltopdf_options** (*dict | None*) – Options for wkhtmltopdf; defaults to `None`. Refer to the description of pdfkit project for more details.
- **wkhtmltopdf_path** (*str | None*) – The path to "*wkhtmltopdf.exe*"; when `wkhtmltopdf_path=None` (default), the default installation path will be used, e.g. "*C:/Program Files/wkhtmltopdf/bin/wkhtmltopdf.exe*" (on Windows).
- **wkhtmltopdf_path** – The path to the wkhtmltopdf executable; if `None` (default), searches standard installation paths.
- **verbose** (*bool | int*) – Whether to print progress to the console; defaults to `False`. Set `verbose=2` to see full output from wkhtmltopdf.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to `None`.
- **raise_error** (*bool*) – Whether to raise exceptions on failure; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for pdfkit.from_url() or pdfkit.from_file().

Examples:

```
>>> from pyhelpers.store import save_html_as_pdf
>>> from pyhelpers.dirs import cd
>>> import subprocess
>>> pdf_pathname = cd("tests", "documents", "pyhelpers.pdf")
>>> web_page_url = 'https://pyhelpers.readthedocs.io/en/latest/'
>>> save_html_as_pdf(web_page_url, pdf_pathname)
>>> subprocess.Popen(pdf_pathname, shell=True)  # Open the PDF file
>>> # Close the PDF file (if opened with Foxit Reader)
>>> # subprocess.call("taskkill /f /im FoxitPDFReader.exe", shell=True)
>>> wkhtmltopdf_options = {'margin-top': '0', 'orientation': 'Landscape'}
>>> # Using custom options for margins and orientation
>>> save_html_as_pdf(
...     web_page_url, pdf_pathname, wkhtmltopdf_options=wkhtmltopdf_options, verbose=True)
>>> subprocess.Popen(pdf_pathname, shell=True)
>>> # subprocess.call("taskkill /f /im FoxitPDFReader.exe", shell=True)
>>> web_page_file = cd("docs", "build", "html", "index.html")
>>> save_html_as_pdf(web_page_file, pdf_pathname, verbose=2)
Updating "pyhelpers.pdf" in "./tests/documents/" ...
Loading pages (1/6)
Counting pages (2/6)
Resolving links (4/6)
Loading headers and footers (5/6)
Printing pages (6/6)
Done
>>> subprocess.Popen(pdf_pathname, shell=True)
>>> # subprocess.call("taskkill /f /im FoxitPDFReader.exe", shell=True)
>>> save_html_as_pdf(web_page_file, pdf_pathname, verbose=True)
Updating "pyhelpers.pdf" in "./tests/documents/" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> subprocess.Popen(pdf_pathname, shell=True)
>>> # subprocess.call("taskkill /f /im FoxitPDFReader.exe", shell=True)
```

save_data

pyhelpers.store.**save_data**(*data*, *path_to_file*, *verbose=False*, *print_kwargs=None*, *show_warning=True*, *raise_error=False*, ***kwargs*)

Save data to a file in a specific format.

Parameters

- **data** (*Any*) – The data to be saved, which can be:
  - a file in Pickle, CSV, Microsoft Excel, JSON, Joblib, Feather or Parquet format;
  - a URL of a web page or an HTML file;
  - an image file in a Matplotlib-supported format.
- **path_to_file** (*str | os.PathLike*) – The path of the file where the `data` will be stored.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **print_kwargs** (*dict | None*) – [Optional] Additional parameters passed to *pyhelpers.store._check_saving_path()*. Defaults to `None`.
- **show_warning** (*bool*) – Whether to display a warning message if an unknown error occurs; defaults to `True`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for one of the following functions: *save_pickle()*, *save_spreadsheet()*, *save_spreadsheets()*, *save_json()*, *save_joblib()*, *save_feather()*, `save_parquet()`, `save_geopackage()`, *save_figure()* or `save_web_page_as_pdf()`.

Examples:

```
>>> from pyhelpers.store import save_data
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> data_dir = cd("tests", "data")
>>> # Get an example dataframe
>>> data = example_dataframe()
>>> data
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> # Save the data to files different formats:
```

(continues on next page)

(continued from previous page)

```
>>> path_to_file = cd(data_dir, "dat.pickle")
>>> save_data(data, path_to_file, verbose=True)
Saving "dat.pickle" in "./tests/data/" ... Done.
>>> path_to_file = cd(data_dir, "dat.csv")
>>> save_data(data, path_to_file, verbose=True, index=True)
Saving "dat.csv" in "./tests/data/" ... Done.
>>> path_to_file = cd(data_dir, "dat.xlsx")
>>> save_data(data, path_to_file, verbose=True, index=True)
Saving "dat.xlsx" in "./tests/data/" ... Done.
>>> path_to_file = cd(data_dir, "dat.txt")
>>> save_data(data, path_to_file, verbose=True, index=True)
Saving "dat.txt" in "./tests/data/" ... Done.
>>> path_to_file = cd(data_dir, "dat.feather")
>>> save_data(data, path_to_file, verbose=True, index=True)
Saving "dat.feather" in "./tests/data/" ... Done.
>>> path_to_file = cd(data_dir, "dat.parquet")
>>> save_data(data, path_to_file, verbose=True)
Saving "dat.parquet" in "./tests/data/" ... Done.
>>> data = data.T.to_dict()  # Convert `dat` to JSON format
>>> path_to_file = cd(data_dir, "dat.json")
>>> save_data(data, path_to_file, verbose=True, indent=4)
Saving "dat.json" to "./tests/data/" ... Done.
```

See also

- Examples for *load_data()*.

4.4.6 Loading data

1anti-flashwhitewhite

load_pickle(path_to_file[, verbose, ...])	Load data from a Pickle file.
load_csv(path_to_file[, delimiter, header, ...])	Load data from a CSV file.
load_spreadsheets(path_to_file[, as_dict, ...])	Load one or multiple sheets from a Microsoft Excel or an OpenDocument format file.
load_json(path_to_file[, engine, verbose, ...])	Load data from a JSON file.
load_joblib(path_to_file[, verbose, ...])	Load data from a Joblib file.
load_feather(path_to_file[, index_col, ...])	Load a dataframe from a Feather file.
load_csr_matrix(path_to_file[, verbose, ...])	Load in a compressed sparse row (CSR) or compressed row storage (CRS).
load_data(path_to_file[, verbose, ...])	Load data from a file.

load_pickle

pyhelpers.store.**load_pickle**(*path_to_file*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load data from a Pickle file.

The function is intended for use with trusted data sources only.

Parameters

- **path_to_file** (*str | os.PathLike*) – Path where the pickle file is saved.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for *pyhelpers.store._check_loading_path()*; defaults to `None`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the function pickle.load().

Returns

Data retrieved from the specified path `path_to_file`.

Return type

Any

Note

- Ensure that `path_to_file` comes from a trusted source to avoid deserialization vulnerabilities.
- Example data can be referred to the function `save_pickle()`.

Examples:

```
>>> from pyhelpers.store import load_pickle
>>> from pyhelpers.dirs import cd

>>> pickle_pathname = cd("tests", "data", "dat.pickle")
>>> pickle_dat = load_pickle(pickle_pathname, verbose=True)
Loading "tests/data/dat.pickle" ... Done.

>>> pickle_dat
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
```

load_csv

pyhelpers.store.**load_csv**(*path_to_file*, *delimiter=','*, *header=0*, *index_col=None*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, *encoding='utf-8'*, ***kwargs*)

Load data from a CSV file.

A single header row (or none) read from an actual file on disk is parsed manually via csv.reader() for efficiency; a multi-row `header` or an in-memory file-like object (e.g. `io.StringIO`) is instead passed directly to pandas.read_csv(), which supports both natively. Which backend is used determines which of `kwargs` are accepted, since csv.reader() and pandas.read_csv() recognize different keyword arguments.

Parameters

- **path_to_file** (*str | os.PathLike | io.StringIO*) – Pathname of the CSV file, or an in-memory file-like object (e.g. `io.StringIO`).
- **delimiter** (*str*) – Delimiter used between values in the data file; defaults to `','`.
- **header** (*int | List[int] | None*) – Index number of the row(s) used as column names; a single integer (or `None`, for no header) is handled via csv.reader(), while a list of integers is handled via pandas.read_csv()'s native multi-row header support. Defaults to `0`.
- **index_col** (*str | int | list | None*) – Index number of the column(s) to use as the row labels of the dataframe; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for the function *_check_loading_path()*; defaults to `None`.
- **raise_error** (*bool*) – Whether to raise an exception if the data fails to load; if `raise_error=False` (default), the error is suppressed instead.
- **encoding** (*str*) – Character encoding used to read `path_to_file`; defaults to `'utf-8'`.
- **kwargs** – [Optional] Additional parameters for csv.reader() or pandas.read_csv(), depending on which backend is used (see above).

Returns

Data retrieved from the specified path `path_to_file`.

Return type

pandas.DataFrame | None

Note

Example data can be referred to in *save_spreadsheet()*.

Examples:

```
>>> from pyhelpers.store import load_csv
>>> from pyhelpers.dirs import cd

>>> csv_pathname = cd("tests", "data", "dat.csv")
>>> csv_dat = load_csv(csv_pathname, index_col=0, verbose=True)
Loading "tests/data/dat.csv" ... Done.
>>> csv_dat
            Longitude   Latitude
City
London      -0.1276474  51.5073219
Birmingham  -1.9026911  52.4796992
Manchester  -2.2451148  53.4794892
Leeds       -1.5437941  53.7974185
```

(continues on next page)

(continued from previous page)

```
>>> csv_pathname = cd("tests", "data", "dat.txt")
>>> csv_dat = load_csv(csv_pathname, index_col=0, verbose=True)
Loading "tests/data/dat.txt" ... Done.
>>> csv_dat
             Longitude    Latitude
City
London      -0.1276474  51.5073219
Birmingham  -1.9026911  52.4796992
Manchester  -2.2451148  53.4794892
Leeds       -1.5437941  53.7974185

>>> csv_dat = load_csv(csv_pathname, header=[0, 1], verbose=True)
Loading "tests/data/dat.txt" ... Done.
>>> csv_dat
         City Easting Northing
       London  530034   180381
0  Birmingham  406689   286822
1  Manchester  383819   398052
2       Leeds  582044   152953
```

load_spreadsheets

pyhelpers.store.**load_spreadsheets**(*path_to_file*, *as_dict=True*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load one or multiple sheets from a Microsoft Excel or an OpenDocument format file.

Parameters

- **path_to_file** (*str | os.PathLike*) – Path where the spreadsheet file is saved.
- **as_dict** (*bool*) – Whether to return the retrieved data as a dictionary; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for the function *pyhelpers.store._check_loading_path()*; defaults to `None`.
- **kwargs** – [Optional] Additional parameters for the method pandas.ExcelFile.parse().

Returns

Data of all worksheets in the file from the specified pathname `path_to_file`.

Return type

list | dict | None

Note

- Example data can be referred to in the functions `save_multiple_spreadsheets()` and *`save_spreadsheet()`*.

Examples:

```
>>> from pyhelpers.store import load_spreadsheets
>>> from pyhelpers.dirs import cd

>>> dat_dir = cd("tests", "data")
>>> path_to_xlsx = cd(dat_dir, "dat.ods")

>>> wb_data = load_spreadsheets(path_to_xlsx, verbose=True, index_col=0)
Loading "tests/data/dat.ods" ...
  'TestSheet1'. ... Done.
  'TestSheet2'. ... Done.

>>> list(wb_data.keys())
['TestSheet1', 'TestSheet2']

>>> wb_data['TestSheet1']
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418

>>> path_to_xlsx = cd(dat_dir, "dat.xlsx")
>>> wb_data = load_spreadsheets(path_to_xlsx, verbose=True, index_col=0)
Loading "tests/data/dat.xlsx" ...
  'TestSheet1'. ... Done.
  'TestSheet2'. ... Done.
  'TestSheet11'. ... Done.
  'TestSheet21'. ... Done.
  'TestSheet12'. ... Done.
  'TestSheet22'. ... Done.

>>> list(wb_data.keys())
['TestSheet1',
 'TestSheet2',
 'TestSheet11',
 'TestSheet21',
 'TestSheet12',
 'TestSheet22']

>>> wb_data = load_spreadsheets(path_to_xlsx, as_dict=False, index_col=0)

>>> type(wb_data)
list

>>> len(wb_data)
6

>>> wb_data[0]
```

(continues on next page)

(continued from previous page)

```
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
```

load_json

pyhelpers.store.**load_json**(*path_to_file*, *engine=None*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load data from a JSON file.

Parameters

- **path_to_file** (*str | os.PathLike*) – Path where the JSON file is saved.
- **engine** (*str | None*) – An open-source Python package for JSON serialization; valid options include `None` (default, for the built-in json module), `'ujson'` (for UltraJSON), `'orjson'` (for orjson) and `'rapidjson'` (for python-rapidjson); defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for the function *pyhelpers.store._check_loading_path()*; defaults to `None`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for json.load() (if `engine=None`), orjson.loads() (if `engine='orjson'`), ujson.load() (if `engine='ujson'`) or rapidjson.load() (if `engine='rapidjson'`).

Returns

Data retrieved from the specified path `path_to_file`.

Return type

dict

Note

- Example data can be referred to in the function *save_json()*.

Examples:

```
>>> from pyhelpers.store import load_json
>>> from pyhelpers.dirs import cd

>>> json_path = cd("tests", "data", "dat.json")
>>> json_dat = load_json(json_path, verbose=True)
Loading "tests/data/dat.json" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> json_dat
{'London': {'Longitude': -0.1276474, 'Latitude': 51.5073219},
 'Birmingham': {'Longitude': -1.9026911, 'Latitude': 52.4796992},
 'Manchester': {'Longitude': -2.2451148, 'Latitude': 53.4794892},
 'Leeds': {'Longitude': -1.5437941, 'Latitude': 53.7974185}}
```

load_joblib

pyhelpers.store.**load_joblib**(*path_to_file*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load data from a Joblib file.

Parameters

- **path_to_file** (*str | os.PathLike*) – Path where the Joblib file is saved.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to `False`.
- **prt_kwargs** (*dict | None*) – [Optional] additional parameters for the function *pyhelpers.store._check_loading_path()*; defaults to `None`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] additional parameters for the function joblib.load().

Returns

Data retrieved from the specified path `path_to_file`.

Return type

Any

Note

- Example data can be referred to in the function *save_joblib()*.

Examples:

```
>>> from pyhelpers.store import load_joblib
>>> from pyhelpers.dirs import cd

>>> joblib_pathname = cd("tests", "data", "dat.joblib")
>>> joblib_dat = load_joblib(joblib_pathname, verbose=True)
Loading "tests/data/dat.joblib" ... Done.

>>> joblib_dat
array([[0.5488135 , 0.71518937, 0.60276338, ..., 0.02010755, 0.82894003,
        0.00469548],
       [0.67781654, 0.27000797, 0.73519402, ..., 0.25435648, 0.05802916,
        0.43441663],
       [0.31179588, 0.69634349, 0.37775184, ..., 0.86219152, 0.97291949,
        0.96083466],
```

(continues on next page)

(continued from previous page)

```
       ...,
       [0.89111234, 0.26867428, 0.84028499, ..., 0.5736796 , 0.73729114,
        0.22519844],
       [0.26969792, 0.73882539, 0.80714479, ..., 0.94836806, 0.88130699,
        0.1419334 ],
       [0.88498232, 0.19701397, 0.56861333, ..., 0.75842952, 0.02378743,
        0.81357508]])
```

load_feather

pyhelpers.store.**load_feather**(*path_to_file*, *index_col=None*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load a dataframe from a Feather file.

Parameters

- **path_to_file** (*str | os.PathLike*) – Path where the feather file is saved.
- **index_col** (*str | int | list | None*) – Index number or name of the column(s) to use as the row labels of the dataframe; defaults to None.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to False.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for the function *pyhelpers.store._check_loading_path()*; defaults to None.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the function pandas.read_feather():
  - columns: Sequence of column names to read. If None, all columns are read.
  - use_threads: Whether to parallelize reading using multiple threads; defaults to True.

Returns

Data retrieved from the specified path path_to_file.

Return type

pandas.DataFrame

Note

- Example data can be referred to in the function *save_feather()*.

Examples:

```
>>> from pyhelpers.store import load_feather
>>> from pyhelpers.dirs import cd

>>> feather_path = cd("tests", "data", "dat.feather")
```

(continues on next page)

(continued from previous page)

```
>>> feather_dat = load_feather(feather_path, index_col=0, verbose=True)
Loading "tests/data/dat.feather" ... Done.

>>> feather_dat
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
```

load_csr_matrix

pyhelpers.store.**load_csr_matrix**(*path_to_file*, *verbose=False*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load in a compressed sparse row (CSR) or compressed row storage (CRS).

Parameters

- **path_to_file** (*str | os.PathLike*) – Path to the CSR file (e.g. with extension ".npz").
- **verbose** (*bool | int*) – Whether to print relevant information in console as the function runs; defaults to `False`.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for *pyhelpers.store._check_loading_path()*; defaults to `None`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the function numpy.load().

Returns

A compressed sparse row.

Return type

scipy.sparse.csr.csr_matrix

Examples:

```
>>> from pyhelpers.store import load_csr_matrix
>>> from pyhelpers.dirs import cd
>>> from scipy.sparse import csr_matrix
>>> data = [1, 2, 3, 4, 5, 6]
>>> indices = [0, 2, 2, 0, 1, 2]
>>> indptr = [0, 2, 3, 6]
>>> csr_mat = csr_matrix((data, indices, indptr), shape=(3, 3))
>>> csr_mat
<3x3 sparse matrix of type '<class 'numpy.int32'>'
    with 6 stored elements in Compressed Sparse Row format>
>>> path_to_csr_npz = cd("tests", "data", "csr_mat.npz")
>>> csr_mat_ = load_csr_matrix(path_to_csr_npz, verbose=True)
Loading "tests/data/csr_mat.npz" ... Done.
>>> # .nnz gets the count of explicitly-stored values (non-zeros)
>>> (csr_mat != csr_mat_).count_nonzero() == 0
np.True_
```

(continues on next page)

(continued from previous page)

```
>>> (csr_mat != csr_mat_).nnz == 0
True
```

load_data

pyhelpers.store.**load_data**(*path_to_file*, *verbose=False*, *show_warning=True*, *prt_kwargs=None*, *raise_error=False*, ***kwargs*)

Load data from a file.

Parameters

- **path_to_file** (*str | os.PathLike*) – Pathname of the file; supported formats include Pickle, CSV, Microsoft Excel spreadsheet, JSON, Joblib, Feather, Parquet and GeoPackage.
- **verbose** (*bool | int*) – Whether to print relevant information in console as the function runs; defaults to `False`.
- **show_warning** (*bool*) – Whether to show a warning message if an unknown error occurs; defaults to `True`.
- **prt_kwargs** (*dict | None*) – [Optional] Additional parameters for the function *pyhelpers.store._check_loading_path()*; defaults to `None`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for one of the following functions: *load_pickle()*, *load_csv()*, *load_spreadsheets()*, *load_json()*, *load_joblib()*, *load_feather()*, `load_parquet()`, or `load_geopackage()`.

Returns

Data retrieved from the specified path `path_to_file`.

Return type

Any

Note

- Example data can be referred to in the function *save_data()*.

Examples:

```
>>> from pyhelpers.store import load_data
>>> from pyhelpers.dirs import cd
>>> data_dir = cd("tests", "data")
>>> dat_pathname = cd(data_dir, "dat.pickle")
>>> pickle_dat = load_data(path_to_file=dat_pathname, verbose=True)
Loading "tests/data/dat.pickle" ... Done.
>>> pickle_dat
           Longitude   Latitude
City
```

(continues on next page)

(continued from previous page)

```
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> dat_pathname = cd(data_dir, "dat.csv")
>>> csv_dat = load_data(path_to_file=dat_pathname, index_col=0, verbose=True)
Loading "tests/data/dat.csv" ... Done.
>>> csv_dat
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> dat_pathname = cd(data_dir, "dat.json")
>>> json_dat = load_data(path_to_file=dat_pathname, verbose=True)
Loading "tests/data/dat.json" ... Done.
>>> json_dat
{'London': {'Longitude': -0.1276474, 'Latitude': 51.5073219},
 'Birmingham': {'Longitude': -1.9026911, 'Latitude': 52.4796992},
 'Manchester': {'Longitude': -2.2451148, 'Latitude': 53.4794892},
 'Leeds': {'Longitude': -1.5437941, 'Latitude': 53.7974185}}
>>> dat_pathname = cd(data_dir, "dat.feather")
>>> feather_dat = load_data(path_to_file=dat_pathname, index_col=0, verbose=True)
Loading "tests/data/dat.feather" ... Done.
>>> feather_dat
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> dat_pathname = cd(data_dir, "dat.joblib")
>>> joblib_dat = load_data(path_to_file=dat_pathname, verbose=True)
Loading "tests/data/dat.joblib" ... Done.
>>> joblib_dat
LinearRegression()
```

4.4.7 Transforming data files

1anti-flashwhitewhite

unzip(path_to_zip_file[, output_dir, ...])	Unzip data from a Zip (compressed) file.
seven_zip(zip_file_path[, output_dir, mode, ...])	Extract data from a compressed file using 7-Zip.
markdown_to_rst(path_to_md, path_to_rst[, ...])	Convert a Markdown (.md) file to a reStructuredText (.rst) file.
xlsx_to_csv(path_to_xlsx[, path_to_csv, ...])	Convert a Microsoft Excel spreadsheet to a CSV file.

unzip

pyhelpers.store.**unzip**(*path_to_zip_file*, *output_dir=None*, *ret_output_dir=False*, *verbose=False*, *raise_error=False*, ***kwargs*)

Unzip data from a Zip (compressed) file.

Parameters

- **path_to_zip_file** (*str | pathlib.Path | os.PathLike*) – The path where the Zip file is saved.
- **output_dir** (*str | None*) – The directory where the extracted data will be saved; defaults to None.
- **ret_output_dir** (*bool*) – Whether to return the path to output directory; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the method zipfile.ZipFile.extractall().

Examples:

```
>>> from pyhelpers.store import unzip
>>> from pyhelpers.dirs import cd, delete_dir

>>> zip_file_path = cd("tests", "data", "zipped.zip")
>>> unzip(path_to_zip_file=zip_file_path, verbose=True)
Extracting "tests/data/zipped.zip" to "tests/data/zipped/" ... Done.

>>> output_dir_1 = cd("tests", "data", "zipped")
>>> out_file_pathname = cd(output_dir_1, "zipped.txt")
>>> with open(out_file_pathname) as f:
...     print(f.read())
test

>>> output_dir_2 = cd("tests", "data", "zipped_alt")
>>> unzip(path_to_zip_file=zip_file_path, output_dir=output_dir_2, verbose=True)
Extracting "tests/data/zipped.zip" to "tests/data/zipped_alt\" ... Done.
>>> out_file_pathname = cd(output_dir_2, "zipped.txt")
>>> with open(out_file_pathname) as f:
...     print(f.read())
test

>>> # Delete the directories "tests/data/zipped/" and "tests/data/zipped_alt/"
>>> delete_dir([output_dir_1, output_dir_2], verbose=True)
Confirm deletion of the following directories:
  "tests/data/zipped/" (Not empty)
  "tests/data/zipped_alt/" (Not empty)
? [No]|Yes: yes
Deleting:
  "tests/data/zipped/" ... Done.
  "tests/data/zipped_alt/" ... Done.
```

seven_zip

pyhelpers.store.**seven_zip**(*zip_file_path*, *output_dir=None*, *mode='aoa'*, *return_output_dir=False*, *verbose=False*, *raise_error=False*, *seven_zip_exe=None*)

Extract data from a compressed file using 7-Zip.

Parameters

- **zip_file_path** (*str | os.PathLike*) – The path where the compressed file is saved.
- **output_dir** (*str | None*) – The directory where the extracted data will be saved; defaults to None.
- **mode** (*str*) – The extraction mode; defaults to 'aoa'.
- **return_output_dir** (*bool*) – Whether to return the path to output directory; defaults to False.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **seven_zip_exe** (*str | None*) – The path to the executable "*7z.exe*"; If seven_zip_exe=None (default), the default installation path will be used, e.g. "*C:\Program Files\7-Zip\7z.exe*" (on Windows).

Examples:

```
>>> from pyhelpers.store import seven_zip
>>> from pyhelpers.dirs import cd, delete_dir

>>> zip_file_path = cd("tests", "data", "zipped.zip")
>>> seven_zip(zip_file_path=zip_file_path, verbose=True)

7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29

Scanning the drive for archives:
1 file, 158 bytes (1 KiB)

Extracting archive: .\tests\data\zipped.zip
--
Path = .\tests\data\zipped.zip
Type = zip
Physical Size = 158

Everything is Ok

Size:       4
Compressed: 158

Done.

>>> output_dir_1 = cd("tests", "data", "zipped")
>>> out_file_path = cd(output_dir_1, "zipped.txt")
>>> with open(out_file_path) as f:
```

(continues on next page)

(continued from previous page)

```
...     print(f.read())
test

>>> output_dir_2 = cd("tests", "data", "zipped_alt")
>>> seven_zip(zip_file_path=zip_file_path, output_dir=output_dir_2, verbose=True)
>>> out_file_pathname = cd("tests", "data", "zipped_alt", "zipped.txt")
>>> with open(out_file_pathname) as f:
...     print(f.read())
test

>>> # Extract a .7z file
>>> zip_file_path = cd("tests", "data", "zipped.7z")
>>> seven_zip(zip_file_path=zip_file_path, output_dir=output_dir_2)
>>> out_file_pathname = cd("tests", "data", "zipped", "zipped.txt")
>>> with open(out_file_pathname) as f:
...     print(f.read())
test

>>> # Delete the directories "tests/data/zipped/" and "tests/data/zipped_alt/"
>>> delete_dir([output_dir_1, output_dir_2], verbose=True)
Confirm deletion of the following directories:
  "tests/data/zipped/" (Not empty)
  "tests/data/zipped_alt/" (Not empty)
? [No]|Yes: yes
Deleting:
  "tests/data/zipped/" ... Done.
  "tests/data/zipped_alt/" ... Done.
```

markdown_to_rst

pyhelpers.store.**markdown_to_rst**(*path_to_md*, *path_to_rst*, *reverse=False*, *engine=None*, *pandoc_exe=None*, *verbose=False*, *raise_error=False*, ***kwargs*)

Convert a Markdown (.md) file to a reStructuredText (.rst) file.

This function relies on Pandoc or pypandoc, given the specified `engine`.

Parameters

- **path_to_md** (*str | os.PathLike*) – The path where the Markdown file is saved.
- **path_to_rst** (*str | os.PathLike*) – The path where the reStructuredText file will be saved.
- **reverse** (*bool*) – Specifies whether to convert an .rst file to a .md file; defaults to `False`.
- **engine** (*None | str*) – The engine/module used for performing the conversion; if `engine=None` (default), the function utilizes Pandoc, `'pypandoc'` otherwise.
- **pandoc_exe** (*str | None*) – The path to the executable "*pandoc.exe*"; If `pandoc_exe=None` (default), the default installation path will be used, e.g. "*C:\Program Files\Pandoc\pandoc.exe*" (on Windows).
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the function pypandoc.convert_file() (if `engine='pypandoc'`).

Examples:

```
>>> from pyhelpers.store import markdown_to_rst
>>> from pyhelpers.dirs import cd

>>> dat_dir = cd("tests", "documents")
>>> path_to_md_file = cd(dat_dir, "readme.md")
>>> path_to_rst_file = cd(dat_dir, "readme.rst")

>>> markdown_to_rst(path_to_md_file, path_to_rst_file, verbose=True)
Converting "tests/documents/readme.md" to "tests/documents/readme.rst" ... Done.

>>> markdown_to_rst(path_to_md_file, path_to_rst_file, engine='pypandoc', verbose=True)
Updating "readme.rst" in "tests/documents/" ... Done.
```

xlsx_to_csv

pyhelpers.store.**xlsx_to_csv**(*path_to_xlsx*, *path_to_csv=None*, *engine=None*, *if_exists='replace'*, *vbscript=None*, *sheet_name='1'*, *ret_null=False*, *verbose=False*, *raise_error=False*, ***kwargs*)

Convert a Microsoft Excel spreadsheet to a CSV file.

See also [STORE-XTC-1].

Parameters

- **path_to_xlsx** (*str | os.PathLike*) – The path of the Microsoft Excel spreadsheet (in *.xlsx* format).
- **path_to_csv** (*str | os.PathLike | None*) – The path of the CSV file:
  - When `path_to_csv=None` (default), a temporary file is generated using tempfile.NamedTemporaryFile().
  - When `path_to_csv=""`, the CSV file is generated in the same directory as the source Microsoft Excel spreadsheet.
  - Otherwise, it specifies a specific path.
- **engine** (*str | None*) – The engine used for converting *.xlsx*/*.xls* to .csv:
  - When `engine=None` (default), a VBScript (Visual Basic Script) is used.
  - When `engine='xlsx2csv'`, the function relies on xlsx2csv.
- **if_exists** (*str*) – The action to take if the target CSV file exists; defaults to `'replace'`.
- **vbscript** (*str | None*) – The path of the VBScript used for converting *.xlsx*/*.xls* to *.csv*; defaults to `None`.
- **sheet_name** (*str*) – The name of the target worksheet in the given Excel file; defaults to `'1'`.

- **ret_null** (*bool*) – Whether to return a value depending on the specified `engine`; defaults to `False`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the function xlsx2csv.Xlsx2csv() (if `engine='xlsx2csv'`).

Returns

The path of the generated CSV file or `None` when `engine=None`; an io.StringIO() buffer when `engine='xlsx2csv'`.

Return type

str | _io.StringIO | None

Examples:

```
>>> from pyhelpers.store import xlsx_to_csv, load_csv
>>> from pyhelpers.dirs import cd
>>> import os

>>> path_to_test_xlsx = cd("tests", "data", "dat.xlsx")
>>> path_to_temp_csv = xlsx_to_csv(path_to_test_xlsx, verbose=True)
Converting "tests/data/dat.xlsx" to a (temporary) CSV file ... Done.

>>> os.path.isfile(path_to_temp_csv)
True

>>> data = load_csv(path_to_temp_csv, index_col=0)
>>> data
             Longitude    Latitude
City
London       -0.1276474  51.5073219
Birmingham   -1.9026911  52.4796992
Manchester   -2.2451148  53.4794892
Leeds        -1.5437941  53.7974185

>>> # Set `engine='xlsx2csv'`
>>> temp_csv_buffer = xlsx_to_csv(path_to_test_xlsx, engine='xlsx2csv', verbose=True)
Converting "tests/data/dat.xlsx" to a (temporary) CSV file ... Done.

>>> # import pandas as pd; data_ = pandas.read_csv(io_buffer, index_col=0)
>>> data_ = load_csv(temp_csv_buffer, index_col=0)
>>> data_
             Longitude   Latitude
City
London       -0.127647  51.507322
Birmingham   -1.902691  52.479699
Manchester   -2.245115  53.479489
Leeds        -1.543794  53.797418

>>> data.astype('float16').equals(data_.astype('float16'))
True
```

(continues on next page)

(continued from previous page)

```
>>> # Remove the temporary CSV file
>>> os.remove(path_to_temp_csv)
```

4.5 geom

Manipulation of geometric/geographical data.

4.5.1 Distance-related calculations

1anti-flashwhitewhite

calc_spherical_distance(pt1, pt2[, unit, ...])	Calculates the distance between two points on a unit sphere.
calc_hypotenuse_distance(pt1, pt2)	Calculates the hypotenuse distance between two points.
find_closest_point(pt, ref_pts[, as_geom])	Finds the closest point in a sequence of reference points to a given point.
find_closest_points(pts, ref_pts[, k, ...])	Finds the closest points from a list of reference points to a set of query points.
find_shortest_path(points_sequence[, ...])	Finds the shortest path through a sequence of points.

calc_spherical_distance

pyhelpers.geom.**calc_spherical_distance**(*pt1*, *pt2*, *unit='mile'*, *decimals=None*)

Calculates the distance between two points on a unit sphere.

This function computes the spherical distance between two points `pt1` and `pt2`. The distance can be returned in either miles (`'mile'`) or kilometers (`'km'`) based on the `unit` parameter.

Parameters

- **pt1** (*shapely.geometry.Point | tuple | numpy.ndarray*) – One point.
- **pt2** (*shapely.geometry.Point | tuple | numpy.ndarray*) – Another point.
- **unit** (*str*) – Unit of distance for output; options include `'mile'` (default) and `'km'`.
- **decimals** (*int | None*) – Number of decimal places for the calculated result; defaults to `None` (no rounding).

Returns

Distance between `pt1` and `pt2` in miles or kilometers (relative to the earth's radius).

Return type

float | None

Examples:

```
>>> from pyhelpers.geom import calc_spherical_distance
>>> from pyhelpers._cache import example_dataframe
>>> example_df = example_dataframe()
>>> example_df
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> london, birmingham = example_df.loc[['London', 'Birmingham']].values
>>> london
array([-0.1276474, 51.5073219])
>>> birmingham
array([-1.9026911, 52.4796992])
>>> arc_len_in_miles = calc_spherical_distance(london, birmingham)
>>> arc_len_in_miles  # in miles
101.10431101941569
>>> arc_len_in_miles = calc_spherical_distance(london, birmingham, decimals=4)
>>> arc_len_in_miles
101.1043
```

Note

This function is modified from the original code available at [GEOM-CDOUS-1]. It assumes the earth is perfectly spherical and returns the distance based on each point's longitude and latitude.

calc_hypotenuse_distance

pyhelpers.geom.**calc_hypotenuse_distance**(*pt1*, *pt2*)

Calculates the hypotenuse distance between two points.

See also [GEOM-CHD-1].

Parameters

- **pt1** (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – One point.
- **pt2** (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – Another point.

Returns

Hypotenuse distance between `pt1` and `pt2`.

Return type

float

Note

- The hypotenuse distance is the straight-line distance between *pt1* and *pt2*.
- Calculated using the formula: `sqrt((x2 - x1)^2 + (y2 - y1)^2)`

- Equivalent to `numpy.hypot(x, y)` which computes `sqrt(x*x + y*y)`.

Examples:

```
>>> from pyhelpers.geom import calc_hypotenuse_distance
>>> from shapely.geometry import Point
>>> pt_1, pt_2 = (1.5429, 52.6347), (1.4909, 52.6271)
>>> hypot_distance = calc_hypotenuse_distance(pt_1, pt_2)
>>> hypot_distance
0.05255244999046248
>>> pt_1_, pt_2_ = map(lambda p: Point(p), (pt_1, pt_2))
>>> pt_1_.wkt
'POINT (1.5429 52.6347)'
>>> pt_2_.wkt
'POINT (1.4909 52.6271)'
>>> hypot_distance = calc_hypotenuse_distance(pt_1_, pt_2_)
>>> hypot_distance
0.05255244999046248
```

find_closest_point

pyhelpers.geom.**find_closest_point**(*pt*, *ref_pts*, *as_geom=True*)

Finds the closest point in a sequence of reference points to a given point.

This function calculates and returns the point closest to `pt` from a sequence of reference points `ref_pts`. The closest point can be returned either as a Shapely Point geometry (shapely.geometry.Point) or as a numpy.ndarray.

Parameters

- **pt** (*tuple | list | shapely.geometry.Point*) – Point for which the closest point is to be found.
- **ref_pts** (*Iterable | numpy.ndarray | list | tuple | shapely.geometry.base.BaseGeometry*) – Sequence of reference points to search for the closest point.
- **as_geom** (*bool*) – Whether to return the closest point as a shapely.geometry.Point; defaults to `True`.

Returns

Closest point to `pt` from `ref_pts`.

Return type

shapely.geometry.Point | numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import find_closest_point
>>> from pyhelpers._cache import example_dataframe
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
```

(continues on next page)

(continued from previous page)

```
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> # Find the city closest to London
>>> london = example_df.loc['London'].values
>>> ref_cities = example_df.loc['Birmingham':, :].values
>>> closest_to_london = find_closest_point(pt=london, ref_pts=ref_cities)
>>> closest_to_london.wkt  # Birmingham
'POINT (-1.9026911 52.4796992)'
>>> # Find the city closest to Leeds
>>> leeds = example_df.loc['Leeds'].values
>>> ref_cities = example_df.loc[:'Manchester', :].values
>>> closest_to_leeds = find_closest_point(pt=leeds, ref_pts=ref_cities)
>>> closest_to_leeds.wkt  # Manchester
'POINT (-2.2451148 53.4794892)'
>>> closest_to_leeds = find_closest_point(pt=leeds, ref_pts=ref_cities, as_geom=False)
>>> closest_to_leeds  # Manchester
array([-2.2451148, 53.4794892])
```

find_closest_points

pyhelpers.geom.**find_closest_points**(*pts*, *ref_pts*, *k=1*, *unique=False*, *as_geom=False*, *ret_idx=False*, *ret_dist=False*, ***kwargs*)

Finds the closest points from a list of reference points to a set of query points.

This function computes the closest points from a list of reference points `ref_pts` to a set of query points `pts`. Various options are available for customization, such as returning multiple nearest neighbors (`k`), removing duplicated points, returning points as Shapely Points (shapely.geometry.Point), returning indices of the closest points, and returning distances between `pts` and the closest points in `ref_pts`.

See also [GEOM-FCPB-1].

Parameters

- **pts** (*numpy.ndarray | list | tuple | Iterable | shapely.geometry.base.BaseGeometry*) – Array of query points with shape (n, 2).
- **ref_pts** (*numpy.ndarray | list | tuple | shapely.geometry.base.BaseGeometry*) – Array of reference points with shape (m, 2).
- **k** (*int | list*) – Number of closest neighbors to find; defaults to 1.
- **unique** (*bool*) – Whether to remove duplicated points from the results; defaults to False.
- **as_geom** (*bool*) – Whether to return the closest points as shapely.geometry.Point; defaults to `False`.
- **ret_idx** (*bool*) – Whether to return indices of the closest points in `ref_pts`; defaults to `False`.
- **ret_dist** (*bool*) – Whether to return distances between `pts` and the closest points in `ref_pts`; defaults to `False`.

- **kwargs** – [Optional] Additional parameters for the class scipy.spatial.cKDTree.

Returns

Closest points among `ref_pts` to each point in `pts`.

Return type

numpy.ndarray | shapely.geometry.MultiPoint

Examples:

```
>>> from pyhelpers.geom import find_closest_points
>>> from pyhelpers._cache import example_dataframe
>>> from shapely.geometry import LineString, MultiPoint
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> cities = [[-2.9916800, 53.4071991],  # Liverpool
...           [-4.2488787, 55.8609825],  # Glasgow
...           [-1.6131572, 54.9738474]]  # Newcastle
>>> ref_cities = example_df.to_numpy()
>>> closest_to_each = find_closest_points(pts=cities, ref_pts=ref_cities, k=1)
>>> closest_to_each  # Liverpool: Manchester; Glasgow: Manchester; Newcastle: Leeds
array([[-2.2451148, 53.4794892],
       [-2.2451148, 53.4794892],
       [-1.5437941, 53.7974185]])
>>> closest_to_each = find_closest_points(
...     pts=cities, ref_pts=ref_cities, k=1, as_geom=True)
>>> closest_to_each.wkt
'MULTIPOINT ((-2.2451148 53.4794892), (-2.2451148 53.4794892), (-1.5437941 53.7974185))'
>>> _, idx = find_closest_points(pts=cities, ref_pts=ref_cities, k=1, ret_idx=True)
>>> idx
array([2, 2, 3])
>>> _, _, dist = find_closest_points(cities, ref_cities, k=1, ret_idx=True, ret_dist=True)
>>> dist
array([0.75005697, 3.11232712, 1.17847198])
>>> cities_geoms_1 = LineString(cities)
>>> closest_to_each = find_closest_points(pts=cities_geoms_1, ref_pts=ref_cities, k=1)
>>> closest_to_each
array([[-2.2451148, 53.4794892],
       [-2.2451148, 53.4794892],
       [-1.5437941, 53.7974185]])
>>> cities_geoms_2 = MultiPoint(cities)
>>> closest_to_each = find_closest_points(cities_geoms_2, ref_cities, k=1, as_geom=True)
>>> closest_to_each.wkt
'MULTIPOINT ((-2.2451148 53.4794892), (-2.2451148 53.4794892), (-1.5437941 53.7974185))'
```

find_shortest_path

pyhelpers.geom.**find_shortest_path**(*points_sequence*, *ret_dist=False*, *as_geom=False*, ***kwargs*)

Finds the shortest path through a sequence of points.

This function constructs a graph where each point is connected to its two nearest neighbors and then uses Depth-First Search (DFS) starting from every point to find the lowest-cost path

among those traversals. The method is quick but yields only an approximate solution.

Parameters

- **points_sequence** (*numpy.ndarray | list | Iterable*) – Sequence of points, typically a 2D array or list of coordinates (e.g. `[[lon, lat], ...]`).
- **ret_dist** (*bool*) – Whether to return the path distance (sum of edge lengths) as the second element of a tuple; defaults to `False`.
- **as_geom** (*bool*) – Whether to return the sorted path as a `shapely.geometry.LineString` object (requires *shapely*); defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the class sklearn.neighbors.NearestNeighbors, such as `metric` (e.g. `'euclidean'`, `'haversine'`).

Returns

The sorted path sequence (numpy.ndarray), optionally wrapped in a `shapely.geometry.LineString` or a tuple with the distance.

Return type

numpy.ndarray | shapely.geometry.LineString | tuple[numpy.ndarray | shapely.geometry.LineString, float]

The original method using 2-Nearest Neighbors and DFS is a **Greedy Heuristic**. For $N > 4$ points, this method rarely finds the mathematically shortest path (optimal TSP-P solution) because the 2-NN graph may contain local cycles or may not be fully connected, preventing DFS from discovering the optimal global order.

Examples:

```
>>> from pyhelpers.geom import find_shortest_path
>>> from pyhelpers._cache import example_dataframe
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> example_df_ = example_df.sample(frac=1, random_state=1)
>>> example_df_
           Longitude   Latitude
City
Leeds       -1.543794  53.797418
Manchester  -2.245115  53.479489
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
>>> cities = example_df_.to_numpy()
>>> cities
array([[-1.5437941, 53.7974185],
       [-2.2451148, 53.4794892],
       [-0.1276474, 51.5073219],
```

(continues on next page)

(continued from previous page)

```
       [-1.9026911, 52.4796992]])
>>> cities_sorted = find_shortest_path(points_sequence=cities)
>>> cities_sorted
array([[-1.5437941, 53.7974185],
       [-2.2451148, 53.4794892],
       [-1.9026911, 52.4796992],
       [-0.1276474, 51.5073219]])
```

This example is illustrated below (see Figure 12):

```
>>> import matplotlib.pyplot as plt
>>> import matplotlib.gridspec as mgs
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(backend='TkAgg')
>>> fig = plt.figure(figsize=(7, 5))
>>> gs = mgs.GridSpec(1, 2, figure=fig)
>>> ax1 = fig.add_subplot(gs[:, 0])
>>> ax1.plot(cities[:, 0], cities[:, 1], label='original')
>>> for city, i, lonlat in zip(example_df_.index, range(len(cities)), cities):
...     ax1.scatter(lonlat[0], lonlat[1])
...     ax1.annotate(city + f' ({i})', xy=lonlat + 0.05)
>>> ax1.legend(loc=3)
>>> ax2 = fig.add_subplot(gs[:, 1])
>>> ax2.plot(cities_sorted[:, 0], cities_sorted[:, 1], label='sorted', color='orange')
>>> for city, i, lonlat in zip(example_df.index[::-1], range(len(cities)), cities_sorted):
...     ax2.scatter(lonlat[0], lonlat[1])
...     ax2.annotate(city + f' ({i})', xy=lonlat + 0.05)
>>> ax2.legend(loc=3)
>>> fig.tight_layout()
>>> fig.show()
>>> # from pyhelpers.store import save_figure
>>> # path_to_fig_ = "docs/source/_images/geom-find_shortest_path-demo"
>>> # save_figure(fig, f"{path_to_fig_}.svg", verbose=True)
>>> # save_figure(fig, f"{path_to_fig_}.pdf", verbose=True)
```

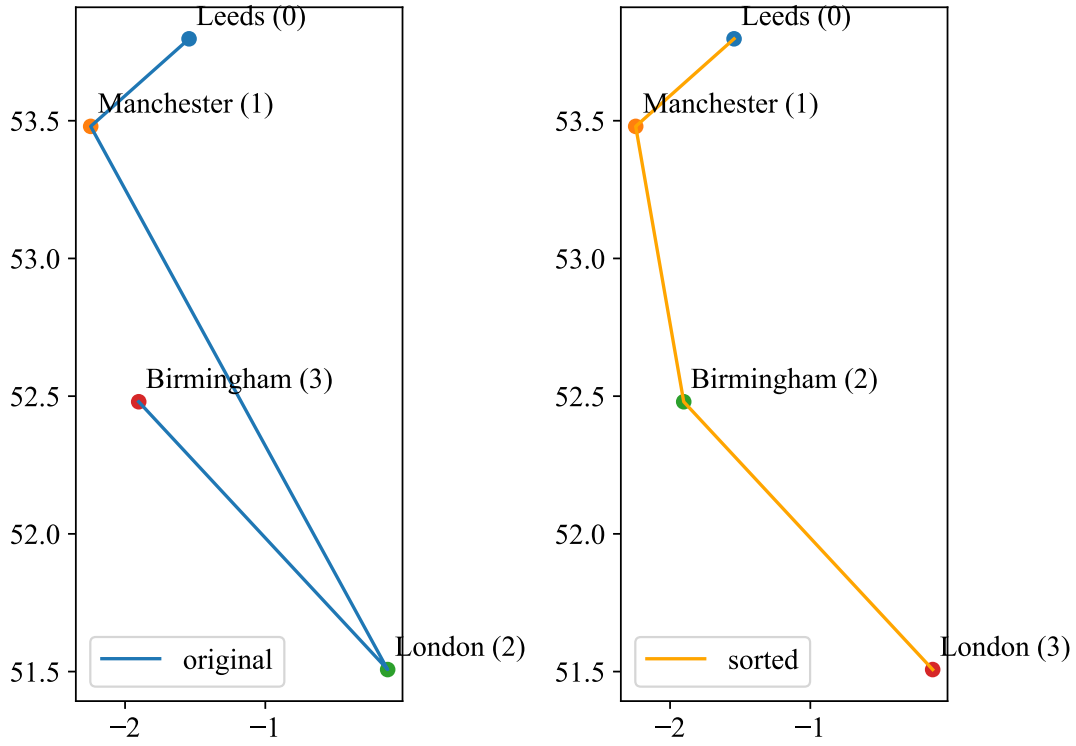

Figure 12: An example of sorting a sequence of points given the shortest path.

4.5.2 Geometric properties and shape sketching

Properties

1anti-flashwhitewhite

get_midpoint(x1, y1, x2, y2[, as_geom])	Gets the midpoint between two points (applicable for vectorized computation).
get_geometric_midpoint(pt1, pt2[, as_geom])	Gets the midpoint between two points.
get_geometric_midpoint_calc(pt1, pt2[, as_geom])	Gets the midpoint between two points by pure calculation.
get_rectangle_centroid(rectangle[, as_geom])	Gets coordinates of the centroid of a rectangle.
get_square_vertices(ctr_x, ctr_y, side_length)	Gets the four vertices of a square given its center and side length.
get_square_vertices_calc(ctr_x, ctr_y, ...)	Gets the four vertices of a square given its center and side length (by elementary calculation).

get_midpoint

pyhelpers.geom.**get_midpoint**(*x1*, *y1*, *x2*, *y2*, *as_geom=False*)

Gets the midpoint between two points (applicable for vectorized computation).

Parameters

- **x1** (*float | int | Iterable | numpy.ndarray*) – Longitude(s) or easting(s) of a point (an array of points).
- **y1** (*float | int | Iterable | numpy.ndarray*) – Latitude(s) or northing(s) of a point (an array of points).
- **x2** (*float | int | Iterable | numpy.ndarray*) – Longitude(s) or easting(s) of another point (another array of points).
- **y2** (*float | int | Iterable | numpy.ndarray*) – Latitude(s) or northing(s) of another point (another array of points).
- **as_geom** (*bool*) – Whether to return shapely.geometry.Point; defaults to `False`.

Returns

The midpoint between `(x1, y1)` and `(x2, y2)` (or midpoints between two sequences of points).

Return type

numpy.ndarray | shapely.geometry.Point | shapely.geometry.MultiPoint

Examples:

```
>>> from pyhelpers.geom import get_midpoint
>>> x_1, y_1 = 1.5429, 52.6347
>>> x_2, y_2 = 1.4909, 52.6271
>>> midpt = get_midpoint(x_1, y_1, x_2, y_2)
>>> midpt
array([ 1.5169, 52.6309])
>>> midpt = get_midpoint(x_1, y_1, x_2, y_2, as_geom=True)
>>> midpt.wkt
'POINT (1.5169 52.6309)'
>>> x_1, y_1 = (1.5429, 1.4909), (52.6347, 52.6271)
>>> x_2, y_2 = [2.5429, 2.4909], [53.6347, 53.6271]
>>> midpt = get_midpoint(x_1, y_1, x_2, y_2)
>>> midpt
array([[ 2.0429, 53.1347],
       [ 1.9909, 53.1271]])
>>> midpt = get_midpoint(x_1, y_1, x_2, y_2, as_geom=True)
>>> midpt.wkt
'MULTIPOINT (2.0429 53.1347, 1.9909 53.1271)'
```

get_geometric_midpoint

pyhelpers.geom.**get_geometric_midpoint**(*pt1*, *pt2*, *as_geom=False*)

Gets the midpoint between two points.

Parameters

- **pt1** (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – One point.

- **pt2** (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – Another point represented similarly to `pt1`.
- **as_geom** (*bool*) – Whether to return shapely.geometry.Point; defaults to `False`.

Returns
The midpoint between `pt1` and `pt2`.

Return type
tuple | shapely.geometry.Point | None

Examples:

```
>>> from pyhelpers.geom import get_geometric_midpoint
>>> pt_1, pt_2 = (1.5429, 52.6347), (1.4909, 52.6271)
>>> geometric_midpoint = get_geometric_midpoint(pt_1, pt_2)
>>> geometric_midpoint
(1.5169, 52.6309)
>>> geometric_midpoint = get_geometric_midpoint(pt_1, pt_2, as_geom=True)
>>> geometric_midpoint.wkt
'POINT (1.5169 52.6309)'
```

See also

- Examples for the function *get_geometric_midpoint_calc()*.

get_geometric_midpoint_calc

pyhelpers.geom.**get_geometric_midpoint_calc**(*pt1*, *pt2*, *as_geom=False*)

Gets the midpoint between two points by pure calculation.

See also [GEOM-GGMC-1].

Parameters

- **pt1** (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – One point.
- **pt2** (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – Another point represented similarly to `pt1`.
- **as_geom** (*bool*) – Whether to return shapely.geometry.Point; defaults to `False`.

Returns
The midpoint between `pt1` and `pt2`.

Return type
tuple | shapely.geometry.Point | None

Examples:

```
>>> from pyhelpers.geom import get_geometric_midpoint_calc
>>> pt_1, pt_2 = (1.5429, 52.6347), (1.4909, 52.6271)
>>> geometric_midpoint = get_geometric_midpoint_calc(pt_1, pt_2)
```

(continues on next page)

(continued from previous page)

```
>>> geometric_midpoint
(1.5168977420748175, 52.630902845583094)
>>> geometric_midpoint = get_geometric_midpoint_calc(pt_1, pt_2, as_geom=True)
>>> geometric_midpoint.wkt
'POINT (1.5168977420748175 52.630902845583094)'
```

See also

- Examples for the function *get_geometric_midpoint()*.

get_rectangle_centroid

pyhelpers.geom.**get_rectangle_centroid**(*rectangle*, *as_geom=False*)

Gets coordinates of the centroid of a rectangle.

Parameters

- **rectangle** (*list | tuple | numpy.ndarray | shapely.geometry.Polygon | shapely.geometry.MultiPolygon*) – Variable/object representing a rectangle.
- **as_geom** (*bool*) – Whether to return a shapely.geometry.Point object; defaults to False.

Returns

Coordinates of the centroid of the rectangle.

Return type

numpy.ndarray | shapely.geometry.Point

Examples:

```
>>> from pyhelpers.geom import get_rectangle_centroid
>>> from shapely.geometry import Polygon
>>> import numpy
>>> coords_1 = [[0, 0], [0, 1], [1, 1], [1, 0]]
>>> rectangle = Polygon(coords_1)
>>> rectangle_centroid = get_rectangle_centroid(rectangle)
>>> rectangle_centroid
array([0.5, 0.5])
>>> rectangle = numpy.array(coords_1)
>>> rectangle_centroid = get_rectangle_centroid(rectangle)
>>> rectangle_centroid
array([0.5, 0.5])
>>> rectangle_centroid = get_rectangle_centroid(rectangle, as_geom=True)
>>> type(rectangle_centroid)
shapely.geometry.point.Point
>>> rectangle_centroid.wkt
'POINT (0.5 0.5)'
>>> rectangle = [[(0, 0), (0, 1), (1, 1), (1, 0)], [(1, 1), (1, 2), (2, 2), (2, 1)]]
>>> rectangle_centroid = get_rectangle_centroid(rectangle)
>>> rectangle_centroid
array([1., 1.])
```

get_square_vertices

pyhelpers.geom.**get_square_vertices**(*ctr_x*, *ctr_y*, *side_length*, *rotation_theta=0*)

Gets the four vertices of a square given its center and side length.

See also [GEOM-GSV-1].

Parameters

- **ctr_x** (*int | float*) – X-coordinate of the square's center.
- **ctr_y** (*int | float*) – Y-coordinate of the square's center.
- **side_length** (*int | float*) – Side length of the square.
- **rotation_theta** (*int | float*) – Rotation angle (in degrees) to rotate the square anticlockwise; defaults to 0.

Returns

Vertices of the square as an array([Lower left, Upper left, Upper right, Lower right]).

Return type

numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import get_square_vertices
>>> ctr_1, ctr_2 = -5.9375, 56.8125
>>> side_len = 0.125
>>> vts = get_square_vertices(ctr_1, ctr_2, side_len, rotation_theta=0)
>>> vts
array([[-6.   , 56.75 ],
       [-6.   , 56.875],
       [-5.875, 56.875],
       [-5.875, 56.75 ]])
>>> # Rotate the square by 30° (anticlockwise)
>>> vts = get_square_vertices(ctr_1, ctr_2, side_len, rotation_theta=30)
>>> vts
array([[-5.96037659, 56.72712341],
       [-6.02287659, 56.83537659],
       [-5.91462341, 56.89787659],
       [-5.85212341, 56.78962341]])
```

get_square_vertices_calc

pyhelpers.geom.**get_square_vertices_calc**(*ctr_x*, *ctr_y*, *side_length*, *rotation_theta=0*)

Gets the four vertices of a square given its center and side length (by elementary calculation).

See also [GEOM-GSVC-1].

Parameters

- **ctr_x** (*int | float*) – X-coordinate of the square's center.
- **ctr_y** (*int | float*) – Y-coordinate of the square's center.
- **side_length** (*int | float*) – Side length of the square.
- **rotation_theta** (*int | float*) – Rotation angle (in degrees) to rotate the square anticlockwise; defaults to 0.

Returns

Vertices of the square as an array([Lower left, Upper left, Upper right, Lower right]).

Return type

numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import get_square_vertices_calc
>>> ctr_1, ctr_2 = -5.9375, 56.8125
>>> side_len = 0.125
>>> vts = get_square_vertices_calc(ctr_1, ctr_2, side_len, rotation_theta=0)
>>> vts
array([[-6.   , 56.75 ],
       [-6.   , 56.875],
       [-5.875, 56.875],
       [-5.875, 56.75 ]])
>>> # Rotate the square by 30° (anticlockwise)
>>> vts = get_square_vertices_calc(ctr_1, ctr_2, side_len, rotation_theta=30)
>>> vts
array([[-5.96037659, 56.72712341],
       [-6.02287659, 56.83537659],
       [-5.91462341, 56.89787659],
       [-5.85212341, 56.78962341]])
```

See also

- Examples for the function *get_square_vertices()*.

Sketching

1anti-flashwhitewhite

sketch_square(ctr_x, ctr_y, side_length[, ...])	Sketches a square on a plot given its center coordinates, side length and rotation angle.

sketch_square

pyhelpers.geom.**sketch_square**(*ctr_x*, *ctr_y*, *side_length*, *rotation_theta=0*, *annotation=False*, *annot_font_size=12*, *fig_size=(6.4, 4.8)*, *ret_vertices=False*, ***kwargs*)

Sketches a square on a plot given its center coordinates, side length and rotation angle.

This function plots a square with its center at coordinates `(ctr_x, ctr_y)`, side length `side_length`, and rotated by *rotation_theta* degrees (anticlockwise).

Parameters

- **ctr_x** (*int | float*) – X coordinate of the center of the square.
- **ctr_y** (*int | float*) – Y coordinate of the center of the square.
- **side_length** (*int | float*) – Side length of the square.

- **rotation_theta** (*int | float*) – Rotation angle of the square in degrees; defaults to 0.
- **annotation** (*bool*) – Whether to annotate vertices of the square; defaults to True.
- **annot_font_size** (*int*) – Font size of annotation texts; defaults to 12.
- **fig_size** (*tuple | list*) – Size of the figure; defaults to (6.4, 4.8).
- **ret_vertices** (*bool*) – Whether to return the vertices of the square; defaults to False.
- **kwargs** – Additional parameters for matplotlib.axes.Axes.annotate.

Returns

Vertices of the square as an array([ll, ul, ur, lr]).

Return type

numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import sketch_square
>>> from pyhelpers.settings import mpl_preferences
>>> import matplotlib.pyplot as plt
>>> mpl_preferences()
>>> c1, c2 = 1, 1
>>> side_len = 2
>>> sketch_square(c1, c2, side_len, rotation_theta=0, annotation=True, fig_size=(5, 5))
>>> plt.show()
>>> # from pyhelpers.store import save_fig
>>> # path_to_fig_ = "docs/source/_images/geom-sketch_square-demo-1"
>>> # save_fig(f"{path_to_fig_}.svg", verbose=True)
>>> # save_fig(f"{path_to_fig_}.pdf", verbose=True)
```

The above example is illustrated in Figure 13:

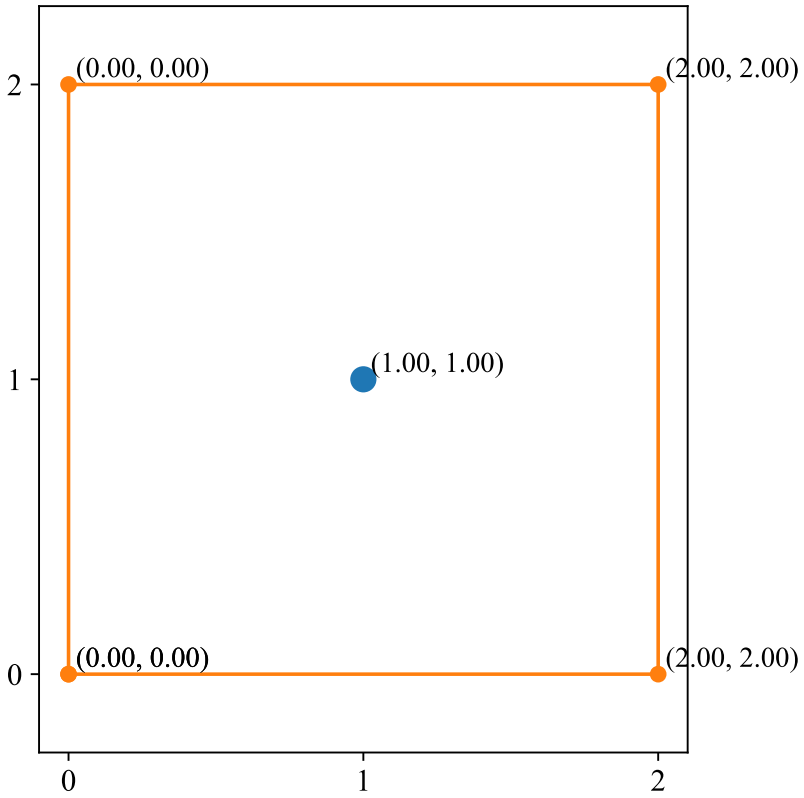

Figure 13: An example of a sketch of a square, created by the function *sketch_square()*.

```
>>> sketch_square(c1, c2, side_len, rotation_theta=75, annotation=True, fig_size=(5, 5))
>>> plt.show()
>>> # save_fig("docs/source/_images/geom-sketch_square-demo-2.svg", verbose=True)
>>> # save_fig("docs/source/_images/geom-sketch_square-demo-2.pdf", verbose=True)
```

This second example is illustrated in Figure 14:

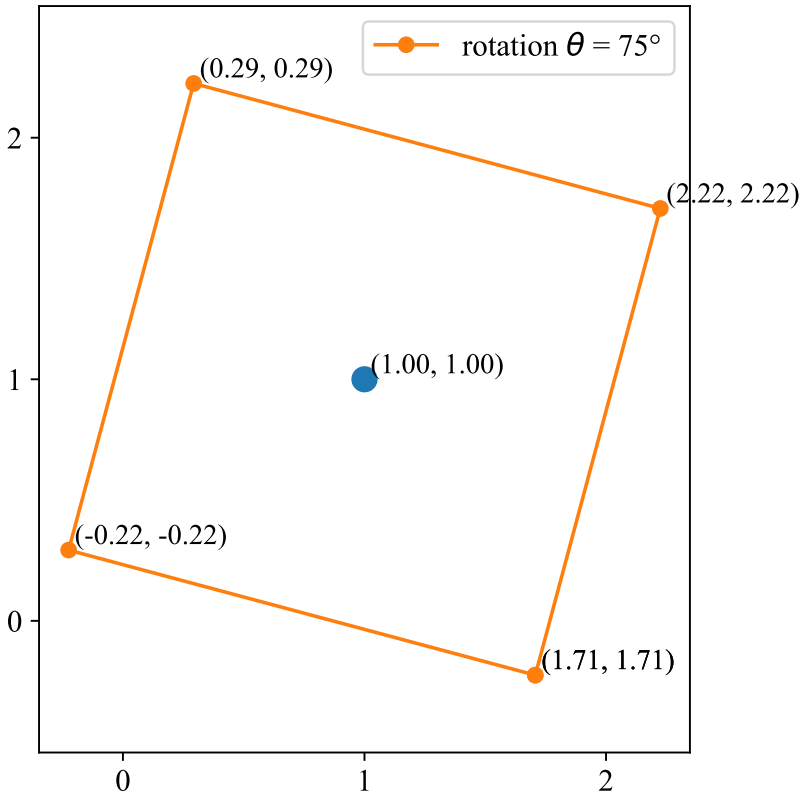

Figure 14: An example of a sketch of a square rotated 75 degrees anticlockwise about the center.

4.5.3 Geometric data transformation

Data type

1anti-flashwhitewhite

transform_point_type(*pts[, as_geom])	Transforms iterable data to geometric type or vice versa.
get_point_coordinates(pt)	Extracts (x, y) coordinates from a point-like object.
get_coordinates_as_array(geom_obj[, unique, ...])	Retrieves an array of coordinates from the input geometry object.

transform_point_type

pyhelpers.geom.**transform_point_type**(**pts*, *as_geom=True*)

Transforms iterable data to geometric type or vice versa.

Parameters

- **pts** (*list | tuple | shapely.geometry.Point*) – Iterable data representing points (e.g. list of lists/tuples).
- **as_geom** (*bool*) – Whether to return points as shapely.geometry.Point; defaults to True.

Returns

A sequence of points, including `None` if errors occur.

Return type

Generator

Examples:

```
>>> from pyhelpers.geom import transform_point_type
>>> from pyhelpers._cache import example_dataframe
>>> from shapely.geometry import Point
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> pt1 = example_df.loc['London'].values  # array([-0.1276474, 51.5073219])
>>> pt2 = example_df.loc['Birmingham'].values  # array([-1.9026911, 52.4796992])
>>> geom_points = transform_point_type(pt1, pt2)
>>> for x in geom_points:
...     print(x)
POINT (-0.1276474 51.5073219)
POINT (-1.9026911 52.4796992)
>>> geom_points = transform_point_type(pt1, pt2, as_geom=False)
>>> for x in geom_points:
...     print(x)
[-0.1276474 51.5073219]
[-1.9026911 52.4796992]
>>> pt1, pt2 = map(lambda p: Point(p), (pt1, pt2))
>>> geom_points = transform_point_type(pt1, pt2)
>>> for x in geom_points:
...     print(x)
POINT (-0.1276474 51.5073219)
POINT (-1.9026911 52.4796992)
>>> geom_points = transform_point_type(pt1, pt2, as_geom=False)
>>> for x in geom_points:
...     print(x)
(-0.1276474, 51.5073219)
(-1.9026911, 52.4796992)
>>> geom_points_ = transform_point_type(Point([1, 2, 3]), as_geom=False)
>>> for x in geom_points_:
...     print(x)
(1.0, 2.0, 3.0)
```

get_point_coordinates

pyhelpers.geom.**get_point_coordinates**(*pt*)

Extracts (x, y) coordinates from a point-like object.

Supported types include Shapely Points, lists, tuples, and NumPy arrays.

Parameters

pt (*shapely.geometry.Point | list | tuple | numpy.ndarray*) – A point-like object containing at least two coordinates.

Returns

A tuple of (x, y) coordinates.

Return type
tuple[float, float]

Raises
ValueError – If the input format is unrecognized or has insufficient length.

Examples:

```
>>> from pyhelpers.geom import get_point_coordinates
>>> from shapely.geometry import Point
>>> get_point_coordinates(Point(1.0, 2.0))
(1.0, 2.0)
>>> get_point_coordinates([1.5, 2.5, 0.0])
(1.5, 2.5)
```

get_coordinates_as_array

pyhelpers.geom.**get_coordinates_as_array**(*geom_obj*, *unique=False*, *dtype=None*)

Retrieves an array of coordinates from the input geometry object.

Parameters

- **geom_obj** (*numpy.ndarray | Iterable | shapely.Geometry*) – Input geometry object.
- **unique** (*bool*) – Whether to remove duplicated points; defaults to `False`.
- **dtype** (*None | type | str*) – The desired data-type for the array; if `dtype=None` (default), it takes the default `dtype` adopted by NumPy.

Returns
Array of coordinates extracted from the geometry object.

Return type
numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import get_coordinates_as_array
>>> from pyhelpers._cache import example_dataframe
>>> from shapely.geometry import Polygon, MultiPoint, MultiPolygon, GeometryCollection
>>> from numpy import array_equal
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> geom_obj_1 = example_df.to_numpy()
>>> geom_coords_1 = get_coordinates_as_array(geom_obj=geom_obj_1)
>>> geom_coords_1
array([[-0.1276474, 51.5073219],
       [-1.9026911, 52.4796992],
       [-2.2451148, 53.4794892],
```

(continues on next page)

(continued from previous page)

```
       [-1.5437941, 53.7974185]])
>>> geom_obj_2 = Polygon(example_df.to_numpy())
>>> geom_coords_2 = get_coordinates_as_array(geom_obj=geom_obj_2, unique=True)
>>> array_equal(geom_coords_2, geom_coords_1)
True
>>> geom_obj_3 = MultiPoint(example_df.to_numpy())
>>> geom_coords_3 = get_coordinates_as_array(geom_obj=geom_obj_3)
>>> array_equal(geom_coords_3, geom_coords_1)
True
>>> geom_obj_4 = MultiPolygon([geom_obj_2, geom_obj_2])
>>> geom_coords_4 = get_coordinates_as_array(geom_obj=geom_obj_4, unique=True)
>>> array_equal(geom_coords_4, geom_coords_1)
True
>>> geom_obj_5 = GeometryCollection([geom_obj_2, geom_obj_3, geom_obj_4])
>>> geom_coords_5 = get_coordinates_as_array(geom_obj=geom_obj_5, unique=True)
>>> array_equal(geom_coords_5, geom_coords_1)
True
```

Coordinate system

1anti-flashwhitewhite

wgs84_to_osgb36(longitudes, latitudes[, ...])	Converts latitude and longitude (WGS84) to British national grid (OSGB36).
osgb36_to_wgs84(eastings, northings[, as_array])	Converts British national grid (OSGB36) to latitude and longitude (WGS84).

wgs84_to_osgb36

pyhelpers.geom.**wgs84_to_osgb36**(*longitudes*, *latitudes*, *as_array=False*, ***kwargs*)

Converts latitude and longitude (WGS84) to British national grid (OSGB36).

Parameters

- **longitudes** (*int | float | Iterable*) – Longitude of a point on Earth's surface in degrees; abbreviated as *long.*, λ or *lambda*.
- **latitudes** (*int | float | Iterable*) – Latitude of a point on Earth's surface in degrees; abbreviated as *lat.*, φ or *phi*.
- **as_array** (*bool*) – Whether to return an array; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the function pyproj.Transformer.transform.

Returns

Geographic Cartesian coordinates *(Easting, Northing)* or *(X, Y)*.

Return type

tuple | numpy.ndarry

Examples:

```
>>> from pyhelpers.geom import wgs84_to_osgb36
>>> from pyhelpers._cache import example_dataframe
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> lon, lat = example_df.loc['London'].values
>>> x, y = wgs84_to_osgb36(longitudes=lon, latitudes=lat)
>>> print(f"London (Easting, Northing): {(x, y)}")
London (Easting, Northing): (530039.558844505, 180371.68016544735)
>>> lonlat_array = example_df.to_numpy()
>>> lonlat_array
array([[-0.1276474, 51.5073219],
       [-1.9026911, 52.4796992],
       [-2.2451148, 53.4794892],
       [-1.5437941, 53.7974185]])
>>> lons, lats = lonlat_array.T  # lonlat_array[:, 0], lonlat_array[:, 1]
>>> xs, ys = wgs84_to_osgb36(longitudes=lons, latitudes=lats)
>>> xs
array([530039.5588445 , 406705.8870136 , 383830.03903573, 430147.44735387])
>>> ys
array([180371.68016545, 286868.16664219, 398113.05583091, 433553.32711728])
>>> xy_array = wgs84_to_osgb36(longitudes=lons, latitudes=lats, as_array=True)
>>> xy_array
array([[530039.5588445 , 180371.68016545],
       [406705.8870136 , 286868.16664219],
       [383830.03903573, 398113.05583091],
       [430147.44735387, 433553.32711728]])
```

osgb36_to_wgs84

pyhelpers.geom.**osgb36_to_wgs84**(*eastings*, *northings*, *as_array=False*, ***kwargs*)

Converts British national grid (OSGB36) to latitude and longitude (WGS84).

Parameters

- **eastings** (*int | float | Iterable*) – Easting (X) coordinate, eastward-measured distance.
- **northings** (*int | float | Iterable*) – Northing (Y) coordinate, northward-measured distance.
- **as_array** (*bool*) – Whether to return an array; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the function pyproj.Transformer.transform.

Returns

Geographic coordinates *(Longitude, Latitude)*.

Return type

tuple | numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import osgb36_to_wgs84
>>> from pyhelpers._cache import example_dataframe
>>> example_df = example_dataframe(osgb36=True)
>>> example_df
                Easting       Northing
City
London      530039.558844  180371.680166
Birmingham  406705.887014  286868.166642
Manchester  383830.039036  398113.055831
Leeds       430147.447354  433553.327117
>>> x, y = example_df.loc['London'].values
>>> lon, lat = osgb36_to_wgs84(eastings=x, northings=y)
>>> print(f"London (Longitude, Latitude): {(lon, lat)}")
London (Longitude, Latitude): (-0.12764738749567286, 51.50732189539607)
>>> xy_array = example_df.to_numpy()
>>> xs, ys = xy_array.T  # xy_array[:, 0], xy_array[:, 1]
>>> lons, lats = osgb36_to_wgs84(eastings=xs, northings=ys)
>>> lons
array([-0.12764739, -1.90269109, -2.24511479, -1.54379409])
>>> lats
array([51.5073219, 52.4796992, 53.4794892, 53.7974185])
>>> lonlat_array = osgb36_to_wgs84(eastings=xs, northings=ys, as_array=True)
>>> lonlat_array
array([[-0.12764739, 51.5073219 ],
       [-1.90269109, 52.4796992 ],
       [-2.24511479, 53.4794892 ],
       [-1.54379409, 53.7974185 ]])
```

Dimension / Projection

1anti-flashwhitewhite

drop_axis(geom[, axis, as_array])	Drops an axis from a given 3D geometry object.
project_point_to_line(point, line[, ...])	Finds the projected point from a given point to a line.

drop_axis

pyhelpers.geom.**drop_axis**(*geom*, *axis='z'*, *as_array=False*)

Drops an axis from a given 3D geometry object.

Parameters

- **geom** (*shapely.geometry.base.BaseGeometry*) – Geometry object that has *X*, *Y* and *Z* coordinates.
- **axis** (*str*) – Axis to drop; options include `'x'`, `'y'` and `'z'`; defaults to `'z'`.
- **as_array** (*bool*) – Whether to return an array representation; defaults to `False`.

Returns

Geometry object without the specified axis, or an array representation.

Return type

shapely.geometry.base.BaseGeometry | numpy.ndarray

Examples:

```
>>> from pyhelpers.geom import drop_axis
>>> from shapely.geometry import Point, LineString, Polygon, MultiLineString
>>> geom_1 = Point([1, 2, 3])
>>> geom_1.wkt
'POINT Z (1 2 3)'
>>> geom_1_ = drop_axis(geom_1, 'x')
>>> geom_1_.wkt
'POINT (2 3)'
>>> geom_1_ = drop_axis(geom_1, 'x', as_array=True)
>>> geom_1_
array([2., 3.])
>>> geom_2 = LineString([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
>>> geom_2.wkt
'LINESTRING Z (1 2 3, 2 3 4, 3 4 5)'
>>> geom_2_ = drop_axis(geom_2, 'y')
>>> geom_2_.wkt
'LINESTRING (1 3, 2 4, 3 5)'
>>> geom_2_ = drop_axis(geom_2, 'y', as_array=True)
>>> geom_2_
array([[1., 3.],
       [2., 4.],
       [3., 5.]])
>>> geom_3 = Polygon([[6, 3, 5], [6, 3, 0], [6, 1, 0], [6, 1, 5], [6, 3, 5]])
>>> geom_3.wkt
'POLYGON Z ((6 3 5, 6 3 0, 6 1 0, 6 1 5, 6 3 5))'
>>> geom_3_ = drop_axis(geom_3, 'z')
>>> geom_3_.wkt
'POLYGON ((6 3, 6 3, 6 1, 6 1, 6 3))'
>>> geom_3_ = drop_axis(geom_3, 'z', as_array=True)
>>> geom_3_
array([[6., 3.],
       [6., 3.],
       [6., 1.],
       [6., 1.],
       [6., 3.]])
>>> ls1 = LineString([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
>>> ls2 = LineString([[2, 3, 4], [1, 2, 3], [3, 4, 5]])
>>> geom_4 = MultiLineString([ls1, ls2])
>>> geom_4.wkt
'MULTILINESTRING Z ((1 2 3, 2 3 4, 3 4 5), (2 3 4, 1 2 3, 3 4 5))'
>>> geom_4_ = drop_axis(geom_4, 'z')
>>> geom_4_.wkt
'MULTILINESTRING ((1 2, 2 3, 3 4), (2 3, 1 2, 3 4))'
>>> geom_4_ = drop_axis(geom_4, 'z', as_array=True)
>>> geom_4_
array([[[1., 2.],
        [2., 3.],
        [3., 4.]],
       [[2., 3.],
        [1., 2.],
        [3., 4.]]])
```

project_point_to_line

pyhelpers.geom.**project_point_to_line**(*point*, *line*, *drop_dimension=None*)

Finds the projected point from a given point to a line.

Parameters

- **point** (*shapely.geometry.Point*) – Point geometry object representing the starting point.
- **line** (*shapely.geometry.LineString*) – Line geometry object to which the point is projected.
- **drop_dimension** (*str | None*) – Dimension to drop during projection; options include `'x'`, `'y'` and `'z'`; defaults to `None`.

Returns

Tuple containing the original point (with all or partial dimensions, based on `drop_dimension`) and the projected point on the line.

Return type

tuple

Examples:

```
>>> from pyhelpers.geom import project_point_to_line
>>> from shapely.geometry import Point, LineString, MultiPoint
>>> pt = Point([399297, 655095, 43])
>>> ls = LineString([[399299, 655091, 42], [399295, 655099, 42]])
>>> _, pt_proj = project_point_to_line(point=pt, line=ls)
>>> pt_proj.wkt
'POINT Z (399297 655095 42)'
```

This example is illustrated below (see Figure 15):

```
>>> import matplotlib.pyplot as plt
>>> from pyhelpers.settings import mpl_preferences
>>> mpl_preferences(font_size=12)
>>> fig = plt.figure()
>>> ax = fig.add_subplot(projection='3d')
>>> ls_zs = list(map(lambda c: c[2], ls.coords))
>>> ax.plot(ls.coords.xy[0], ls.coords.xy[1], ls_zs, label='Line')
>>> ax.scatter(pt.x, pt.y, pt.z, label='Point')
>>> ax.scatter(pt_proj.x, pt_proj.y, pt_proj.z, label='Projected point')
>>> for i in MultiPoint([*ls.coords, pt, pt_proj]).geoms:
...     pos = tuple(map(int, i.coords[0]))
...     ax.text3D(pos[0], pos[1], pos[2], str(pos))
>>> ax.legend(loc=3)
>>> fig.tight_layout()
>>> ax.set_xticklabels([])
>>> ax.set_yticklabels([])
>>> ax.set_zticklabels([])
>>> fig.show()
>>> # from pyhelpers.store import save_figure
>>> # path_to_fig_ = "docs/source/_images/geom-project_point_to_line-demo"
>>> # save_figure(fig, f"{path_to_fig_}.svg", verbose=True)
>>> # save_figure(fig, f"{path_to_fig_}.pdf", verbose=True)
```

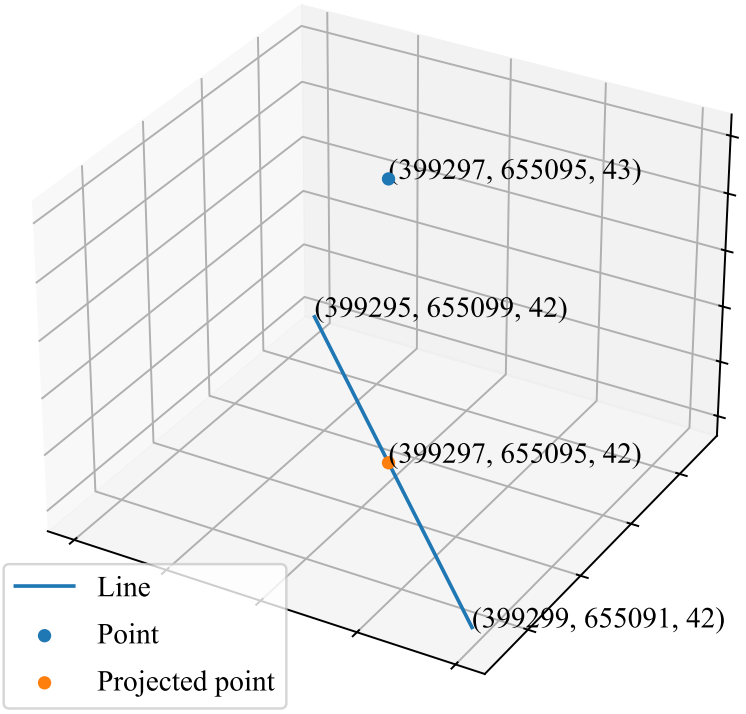

Figure 15: An example of projecting a point onto a line.

4.6 text

Manipulation of textual data.

4.6.1 Textual data (pre)processing

1anti-flashwhitewhite

clean_html_text(input_text)	Clean and normalize text extracted from HTML content.
remove_punctuation(input_text[, ...])	Removes punctuation from textual data.
get_acronym(input_text[, only_capitals, ...])	Generates an acronym (in capital letters) from textual data.
split_on_uppercase(input_text[, join_with, ...])	Extracts words from a string by splitting it at occurrences of uppercase letters.
numeral_english_to_arabic(input_text)	Converts a number written in English words into its equivalent numerical value represented in Arabic numerals.
count_words(input_text[, lowercase, ...])	Counts the occurrences of each word in the given text.
calculate_idf(documents[, lowercase, ...])	Calculates Inverse Document Frequency (IDF) for a sequence of textual documents.

continues on next page

Table 29 – continued from previous page

calculate_tfidf(documents, **kwargs)	Calculates TF-IDF (Term Frequency-Inverse Document Frequency) for the given textual documents.

clean_html_text

pyhelpers.text.**clean_html_text**(*input_text*)

Clean and normalize text extracted from HTML content.

Performs multiple cleaning operations on HTML text including:

- Decoding HTML entities (including double-encoded entities)
- Converting non-breaking spaces to regular spaces
- Removing all HTML tags
- Normalizing whitespace and trimming the result

Parameters

input_text (*str*) – Raw text containing HTML markup and entities.

Returns

Cleaned text with all HTML artifacts removed and normalized whitespace.

Return type

str

Examples:

```
>>> from pyhelpers.text import clean_html_text
>>> clean_html_text('&lt;p&gt;Hello world!&lt;/p&gt;')
'Hello world!'
```

remove_punctuation

pyhelpers.text.**remove_punctuation**(*input_text*, *normalize_whitespace=True*, *preserve_kebab_case=True*, *preserve_snake_case=True*, *exclude=None*)

Removes punctuation from textual data.

Parameters

- **input_text** (*str*) – The input text.
- **normalize_whitespace** (*bool*) – Whether to collapse all whitespace into single spaces. Defaults to `True`.
- **preserve_kebab_case** (*bool*) – Whether to preserve hyphens in Kebab case (e.g., `'kebab-case'`). Defaults to `True`.
- **preserve_snake_case** – Whether to preserve underscores in Snake case (e.g. `'snake_case'`). Defaults to `True`.
- **exclude** (*str | list | set | None*) – Punctuation marks to always keep, overriding other parameters. Defaults to `None`.

Returns

The processed text that is without punctuation (and optionally without whitespace).

Return type

str

Examples:

```
>>> from pyhelpers.text import remove_punctuation
>>> remove_punctuation('Hello, world!')
'Hello world'
>>> input_text = '   How   are you? '
>>> remove_punctuation(input_text, normalize_whitespace=True)
'How are you'
>>> remove_punctuation(input_text, normalize_whitespace=False)
'How   are you'
>>> input_text = 'No-punctuation!'
>>> remove_punctuation(input_text, preserve_kebab_case=False)
'No punctuation'
>>> input_text = 'Hello world!  This is a test. :-)'
>>> remove_punctuation(input_text)
'Hello world This is a test'
>>> remove_punctuation(input_text, normalize_whitespace=False)
'Hello world    This is a test'
>>> input_text = "The 'hyphen' is-cool; but underscores_are_not."
>>> remove_punctuation(input_text)
'The hyphen is-cool but underscores_are_not'
>>> remove_punctuation(input_text, preserve_kebab_case=False, exclude=';')
'The hyphen is cool; but underscores_are_not'
```

get_acronym

pyhelpers.text.**get_acronym**(*input_text*, *only_capitals=False*, *capitals_in_words=False*, *keep_punctuation=False*)

Generates an acronym (in capital letters) from textual data.

An acronym is typically formed by taking the initial letters of each word in a phrase and combining them into a single string of uppercase letters.

Parameters

- **input_text** (*str*) – The input text from which to generate the acronym.
- **only_capitals** (*bool*) – Whether to include only capital letters in the acronym; defaults to False.
- **capitals_in_words** (*bool*) – Whether to treat all capital letters within words as part of the acronym; defaults to False.
- **keep_punctuation** (*bool*) – Whether to retain punctuation in the input text; defaults to False.

Returns

The acronym generated from the input text.

Return type

str

Examples:

```
>>> from pyhelpers.text import get_acronym
>>> text_a = 'This is an apple.'
>>> get_acronym(text_a)
'TIAA'
>>> text_b = "I'm at the University of Birmingham."
>>> get_acronym(text_b, only_capitals=True)
'IUB'
>>> text_c = 'There is a "ConnectionError"!'
>>> get_acronym(text_c, capitals_in_words=True)
'TIACE'
>>> get_acronym(text_c, only_capitals=True, capitals_in_words=True)
'TCE'
>>> get_acronym(text_c, only_capitals=True, capitals_in_words=True, keep_punctuation=True)
'T"CE"!'
>>> get_acronym(text_c, only_capitals=True, keep_punctuation=True)
'T"C"!'
>>> get_acronym(text_c, capitals_in_words=True, keep_punctuation=True)
'TIA"CE"!'
```

split_on_uppercase

pyhelpers.text.**split_on_uppercase**(*input_text*, *join_with=None*, *split_numbers=True*)

Extracts words from a string by splitting it at occurrences of uppercase letters.

This function takes a string and splits it into individual words wherever an uppercase letter is encountered. Optionally, it can join these words back into a single string using a specified delimiter.

Parameters

- **input_text** (*str*) – Input text containing words starting with uppercase letters.
- **join_with** (*str | None*) – Optional delimiter to join words. Defaults to None.
- **split_numbers** (*bool*) – Whether to split numbers from letters; defaults to True.

Returns

If join_with=None, the function returns a list of words extracted from input_text where each word starts with an uppercase letter; if join_with is specified, it returns a single string where these words are joined by join_with.

Return type

list | str

Examples:

```
>>> from pyhelpers.text import split_on_uppercase
>>> split_on_uppercase('Network_Waymarks')
['Network', 'Waymarks']
>>> split_on_uppercase('NetworkRailRetainingWall', join_with=' ')
'Network Rail Retaining Wall'
>>> split_on_uppercase('BCRRE_Projects')
['BCRRE', 'Projects']
```

numeral_english_to_arabic

pyhelpers.text.**numeral_english_to_arabic**(*input_text*)

Converts a number written in English words into its equivalent numerical value represented in Arabic numerals.

Parameters

input_text (*str*) – A number expressed in the English language.

Returns

The equivalent Arabic number.

Return type

int

Examples:

```
>>> from pyhelpers.text import numeral_english_to_arabic
>>> numeral_english_to_arabic('one')
1
>>> numeral_english_to_arabic('one hundred and one')
101
>>> numeral_english_to_arabic('a thousand two hundred and three')
1203
>>> numeral_english_to_arabic('200 and five')
205
>>> numeral_english_to_arabic('Two hundred and fivety')  # Two hundred and fifty
ValueError: Illegal word: "fivety"
```

count_words

pyhelpers.text.**count_words**(*input_text*, *lowercase=False*, *ignore_punctuation=False*, *stop_words=None*, ***kwargs*)

Counts the occurrences of each word in the given text.

Parameters

- **input_text** (*str*) – The input text from which words will be counted.
- **lowercase** (*bool*) – Whether to convert the text to lowercase before counting; defaults to `False`.
- **ignore_punctuation** (*bool*) – Whether to exclude punctuation marks from the text; defaults to `False`.
- **stop_words** (*list[str] | bool | None*) – List of words to be excluded from the word count;
  - If `stop_words=None` (default), no words are excluded.
  - If `stop_words=True`, NLTK's built-in stopwords are used.
- **kwargs** – [Optional] Additional parameters for the function nltk.word_tokenize().

Returns

A dictionary where keys are unique words and values are their respective counts.

Return type
dict

Examples:

```
>>> from pyhelpers.text import count_words
>>> input_text = 'This is an apple. That is a pear. Hello world!'
>>> count_words(input_text)
{'This': 1,
 'is': 2,
 'an': 1,
 'apple': 1,
 '.': 2,
 'That': 1,
 'a': 1,
 'pear': 1,
 'Hello': 1,
 'world': 1,
 '!': 1}
>>> count_words(input_text, lowercase=True)
{'this': 1,
 'is': 2,
 'an': 1,
 'apple': 1,
 '.': 2,
 'that': 1,
 'a': 1,
 'pear': 1,
 'hello': 1,
 'world': 1,
 '!': 1}
>>> count_words(input_text, lowercase=True, ignore_punctuation=True)
{'this': 1,
 'is': 2,
 'an': 1,
 'apple': 1,
 'that': 1,
 'a': 1,
 'pear': 1,
 'hello': 1,
 'world': 1}
>>> count_words(input_text, lowercase=True, ignore_punctuation=True, stop_words=['is'])
{'this': 1,
 'an': 1,
 'apple': 1,
 'that': 1,
 'a': 1,
 'pear': 1,
 'hello': 1,
 'world': 1}
>>> count_words(input_text, lowercase=True, ignore_punctuation=True, stop_words=True)
{'apple': 1, 'pear': 1, 'hello': 1, 'world': 1}
```

calculate_idf

pyhelpers.text.**calculate_idf**(*documents*, *lowercase=True*, *ignore_punctuation=True*, *stop_words=None*, *smoothing_factor=1*, *log_base=None*, ***kwargs*)

Calculates Inverse Document Frequency (IDF) for a sequence of textual documents.

Parameters

- **documents** (*Iterable | Sequence*) – A sequence of textual data.
- **lowercase** (*bool*) – Whether to convert the documents to lowercase before calculating IDF; defaults to `True`.
- **ignore_punctuation** (*bool*) – Whether to exclude punctuation marks from the textual data; defaults to `True`.
- **stop_words** (*list[str] | bool | None*) – List of words to be excluded from the IDF calculation;
  - If `stop_words=None` (default), no words are excluded.
  - If `stop_words=True`, NLTK's built-in stopwords are used.
- **smoothing_factor** (*int | float*) – Factor added to the denominator in the IDF formula:
  - For smaller corpora: Use `smoothing_factor=1` (default) to prevent IDF values from becoming too extreme (e.g. zero for terms in all documents). This adjustment ensures more stable IDF values reflecting term rarity.
  - For larger corpora: Use `smoothing_factor=0` for standard IDF calculations, which provides a measure of how terms are distributed across documents. This can generally be sufficient and standard practice.
- **log_base** (*int*) – The logarithm base in the IDF formula; when `log_base=None` (default), use math.e.
- **kwargs** – [Optional] Additional parameters for the function nltk.word_tokenize(); also refer to the function *count_words()*, which calculates term frequencies (TF) for each document.

Returns

Tuple containing:

- Term frequency (TF) of each document as a list of dictionaries, where each dictionary represents TF for one document.
- Inverse document frequency (IDF) as a dictionary, where keys are unique terms across all documents and values are their IDF scores.

Return type

tuple[list[dict], dict]

Examples:

```
>>> from pyhelpers.text import calculate_idf
>>> documents = [
```

(continues on next page)

(continued from previous page)

```
...     'This is an apple.',
...     'That is a pear.',
...     'It is human being.',
...     'Hello world!']
>>> docs_tf, corpus_idf = calculate_idf(documents)
>>> docs_tf
[{'this': 1, 'is': 1, 'an': 1, 'apple': 1},
 {'that': 1, 'is': 1, 'a': 1, 'pear': 1},
 {'it': 1, 'is': 1, 'human': 1, 'being': 1},
 {'hello': 1, 'world': 1}]
>>> corpus_idf
{'this': 0.6931471805599453,
 'is': 0.0,
 'an': 0.6931471805599453,
 'apple': 0.6931471805599453,
 'that': 0.6931471805599453,
 'a': 0.6931471805599453,
 'pear': 0.6931471805599453,
 'it': 0.6931471805599453,
 'human': 0.6931471805599453,
 'being': 0.6931471805599453,
 'hello': 0.6931471805599453,
 'world': 0.6931471805599453}
>>> docs_tf, corpus_idf = calculate_idf(documents, ignore_punctuation=False)
>>> docs_tf
[{'this': 1, 'is': 1, 'an': 1, 'apple': 1, '.': 1},
 {'that': 1, 'is': 1, 'a': 1, 'pear': 1, '.': 1},
 {'it': 1, 'is': 1, 'human': 1, 'being': 1, '.': 1},
 {'hello': 1, 'world': 1, '!': 1}]
>>> corpus_idf
{'this': 0.6931471805599453,
 'is': 0.0,
 'an': 0.6931471805599453,
 'apple': 0.6931471805599453,
 '.': 0.0,
 'that': 0.6931471805599453,
 'a': 0.6931471805599453,
 'pear': 0.6931471805599453,
 'it': 0.6931471805599453,
 'human': 0.6931471805599453,
 'being': 0.6931471805599453,
 'hello': 0.6931471805599453,
 'world': 0.6931471805599453,
 '!': 0.6931471805599453}
```

calculate_tfidf

pyhelpers.text.**calculate_tfidf**(*documents*, ***kwargs*)

Calculates TF-IDF (Term Frequency-Inverse Document Frequency) for the given textual documents.

TF (Term Frequency) measures how frequently a term appears in a document relative to its length. IDF (Inverse Document Frequency) measures how important a term is across the entire corpus of documents.

Parameters

- **documents** (*Iterable | Sequence*) – A sequence of textual data.

- **kwargs** – [Optional] Additional parameters for the function *calculate_idf()*; also refer to *count_words()*.

Returns
TF-IDF values for the input textual data, represented as a dictionary.

Return type
dict

Examples:

```
>>> from pyhelpers.text import calculate_tfidf
>>> documents = [
...     'This is an apple.',
...     'That is a pear.',
...     'It is human being.',
...     'Hello world!']
>>> tfidf = calculate_tfidf(documents)
>>> tfidf
{'this': 0.6931471805599453,
 'is': 0.0,
 'an': 0.6931471805599453,
 'apple': 0.6931471805599453,
 'that': 0.6931471805599453,
 'a': 0.6931471805599453,
 'pear': 0.6931471805599453,
 'it': 0.6931471805599453,
 'human': 0.6931471805599453,
 'being': 0.6931471805599453,
 'hello': 0.6931471805599453,
 'world': 0.6931471805599453}
>>> tfidf = calculate_tfidf(documents, lowercase=False)
>>> tfidf
{'This': 0.6931471805599453,
 'is': 0.0,
 'an': 0.6931471805599453,
 'apple': 0.6931471805599453,
 'That': 0.6931471805599453,
 'a': 0.6931471805599453,
 'pear': 0.6931471805599453,
 'It': 0.6931471805599453,
 'human': 0.6931471805599453,
 'being': 0.6931471805599453,
 'Hello': 0.6931471805599453,
 'world': 0.6931471805599453}
```

4.6.2 Textual data similarity

1anti-flashwhitewhite

euclidean_distance_between_texts(txt1, txt2)	Computes the Euclidean distance between two sentences.
cosine_similarity_between_texts(txt1, txt2)	Calculates the cosine similarity between two sentences.

continues on next page

Table 30 – continued from previous page

find_matched_str(input_str, lookup_list[, ...])	Finds all strings (in a sequence) that match a given string or regex pattern.
find_similar_str(input_str, lookup_list[, ...])	Finds `n` strings that are similar to `input_str` from a sequence of candidates.

euclidean_distance_between_texts

pyhelpers.text.**euclidean_distance_between_texts**(*txt1*, *txt2*)

Computes the Euclidean distance between two sentences.

Parameters

- **txt1** (*str*) – The first text.
- **txt2** (*str*) – The second text.

Returns

The Euclidean distance between the input texts.

Return type

float

Examples:

```
>>> from pyhelpers.text import euclidean_distance_between_texts
>>> txt1, txt2 = 'This is an apple.', 'That is a pear.'
>>> euclidean_distance = euclidean_distance_between_texts(txt1, txt2)
>>> euclidean_distance
2.449489742783178
```

cosine_similarity_between_texts

pyhelpers.text.**cosine_similarity_between_texts**(*txt1*, *txt2*, *cosine_distance=False*)

Calculates the cosine similarity between two sentences.

Parameters

- **txt1** (*str*) – The first text.
- **txt2** (*str*) – The second text.
- **cosine_distance** (*bool*) – Whether to return cosine distance (i.e. `1` - cosine similarity); defaults to `False`.

Returns

The cosine similarity (or distance) between the input texts.

Return type

float

Examples:

```
>>> from pyhelpers.text import cosine_similarity_between_texts
>>> txt1, txt2 = 'This is an apple.', 'That is a pear.'
>>> cos_sim = cosine_similarity_between_texts(txt1, txt2)
>>> cos_sim
np.float64(0.25)
```

(continues on next page)

(continued from previous page)

```
>>> cos_dist = cosine_similarity_between_texts(txt1, txt2, cosine_distance=True)
>>> cos_dist  # 1 - cos_sim
np.float64(0.75)
>>> txt1, txt2 = 'up-to-date', 'Up to date'
>>> cos_sim = cosine_similarity_between_texts(txt1, txt2)
>>> cos_sim  # ≈1
np.float64(1.0000000000000002)
>>> cos_dist = cosine_similarity_between_texts(txt1, txt2, cosine_distance=True)
>>> cos_dist  # ≈0
np.float64(-2.220446049250313e-16)
```

find_matched_str

pyhelpers.text.**find_matched_str**(*input_str*, *lookup_list*, *use_regex=True*)

Finds all strings (in a sequence) that match a given string or regex pattern.

Parameters

- **input_str** (*str*) – The string to match.
- **lookup_list** (*Iterable[str]*) – A sequence of strings for lookup.
- **use_regex** (*bool*) – Whether to treat *text* as a regex pattern; defaults to *True*.

Returns

A generator containing all strings that match `text`.

Return type

Generator | None

Examples:

```
>>> from pyhelpers.text import find_matched_str
>>> lookup_lst = ['abc', 'aapl', 'app', 'ap', 'ape', 'apex', 'apel']
>>> text_ = find_matched_str('apple', lookup_lst)
>>> list(text_)
[]
>>> lookup_lst += ['apple']
>>> lookup_lst
['abc', 'aapl', 'app', 'ap', 'ape', 'apex', 'apel', 'apple']
>>> text_ = find_matched_str('apple', lookup_lst)
>>> list(text_)
['apple']
>>> text_ = find_matched_str(r'app(le)?', lookup_lst)
>>> list(text_)
['app', 'apple']
>>> text_ = find_matched_str('app(le)?', lookup_lst, use_regex=False)
>>> list(text_)  # No match because regex behavior is disabled
[]
```

find_similar_str

pyhelpers.text.**find_similar_str**(*input_str*, *lookup_list*, *n=1*, *ignore_punctuation=True*, *engine='difflib'*, ***kwargs*)

Finds `n` strings that are similar to `input_str` from a sequence of candidates.

Parameters

- **input_str** (*str*) – The string to find similar matches for.

- **lookup_list** (*Iterable*) – A sequence of strings to search for matches.
- **n** (*int | None*) – Number of similar strings to return; defaults to 1; when n=None, the function returns the entire lookup_list sorted by similarity in descending order.
- **ignore_punctuation** (*bool*) – Whether to ignore punctuation in the comparison; defaults to True.
- **engine** (*str | Callable*) – Method for finding similarities; options include:
  - 'difflib' (default), which uses difflib.get_close_matches().
  - 'rapidfuzz' (or 'fuzz'), which uses rapidfuzz.fuzz.QRatio().
- **kwargs** – [Optional] Additional parameters for the chosen engine; for instance, cutoff for 'difflib' and score_cutoff for 'rapidfuzz'.

Returns

A string or list of strings similar to input_str, depending on n and the engine used.

Return type

str | list | None

Note

- By default, the function uses the built-in difflib module.
- When engine='rapidfuzz' (or simply, engine='fuzz'), the function relies on RapidFuzz, which is not a dependency of pyhelpers. Install it separately using pip or conda.

Examples:

```
>>> from pyhelpers.text import find_similar_str
>>> lookup_list = ['Anglia',
...                'East Coast',
...                'East Midlands',
...                'North and East',
...                'London North Western',
...                'Scotland',
...                'South East',
...                'Wales',
...                'Wessex',
...                'Western']
>>> find_similar_str('angle', lookup_list)
'Anglia'
>>> find_similar_str('angle', lookup_list, n=2)
['Anglia', 'Wales']
>>> find_similar_str('angle', lookup_list, engine='fuzz')
'Anglia'
>>> find_similar_str('angle', lookup_list, n=2, engine='fuzz')
['Anglia', 'Wales']
```

(continues on next page)

(continued from previous page)

```
>>> find_similar_str('x', lookup_list) is None
True
>>> find_similar_str('x', lookup_list, cutoff=0.25)
'Wessex'
>>> find_similar_str('x', lookup_list, n=2, cutoff=0.25)
'Wessex'
>>> find_similar_str('x', lookup_list, engine='fuzz')
'Wessex'
>>> find_similar_str('x', lookup_list, n=2, engine='fuzz')
['Wessex', 'Western']
```

4.7 dbms

Communication with databases.

The current release includes classes for PostgreSQL *and* Microsoft SQL Server.

4.7.1 Databases

1anti-flashwhitewhite

PostgreSQL([host, port, username, password, ...])	A class for basic communication with PostgreSQL databases.
MSSQL([host, port, username, password, ...])	A class for basic communication with Microsoft SQL Server databases.

PostgreSQL

class pyhelpers.dbms.**PostgreSQL**(*host=None*, *port=None*, *username=None*, *password=None*, *database_name=None*, *confirm_db_creation=False*, *verbose=False*, *raise_error=False*)

A class for basic communication with PostgreSQL databases.

Parameters

- **host** (*str | None*) – Host name/address of a PostgreSQL server, e.g. `'localhost'` or `'127.0.0.1'`. If `host=None`, it defaults to `'localhost'`.
- **port** (*int | None*) – Listening port used by PostgreSQL. If `port=None`, it defaults to `5432`.
- **username** (*str | None*) – Username of the PostgreSQL server. If `username=None`, it defaults to `'postgres'`.
- **password** (*str | int | None*) – User password. If `password=None`, it must be manually entered to connect to the PostgreSQL server.
- **database_name** (*str | None*) – Name of the database. If `database_name=None`, it defaults to `'postgres'`.
- **confirm_db_creation** (*bool*) – Whether to prompt for confirmation before creating a new database if the specified database does not exist; defaults to `False`.

- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Variables

- **host** (*str*) – Host name/address.
- **port** (*int*) – Listening port used by PostgreSQL.
- **username** (*str*) – Username.
- **database_name** (*str*) – Name of the database.
- **credentials** (*dict*) – Basic information about the server/database being connected.
- **address** (*str*) – Representation of the database address.
- **engine** (*sqlalchemy.engine.Engine*) – SQLAlchemy connectable engine to a PostgreSQL server; see also [DBMS-PS-2].

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> # Connect the default database 'postgres'
>>> postgres = PostgreSQL(verbose=True)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> postgres.address
'postgres:***@localhost:5432/postgres'
>>> # Connect a database 'testdb' (which will be created if it does not exist)
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.address
'postgres:***@localhost:5432/testdb'
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.address
'postgres:***@localhost:5432/postgres'
```

Define a proxy object that inherits from this class:

```
>>> class ExampleProxyObj(PostgreSQL):
...     def __init__(self, **kwargs):
...         super().__init__(**kwargs)
>>> example_proxy = ExampleProxyObj(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> example_proxy.address
'postgres:***@localhost:5432/testdb'
```

(continues on next page)

(continued from previous page)

```
>>> example_proxy.database_name
'testdb'
>>> example_proxy.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> example_proxy.database_name
'postgres'
```

Attributes

1anti-flashwhitewhite

BUILTIN_SCHEMAS	Built-in schemas of PostgreSQL.
DEFAULT_DATABASE	Default database name (usually created during PostgreSQL installation).
DEFAULT_DIALECT	Default dialect used by SQLAlchemy to communicate with PostgreSQL; see also [DBMS-PS-1].
DEFAULT_DRIVER	Default name of the database driver.
DEFAULT_HOST	Default host name/address (typically *localhost*).
DEFAULT_PORT	Default listening port used by PostgreSQL.
DEFAULT_SCHEMA	Default schema name created during PostgreSQL installation.
DEFAULT_USERNAME	Default username.

PostgreSQL.BUILTIN_SCHEMAS

PostgreSQL.**BUILTIN_SCHEMAS: set = {'information_schema'}**

Built-in schemas of PostgreSQL.

PostgreSQL.DEFAULT_DATABASE

PostgreSQL.**DEFAULT_DATABASE: str = 'postgres'**

Default database name (usually created during PostgreSQL installation).

PostgreSQL.DEFAULT_DIALECT

PostgreSQL.**DEFAULT_DIALECT: str = 'postgresql'**

Default dialect used by SQLAlchemy to communicate with PostgreSQL; see also [DBMS-PS-1].

PostgreSQL.DEFAULT_DRIVER

PostgreSQL.**DEFAULT_DRIVER: str = 'psycopg2'**

Default name of the database driver.

PostgreSQL.DEFAULT_HOST

PostgreSQL.**DEFAULT_HOST: str = 'localhost'**
Default host name/address (typically *localhost*).

PostgreSQL.DEFAULT_PORT

PostgreSQL.**DEFAULT_PORT: str = 5432**
Default listening port used by PostgreSQL.

PostgreSQL.DEFAULT_SCHEMA

PostgreSQL.**DEFAULT_SCHEMA: str = 'public'**
Default schema name created during PostgreSQL installation.

PostgreSQL.DEFAULT_USERNAME

PostgreSQL.**DEFAULT_USERNAME: str = 'postgres'**
Default username.

Methods

1anti-flashwhitewhite

add_primary_keys(primary_keys, table_name[, ...])	Adds a primary key or multiple primary keys to a table.
alter_table_schema(table_name, schema_name, ...)	Moves a table from one schema to another within the currently-connected database.
connect_database([database_name, verbose, ...])	Establishes a connection to a database.
create_database(database_name[, verbose])	Creates a database.
create_schema(schema_name[, verbose, ...])	Creates a schema.
create_table(table_name, column_specs[, ...])	Creates a table.
database_exists([database_name])	Checks if a specified database exists.
disconnect_all_others()	Terminates connections to all databases except the currently-connected one.
disconnect_database([database_name, ...])	Disconnects from a database.
drop_database([database_name, ...])	Deletes/drops a database.
drop_schema(schema_names[, ...])	Deletes/drops one or multiple schemas.
drop_table(table_name[, schema_name, ...])	Deletes/drops a specified table.
get_column_dtype(table_name[, column_names, ...])	Retrieves information about data types of all or specific columns of a table.
get_column_info(table_name[, schema_name, ...])	Retrieves information about columns of a table.
get_column_names(table_name[, schema_name])	Retrieves column names of a table.
get_database_names([names_only])	Retrieves the names of all existing databases.
get_database_size([database_name])	Retrieves the size of a database.
get_primary_keys(table_name[, schema_name, ...])	Retrieves the primary keys of a table.

continues on next page

Table 33 – continued from previous page

get_schema_info([names_only, include_all, ...])	Retrieves information about existing schemas.
get_table_names([schema_name, verbose])	Retrieves the names of all tables in a schema.
import_data(data, table_name[, schema_name, ...])	Imports tabular data into the database.
null_text_to_empty_string(table_name[, ...])	Converts null values (in text columns) to empty strings.
psql_insert_copy(sql_table, sql_db_engine, ...)	Callable function using *PostgreSQL* `COPY` clause for executing data insertion.
read_sql_query(sql_query[, method, ...])	Reads table data by executing a SQL query (recommended for large tables).
read_table(table_name[, schema_name, ...])	Reads data from a specified table.
schema_exists(schema_name)	Checks if a schema exists.
table_exists(table_name[, schema_name])	Checks if a table exists.
validate_column_names(table_name[, ...])	Validates column names for a query statement.

PostgreSQL.add_primary_keys

PostgreSQL.**add_primary_keys**(*primary_keys*, *table_name*, *schema_name=None*)

Adds a primary key or multiple primary keys to a table.

Parameters

- **primary_keys** (*str | list | None*) – (List of) primary key(s) to be added.
- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; defaults to `None`.

See also

- Examples for the method *get_primary_keys()*.

PostgreSQL.alter_table_schema

PostgreSQL.**alter_table_schema**(*table_name*, *schema_name*, *new_schema_name*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Moves a table from one schema to another within the currently-connected database.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str*) – Name of the current schema containing the table.
- **new_schema_name** (*str*) – Name of the new schema to move the table to.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to `True`.

- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Create a new table named "test_table" in the schema "testdb"
>>> new_tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(table_name=new_tbl_name, column_specs=col_spec, verbose=True)
Creating a table: "public"."test_table" ... Done.
>>> # Create a new schema "test_schema"
>>> testdb.create_schema(schema_name='test_schema', verbose=True)
Creating a schema: "test_schema" ... Done.
>>> # Move the table "public"."test_table" to the schema "test_schema"
>>> testdb.alter_table_schema(
...     table_name='test_table', schema_name='public', new_schema_name='test_schema',
...     verbose=True)
To move the table "test_table" from the schema "public" to "test_schema"
? [No]|Yes: yes
Moving "public"."test_table" to "test_schema" ... Done.
>>> tbl_names = testdb.get_table_names(schema_name='test_schema')
>>> tbl_names
{'test_schema': ['test_table']}
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.connect_database

`PostgreSQL.`**`connect_database`**(*database_name=None*, *verbose=False*, *raise_error=False*)

Establishes a connection to a database.

Parameters

- **database_name** (*str | None*) – Name of the database. If `database_name=None` (default), the database name must be input manually.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
```

(continues on next page)

(continued from previous page)

```
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.connect_database(verbose=True)
Being connected with postgres:***@localhost:5432/testdb.
>>> testdb.connect_database(database_name='postgres', verbose=True)
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> testdb.database_name
'postgres'
>>> testdb.connect_database(database_name='testdb', verbose=True)
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.database_name
'postgres'
```

PostgreSQL.create_database

PostgreSQL.**create_database**(*database_name*, *verbose=False*)

Creates a database.

Parameters

- **database_name** (*str*) – Name of the database to be created.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.create_database(database_name='testdb1', verbose=True)
Creating a database: "testdb1" ... Done.
>>> testdb.database_name
'testdb1'
>>> # Delete the database "testdb1"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb1" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb1" ... Done.
>>> testdb.database_name
'postgres'
>>> # Delete the database "testdb"
>>> testdb.drop_database(database_name='testdb', verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.database_name
```

(continues on next page)

(continued from previous page)

```
'postgres'
```

PostgreSQL.create_schema

PostgreSQL.**create_schema**(*schema_name*, *verbose=False*, *raise_error=False*)

Creates a schema.

Parameters

- **schema_name** (*str*) – Name of the schema to be created.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> test_schema_name = 'test_schema'
>>> testdb.create_schema(schema_name=test_schema_name, verbose=True)
Creating a schema: "test_schema" ... Done.
>>> testdb.schema_exists(schema_name=test_schema_name)
True
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.create_table

PostgreSQL.**create_table**(*table_name*, *column_specs*, *schema_name=None*, *verbose=False*, *raise_error=False*)

Creates a table.

Parameters

- **table_name** (*str*) – Name of the table to be created.
- **column_specs** (*str*) – Specifications for each column of the table.
- **schema_name** (*str | None*) – Name of the schema; if schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public').
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Create a new table named 'test_table'
>>> tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(table_name=tbl_name, column_specs=col_spec, verbose=True)
Creating a table "public"."test_table" ... Done.
>>> testdb.table_exists(table_name=tbl_name)
True
>>> # Get information about all columns of the table "public"."test_table"
>>> test_tbl_col_info = testdb.get_column_info(table_name=tbl_name, as_dict=False)
>>> test_tbl_col_info.head()
                    column_0    column_1
table_catalog         testdb      testdb
table_schema          public      public
table_name        test_table  test_table
column_name       col_name_1  col_name_2
ordinal_position           1           2
>>> # Get column names of the table "public"."test_table"
>>> test_tbl_col_names = testdb.get_column_names(table_name=tbl_name)
>>> test_tbl_col_names
['col_name_1', 'col_name_2']
>>> # Get data types of all columns of the table
>>> test_tbl_dtypes = testdb.get_column_dtype(table_name=tbl_name)
>>> test_tbl_dtypes
{'col_name_1': 'integer', 'col_name_2': 'text'}
>>> testdb.validate_column_names(table_name=tbl_name)
'"col_name_1", "col_name_2"'
>>> # Drop the table "public"."test_table"
>>> testdb.drop_table(table_name=tbl_name, verbose=True)
To drop the table "public"."test_table" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "public"."test_table" ... Done.
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.database_exists

PostgreSQL.**database_exists**(*database_name=None*)

Checks if a specified database exists.

Parameters

database_name (*str | None*) – Name of the database to check; defaults to None.

Returns

True if the database exists, otherwise False.

Return type

bool

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Check whether the database "testdb" exists now
>>> testdb.database_exists(database_name='testdb')  # testdb.database_exists()
True
>>> testdb.database_name
'testdb'
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> # Check again whether the database "testdb" still exists now
>>> testdb.database_exists(database_name='testdb')
False
>>> testdb.database_name
'postgres'
```

PostgreSQL.disconnect_all_others

PostgreSQL.**disconnect_all_others**()

Terminates connections to all databases except the currently-connected one.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.disconnect_all_others()
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.disconnect_database

PostgreSQL.**disconnect_database**(*database_name=None*, *verbose=False*, *raise_error=False*)

Disconnects from a database.

See also [DBMS-PS-DD-1].

Parameters

- **database_name** (*str | None*) – Name of the database to disconnect from. If `database_name=None` (default), disconnect from the current database.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if

raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Creating a database: "testdb" ... Done.
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.disconnect_database()
>>> testdb.database_name
'postgres'
>>> # Delete the database "testdb"
>>> testdb.drop_database(database_name='testdb', verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.drop_database

PostgreSQL.**drop_database**(*database_name=None*, *confirmation_required=True*, *verbose=False*)

Deletes/drops a database.

Parameters

- **database_name** (*str | None*) – Name of the database to be dropped. If None (default), drop the currently connected database.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
>>> testdb.database_exists(database_name='testdb')
False
>>> testdb.drop_database(database_name='testdb', verbose=True)
The database "testdb" does not exist.
>>> testdb.database_name
'postgres'
```

PostgreSQL.drop_schema

PostgreSQL.**drop_schema**(*schema_names*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Deletes/drops one or multiple schemas.

Parameters

- **schema_names** (*str | Iterable[str]*) – Name of one schema or names of multiple schemas to be dropped.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> new_schema_names = ['points', 'lines', 'polygons']
>>> for new_schema in new_schema_names:
...     testdb.create_schema(new_schema, verbose=True)
Creating a schema: "points" ... Done.
Creating a schema: "lines" ... Done.
Creating a schema: "polygons" ... Done.
>>> new_schema_names_ = ['test_schema']
>>> testdb.drop_schema(new_schema_names + new_schema_names_, verbose=True)
To drop the following schemas from postgres:***@localhost:5432/testdb:
  "points"
  "lines"
  "polygons"
  "test_schema"
? [No]|Yes: yes
Dropping ...
  "points" ... Done.
  "lines" ... Done.
  "polygons" ... Done.
  "test_schema" (does not exist.)
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.drop_table

PostgreSQL.**drop_table**(*table_name*, *schema_name=None*, *confirmation_required=True*, *verbose=False*, *indent=0*, *raise_error=False*)

Deletes/drops a specified table.

Parameters

- **table_name** (*str*) – Name of the table to be deleted.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None`, it defaults to *DEFAULT_SCHEMA* (i.e., `'public'`).

- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **indent** (*int | str*) – Indentation level; if an integer, represents the number of spaces; If a string, used as the indentation character (e.g. `'\t'`); defaults to `0` (no space).
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

See also

- Examples for the method *create_table()*.

PostgreSQL.get_column_dtype

`PostgreSQL.`**`get_column_dtype`**(*table_name*, *column_names=None*, *schema_name=None*)

Retrieves information about data types of all or specific columns of a table.

Parameters

- **table_name** (*str*) – Name of the table.
- **column_names** (*str | list | None*) – (List of) column name(s); if `column_names=None` (default), data types of all available columns are retrieved.
- **schema_name** – Name of the schema; if `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'public'`).

Returns

Data types of all or specific columns of a table.

Return type

dict | None

See also

- Examples for the method *create_table()*.

PostgreSQL.get_column_info

`PostgreSQL.`**`get_column_info`**(*table_name*, *schema_name=None*, *as_dict=True*)

Retrieves information about columns of a table.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'public'`).

- **as_dict** (*bool*) – Whether to return the column information as a dictionary; defaults to `True`.

Returns
Information about all columns of the given table.

Return type
pandas.DataFrame | dict

See also

- Examples for the method *create_table()*.

PostgreSQL.get_column_names

`PostgreSQL.`**`get_column_names`**(*table_name*, *schema_name=None*)
Retrieves column names of a table.

Parameters

- **table_name** (*str*) – Name of the table to retrieve column names from.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to `None`.

Returns
List of column names in the specified table in the currently-connected database.

Return type
list

See also

- Examples for the method *create_table()*.

PostgreSQL.get_database_names

`PostgreSQL.`**`get_database_names`**(*names_only=True*)
Retrieves the names of all existing databases.

Parameters
names_only (*bool*) – Whether to return only the names of the databases; defaults to `True`.

Returns
A list of database names if `names_only` is `True`; otherwise, a dataframe containing detailed information.

Return type
list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> # Connect the default database 'postgres'
>>> postgres = PostgreSQL(verbose=True)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/postgres ... Successfully.
>>> isinstance(postgres.get_database_names(), list)
True
>>> 'postgres' in postgres.get_database_names()
True
```

PostgreSQL.get_database_size

PostgreSQL.**get_database_size**(*database_name=None*)

Retrieves the size of a database.

Parameters

database_name (*str | None*) – Name of the database. If database_name=None (default), it retrieves the size of the currently-connected database.

Returns

Size of the database.

Return type

str | None

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.get_database_size()
'7900 kB'
>>> testdb.DEFAULT_DATABASE
'postgres'
>>> testdb.get_database_size(database_name=testdb.DEFAULT_DATABASE)
'7828 kB'
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.get_primary_keys

PostgreSQL.**get_primary_keys**(*table_name*, *schema_name=None*, *names_only=True*)

Retrieves the primary keys of a table.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; defaults to None.
- **names_only** (*bool*) – Whether to return only the names of the primary keys; defaults to True.

Returns

Primary key(s) of the given table.

Return type

list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> dat = example_dataframe()
>>> dat
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> tbl_name = 'test_table'
>>> testdb.import_data(data=dat, table_name=tbl_name, index=True, verbose=True)
To import data into "public"."test_table" at postgres:***@localhost:5432/postgres
? [No]|Yes: yes
>>> pri_keys = testdb.get_primary_keys(table_name=tbl_name)
>>> pri_keys
[]
>>> testdb.add_primary_keys(primary_keys='City', table_name=tbl_name)
```

The "*test_table*" is illustrated in Figure 16 below:

	City [PK] text	Longitude double precision	Latitude double precision
1	Birmingham	-1.9026911	52.4796992
2	Leeds	-1.5437941	53.7974185
3	London	-0.1276474	51.5073219
4	Manchester	-2.2451148	53.4794892

Figure 16: The table "*test_table*" (with a primary key) in the database.

```
>>> pri_keys = testdb.get_primary_keys(table_name=tbl_name)
>>> pri_keys
['City']
>>> pri_keys = testdb.get_primary_keys(table_name=tbl_name, names_only=False)
>>> pri_keys
  key_column data_type
0       City      text
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
```

(continues on next page)

(continued from previous page)

```
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.get_schema_info

PostgreSQL.**get_schema_info**(*names_only=True*, *include_all=False*, *column_names=None*, *verbose=False*)

Retrieves information about existing schemas.

Parameters

- **names_only** (*bool*) – Whether to return only the names of the schemas; defaults to `True`.
- **include_all** (*bool*) – Whether to list all available schemas; defaults to `False`.
- **column_names** (*list | None*) – Column names for the returned dataframe if `names_only=False`; defaults to `['schema_name', 'schema_id', 'role']` if `column_names=None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

Names of schemas or a dataframe with schema information if requested.

Return type

list | pandas.DataFrame | None

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.get_schema_info()
['public']
>>> testdb.get_schema_info(names_only=False, include_all=True)
          schema_name       schema_owner    oid  nspowner
0              public  pg_database_owner   2200      6171
1            pg_toast           postgres     99        10
2          pg_catalog           postgres     11        10
3  information_schema           postgres  13183        10
>>> testdb.create_schema(schema_name='test_schema', verbose=True)
Creating a schema: "test_schema" ... Done.
>>> testdb.get_schema_info()
['public', 'test_schema']
>>> testdb.drop_schema(schema_names=['public', 'test_schema'], verbose=True)
To drop the following schemas from postgres:***@localhost:5432/testdb:
  "public"
  "test_schema"
? [No]|Yes: yes
Dropping ...
```

(continues on next page)

(continued from previous page)

```
  "public" ... Done.
  "test_schema" ... Done.
>>> testdb.get_schema_info() is None
True
>>> testdb.get_schema_info(verbose=True)
No schema exists in the currently-connected database "testdb".
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.get_table_names

PostgreSQL.**get_table_names**(*schema_name=None*, *verbose=False*)

Retrieves the names of all tables in a schema.

Parameters

- **schema_name** (*str | list | None*) – Name of the schema; if schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public').
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Returns

Table names of the specified schema(s) schema_name.

Return type

dict

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> tbl_names = testdb.get_table_names()
>>> tbl_names
{'public': []}
>>> tbl_names = testdb.get_table_names(schema_name='testdb', verbose=True)
The schema "testdb" does not exist.
>>> tbl_names is None
True
>>> # Create a new table named "test_table" in the schema "testdb"
>>> new_tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(table_name=new_tbl_name, column_specs=col_spec, verbose=True)
Creating a table: "public"."test_table" ... Done.
>>> tbl_names = testdb.get_table_names(schema_name='public')
>>> tbl_names
{'public': ['test_table']}
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.import_data

PostgreSQL.**import_data**(*data*, *table_name*, *schema_name=None*, *if_exists='fail'*, *force_replace=False*, *chunk_size=None*, *dtype=None*, *method='multi'*, *index=False*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Imports tabular data into the database.

See also [DBMS-PS-ID-1] and [DBMS-PS-ID-2].

Parameters

- **data** (*pandas.DataFrame | pandas.io.parsers.TextFileReader | list | tuple*) – Tabular data to be imported into the database.
- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'public'`).
- **if_exists** (*str*) – Action to take if the table already exists. Options are `'replace'`, `'append'` or `'fail'` (default).
- **force_replace** (*bool*) – Whether to force replace an existing table; defaults to `False`.
- **chunk_size** (*int | None*) – Number of rows in each batch to be written at a time; defaults to `None`.
- **dtype** (*dict | None*) – Data types for columns; defaults to `None`.
- **method** (*str | None | Callable*) – Method for SQL insertion clause; defaults to `'multi'`.
  - `None`: Uses standard SQL `INSERT` clause (one per row).
  - `'multi'`: Passes multiple values in a single `INSERT` clause.
  - Callable (e.g. `PostgreSQL.psql_insert_copy`) with signature `(pd_table, conn, keys, data_iter)`.
- **index** (*bool*) – Whether to include the index as a column in the table.
- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the method pandas.DataFrame.to_sql().

See also

- Examples for the method *read_sql_query()*.

PostgreSQL.null_text_to_empty_string

PostgreSQL.**null_text_to_empty_string**(*table_name*, *column_names=None*, *schema_name=None*)

Converts null values (in text columns) to empty strings.

Parameters

- **table_name** (*str*) – Name of the table.
- **column_names** (*str | list | None*) – (List of) column name(s) to convert null values to empty strings; if column_names=None (default), all available columns are included.
- **schema_name** (*str | None*) – Name of the schema; defaults to None.

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> dat = example_dataframe()
>>> dat['Longitude'] = dat['Longitude'].astype(str)
>>> dat.loc['London', 'Longitude'] = None
>>> dat
            Longitude   Latitude
City
London            None  51.507322
Birmingham  -1.9026911  52.479699
Manchester  -2.2451148  53.479489
Leeds       -1.5437941  53.797418
>>> tbl_name = 'test_table'
>>> testdb.import_data(data=dat, table_name=tbl_name, index=True, verbose=True)
To import data into "public"."test_table" at postgres:***@localhost:5432/postgres
? [No]|Yes: yes
>>> testdb.table_exists(table_name=tbl_name)
True
>>> testdb.get_column_dtype(table_name=tbl_name)
{'City': 'text', 'Longitude': 'text', 'Latitude': 'double precision'}
>>> dat_ = testdb.read_table(tbl_name)
>>> dat_.loc[0, 'Longitude'] is None
True
```

	City text	Longitude text	Latitude double precision
1	London	[null]	51.5073219
2	Birmingham	-1.9026911	52.4796992
3	Manchester	-2.2451148	53.4794892
4	Leeds	-1.5437941	53.7974185

Figure 17: The table "*test_table*" in the database "*testdb*".

```
>>> # Replace the 'null' value with an empty string
>>> testdb.null_text_to_empty_string(table_name=tbl_name)
>>> dat_ = testdb.read_table(tbl_name)
>>> dat_.loc[0, 'Longitude']
''
```

	City text	Longitude text	Latitude double precision
1	London		51.5073219
2	Birmingham	-1.9026911	52.4796992
3	Manchester	-2.2451148	53.4794892
4	Leeds	-1.5437941	53.7974185

Figure 18: The table "*test_table*" in the database "*testdb*" (after converting 'null' to empty string).

```
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.psql_insert_copy

static PostgreSQL.**psql_insert_copy**(*sql_table*, *sql_db_engine*, *column_names*, *data_iter*)

Callable function using *PostgreSQL* `COPY` clause for executing data insertion.

Parameters

- **sql_table** (*pandas.io.sql.SQLTable*) – Object that represents the table to insert into.
- **sql_db_engine** (*sqlalchemy.engine.Connection | sqlalchemy.engine.Engine*) – Object that represents the database engine or connection.
- **column_names** (*list[str]*) – List of column names to insert data into.
- **data_iter** (*Iterable*) – Iterable that iterates over the values to be inserted.

> **Note**
>
> This function is modified from the source code available at [DBMS-PS-PIC-1].

PostgreSQL.read_sql_query

PostgreSQL.**read_sql_query**(*sql_query*, *method='tempfile'*, *max_size_spooled=1*, *delimiter=','*, *tempfile_kwargs=None*, *stringio_kwargs=None*, ***kwargs*)

Reads table data by executing a SQL query (recommended for large tables).

See also [DBMS-PS-RSQ-1], [DBMS-PS-RSQ-2] and [DBMS-PS-RSQ-3].

Parameters

- **sql_query** (*str*) – SQL query to be executed.
- **method** (*str*) – Method to be used for buffering temporary data.
  - `'tempfile'` (default): Uses tempfile.TemporaryFile().
  - `'stringio'`: Uses io.StringIO().
  - `'spooled'`: Uses tempfile.SpooledTemporaryFile().
- **max_size_spooled** (*int | float*) – Maximum size of the file generated via tempfile.SpooledTemporaryFile(); defaults to `1` (in gigabyte).
- **delimiter** (*str*) – Delimiter used in data; defaults to `','`.
- **tempfile_kwargs** – [Optional] Additional parameters for tempfile.TemporaryFile() or tempfile.SpooledTemporaryFile(); defaults to `None`.
- **stringio_kwargs** – [Optional] Additional parameters for io.StringIO(), e.g. `initial_value`; defaults to `None`.
- **kwargs** – [Optional] Additional parameters for the function pandas.read_csv().

Returns

Data queried by the statement `sql_query`.

Return type

pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Create an example dataframe
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
```

(continues on next page)

(continued from previous page)

```
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> table = 'England'
>>> schema = 'points'
>>> # Import the data into a table named "points"."England"
>>> testdb.import_data(example_df, table, schema, index=True, verbose=2)
Import data into "points"."England" at postgres:***@localhost:5432/testdb?
 [No]|Yes: yes
Creating a schema: "points" ... Done.
Importing the data into "points"."England" ... Done.
```

The table *"points"."England"* is illustrated in Figure 16 below:

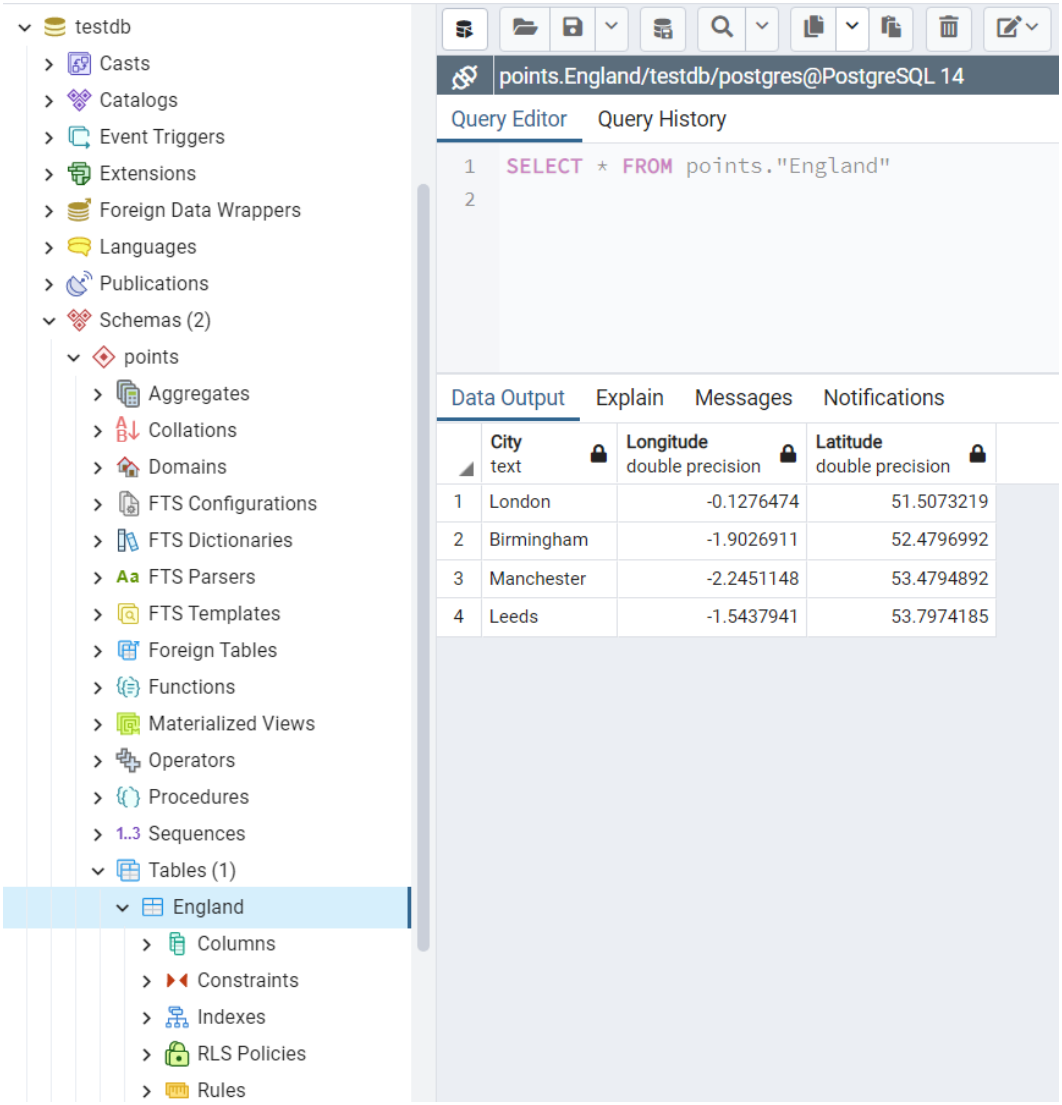

Figure 19: The table *"points"."England"* in the database *"testdb"*.

```
>>> res = testdb.table_exists(table_name=table, schema_name=schema)
>>> print(f'The table "{schema}"."{table}" exists? {res}.')
The table "points"."England" exists? True.
>>> # Retrieve the data using the method .read_table()
>>> example_df_ret = testdb.read_table(table, schema_name=schema, index_col='City')
>>> example_df_ret
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> # Alternatively, read the data by a SQL query statement
>>> sql_qry = f'SELECT * FROM "{schema}"."{table}"'
>>> example_df_ret_alt = testdb.read_sql_query(sql_query=sql_qry, index_col='City')
>>> example_df_ret_alt
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> example_df_ret.equals(example_df_ret_alt)
True
>>> # Delete the table "points"."England"
>>> testdb.drop_table(table_name=table, schema_name=schema, verbose=True)
To drop the table "points"."England" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "points"."England" ... Done.
>>> # Delete the schema "points"
>>> testdb.drop_schema(schema_names=schema, verbose=True)
To drop the schema "points" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "points" ... Done.
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

Aside: a brief example of using the parameter `params` for pandas.read_sql

```
>>> import datetime
>>> import pandas as pd
>>> sql_qry = 'SELECT * FROM "table_name" '
...           'WHERE "timestamp_column_name" BETWEEN %(ts_start)s AND %(ts_end)s'
>>> params = {'d_start': datetime.datetime.today(), 'd_end': datetime.datetime.today()}
>>> data_frame = pd.read_sql(sql=sql_qry, con=testdb.engine, params=params)
```

PostgreSQL.read_table

PostgreSQL.**read_table**(*table_name*, *schema_name=None*, *conditions=None*, *chunk_size=None*, *sorted_by=None*, ***kwargs*)

Reads data from a specified table.

See also [DBMS-PS-RT-1].

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; if `schema_name=None`, it defaults to *DEFAULT_SCHEMA* (i.e., `'public'`).
- **conditions** (*str | None*) – SQL conditions to filter rows; defaults to `None`.
- **chunk_size** (*int | None*) – Number of rows to fetch per iteration; defaults to `None`.
- **sorted_by** (*str | None*) – Name(s) of column(s) by which the retrieved data is sorted; defaults to `None`.
- **kwargs** – [Optional] Additional parameters for the method *read_sql_query()* or the function pandas.read_sql().

Returns
Data of the specified table.

Return type
pandas.DataFrame

See also

- Examples for the method *read_sql_query()*.

PostgreSQL.schema_exists

PostgreSQL.**schema_exists**(*schema_name*)
Checks if a schema exists.

Parameters
schema_name (*str*) – Name of the schema to check.

Returns
`True` if the schema exists, otherwise `False`.

Return type
bool

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.schema_exists('public')
True
>>> testdb.schema_exists('test_schema')  # (if the schema 'test_schema' does not exist)
False
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.table_exists

PostgreSQL.**table_exists**(*table_name*, *schema_name=None*)

Checks if a table exists.

Parameters

- **table_name** (*str*) – Name of the table to check.
- **schema_name** (*str | None*) – Name of the schema; if schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public').

Returns

True if the table exists in the currently-connected database, otherwise False.

Return type

bool

Examples:

```
>>> from pyhelpers.dbms import PostgreSQL
>>> testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> tbl_name = 'points'
>>> testdb.table_exists(table_name=tbl_name)  # (if 'public.points' does not exist)
False
>>> testdb.create_table(table_name=tbl_name, column_specs='column_0 INT', verbose=1)
Creating a table: "public"."points" ... Done.
>>> testdb.table_exists(table_name=tbl_name)
True
>>> testdb.drop_table(table_name=tbl_name, verbose=True)
To drop the table "public"."points" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "public"."points" ... Done.
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.validate_column_names

PostgreSQL.**validate_column_names**(*table_name*, *schema_name=None*, *column_names=None*)

Validates column names for a query statement.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema; defaults to None.
- **column_names** (*str | list | tuple | None*) – Column name(s) for a dataframe.

Returns

Validated column names formatted for a PostgreSQL query statement.

Return type

str

See also

- Examples for the method `create_table()`.

MSSQL

class pyhelpers.dbms.**MSSQL**(*host=None*, *port=None*, *username=None*, *password=None*, *database_name=None*, *confirm_db_creation=False*, *verbose=False*, *raise_error=False*)

A class for basic communication with Microsoft SQL Server databases.

Parameters

- **host** (*str | None*) – Name or IP address of the SQL Server, e.g. `'localhost'` or `'127.0.0.1'`; defaults to `'localhost'` if not specified.
- **port** (*int | str | None*) – Listening port of the SQL Server; defaults to `1433` if not specified (default by installation of the SQL Server).
- **username** (*str | None*) – Username for authentication; if not provided, Windows Authentication is used.
- **password** (*str | int | None*) – Password for the specified username; required for non-Windows Authentication.
- **database_name** (*str | None*) – Name of the initial database to connect to; defaults to `'master'` if not specified.
- **confirm_db_creation** (*bool*) – Whether to prompt for confirmation before creating a new database (if the specified database does not exist); defaults to `False`.
- **verbose** (*bool | int*) – Whether to print connection and operation details to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Variables

- **host** (*str*) – Name or IP address of the SQL Server.
- **port** (*str*) – Listening port of the SQL Server.
- **username** (*str*) – Username used for authentication.
- **database_name** (*str*) – Name of the connected database.
- **credentials** (*dict*) – Contains basic information about the server and database.
- **auth** (*str | None*) – Authentication method used for connection.
- **address** (*str*) – String representation of the database connection address.
- **engine** (*sqlalchemy.engine.Engine*) – SQLAlchemy engine connected to the SQL Server; see also [DBMS-MS-3].

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.address
'<server_name>@localhost:1433/master'
>>> testdb = MSSQL(database_name='testdb', verbose=True)
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)
Drop the database [testdb] from <server_name>@localhost:1433?
 [No]|Yes: yes
Dropping [testdb] ... Done.
```

Attributes

1anti-flashwhitewhite

BUILTIN_SCHEMAS	Names of built-in schemas of Microsoft SQL Server.
DEFAULT_DATABASE	Default database name.
DEFAULT_DIALECT	Default dialect.
DEFAULT_DRIVER	Default name of database driver.
DEFAULT_HOST	Default host (server name).
DEFAULT_ODBC_DRIVER	Default ODBC driver.
DEFAULT_PORT	Default listening port used by Microsoft SQL Server.
DEFAULT_SCHEMA	Default schema name.
DEFAULT_USERNAME	Default username.

MSSQL.BUILTIN_SCHEMAS

MSSQL.**BUILTIN_SCHEMAS: set = {'INFORMATION_SCHEMA', 'db_accessadmin', 'db_backupoperator', 'db_datareader', 'db_datawriter', 'db_ddladmin', 'db_denydatareader', 'db_denydatawriter', 'db_owner', 'db_securityadmin', 'dbo', 'guest', 'sys'}**

Names of built-in schemas of Microsoft SQL Server.

MSSQL.DEFAULT_DATABASE

MSSQL.**DEFAULT_DATABASE: str = 'master'**

Default database name.

MSSQL.DEFAULT_DIALECT

MSSQL.**DEFAULT_DIALECT: str = 'mssql'**

Default dialect. The dialect that SQLAlchemy uses to communicate with Microsoft SQL Server; see also [DBMS-MS-1].

MSSQL.DEFAULT_DRIVER

MSSQL.**DEFAULT_DRIVER: str = 'pyodbc'**

Default name of database driver. See also [DBMS-MS-2].

MSSQL.DEFAULT_HOST

MSSQL.**DEFAULT_HOST: str = 'localhost'**

Default host (server name). Alternatively, `os.environ['COMPUTERNAME']`.

MSSQL.DEFAULT_ODBC_DRIVER

MSSQL.**DEFAULT_ODBC_DRIVER: str = 'ODBC Driver 17 for SQL Server'**

Default ODBC driver.

MSSQL.DEFAULT_PORT

MSSQL.**DEFAULT_PORT: str | int = 1433**

Default listening port used by Microsoft SQL Server.

MSSQL.DEFAULT_SCHEMA

MSSQL.**DEFAULT_SCHEMA: str = 'dbo'**

Default schema name.

MSSQL.DEFAULT_USERNAME

MSSQL.**DEFAULT_USERNAME: str = 'sa'**

Default username.

Methods

1anti-flashwhitewhite

add_primary_key(column_name, table_name[, ...])	Add a primary key constraint to a table.
connect_database([database_name, verbose, ...])	Establish a connection to a database.
create_connection([database_name, mode])	Create a SQLAlchemy connection to a Microsoft SQL Server database.
create_cursor([database_name])	Create a pyodbc cursor.
create_database(database_name[, verbose])	Create a database.
create_engine([database_name, auth, password])	Create a SQLAlchemy connectable engine for connecting to a SQL Server database.
create_schema(schema_name[, verbose, ...])	Create a schema.
create_table(table_name, column_specs[, ...])	Create a table with specified columns.
database_exists([database_name])	Check whether a database exists.
disconnect_database([database_name, ...])	Disconnect from a database.

continues on next page

Table 35 – continued from previous page

drop_database([database_name, ...])	Delete/drop a database.
drop_schema(schema_names[, ...])	Delete/drop one or multiple schemas.
drop_table(table_name[, schema_name, ...])	Delete/drop a table.
get_column_info(table_name[, schema_name, ...])	Retrieve information about columns of a table.
get_column_names(table_name[, schema_name])	Retrieve column names of a table.
get_database_names([names_only])	Get names of all existing databases.
get_file_tables([names_only])	Retrieve information about *FileTables* from the database (if available).
get_primary_keys([table_name, schema_name, ...])	Retrieve the primary keys of table(s) from the currently-connected database.
get_row_count(table_name[, schema_name])	Get the row count of a table.
get_schema_info([names_only, include_all, ...])	Retrieve information about existing schemas.
get_table_names([schema_name, verbose])	Get names of all tables stored in one or multiple schemas.
has_dtypes(table_name, dtypes[, schema_name])	Check whether a table contains columns of specified data types.
import_data(data, table_name[, schema_name, ...])	Import tabular data into the database.
read_columns(table_name, column_names[, ...])	Read data of specific columns of a table.
read_sql_query(sql_query[, method, ...])	Executes a SQL query and read the result into a DataFrame.
read_table(table_name[, schema_name, ...])	Read data from a specified table.
schema_exists(schema_name)	Check whether a schema exists.
specify_conn_str([database_name, auth, password])	Specify the connection string for establishing a connection to a database.
table_exists(table_name[, schema_name])	Check whether a table exists.
validate_column_names(table_name[, ...])	Validate column names for use in a SQL query statement.
varchar_to_geometry_dtype(table_name[, ...])	Alter a VARCHAR column to a GEOMETRY type (MSSQL Specific).

MSSQL.add_primary_key

MSSQL.**add_primary_key**(*column_name*, *table_name*, *schema_name=None*)

Add a primary key constraint to a table.

Parameters

- **column_name** (*str*) – Name of the column to set as the primary key.
- **table_name** (*str*) – Name of the table where the primary key constraint will be added.
- **schema_name** (*str | None*) – Name of the schema where the table is

located; defaults to None.

MSSQL.connect_database

MSSQL.**connect_database**(*database_name=None*, *verbose=False*, *raise_error=False*)

Establish a connection to a database.

Parameters

- **database_name** (*str | None*) – Name of the database to connect to; if database_name=None (default), the database name is input manually.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.connect_database(verbose=True)
Being connected with <server_name>@localhost:1433/testdb.
>>> testdb.connect_database(database_name='master', verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> testdb.database_name
'master'
>>> testdb.connect_database(database_name='testdb', verbose=True)
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database [testdb]
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
>>> testdb.database_name
'master'
```

MSSQL.create_connection

MSSQL.**create_connection**(*database_name=None*, *mode=None*)

Create a SQLAlchemy connection to a Microsoft SQL Server database.

This method establishes and returns a connection to the specified database using either SQLAlchemy or pyodbc, depending on the specified mode.

Parameters

- **database_name** (*str | None*) – Name of the database to connect to; defaults to the name of the currently-connected database if database_name=None.
- **mode** (*None | str*) – Connection mode; defaults to using the existing engine if mode=None. When mode='pyodbc', it uses pyodbc.connect().

Returns
A SQLAlchemy connection or a pyodbc connection to the database.

Return type
sqlalchemy.engine.Connection | pyodbc.Connection

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> import sqlalchemy
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> db_conn = mssql.create_connection()
>>> db_conn.should_close_with_result
False
>>> db_conn.closed
False
>>> res = db_conn.execute(sqlalchemy.text('SELECT 1'))
>>> res.fetchall()
[(1,)]
>>> db_conn.closed
False
>>> db_conn.close()
>>> db_conn.closed
True
```

MSSQL.create_cursor

MSSQL.**create_cursor**(*database_name=None*)

Create a pyodbc cursor.

Parameters
database_name (*str | None*) – Name of the database to connect to; defaults to the name of the currently-connected database if `database_name=None`.

Returns
A pyodbc cursor.

Return type
pyodbc.Cursor

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> db_cur = mssql.create_cursor()
>>> # Get information about all tables in the database [master]
>>> tables_in_db = db_cur.tables(schema='dbo', tableType='TABLE')
>>> list(tables_in_db)
[('master', 'dbo', 'MSreplication_options', 'TABLE', None),
 ('master', 'dbo', 'spt_fallback_db', 'TABLE', None),
 ('master', 'dbo', 'spt_fallback_dev', 'TABLE', None),
 ('master', 'dbo', 'spt_fallback_usg', 'TABLE', None),
 ('master', 'dbo', 'spt_monitor', 'TABLE', None)]
>>> db_cur.close()
```

MSSQL.create_database

MSSQL.**create_database**(*database_name*, *verbose=False*)

Create a database.

Parameters

- **database_name** (*str*) – Name of the database to be created.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.create_database(database_name='testdb1', verbose=True)
Creating a database: [testdb1] ... Done.
>>> testdb.database_name
'testdb1'
>>> # Delete the database [testdb1]
>>> testdb.drop_database(verbose=True)
To drop the database [testdb1] from <server_name>@localhost:5432
? [No]|Yes: yes
Dropping [testdb1] ... Done.
>>> testdb.database_name
'master'
>>> # Delete the database [testdb]
>>> testdb.drop_database(database_name='testdb', verbose=True)
To drop the database [testdb] from <server_name>@localhost:5432
? [No]|Yes: yes
Dropping [testdb] ... Done.
>>> testdb.database_name
'master'
```

MSSQL.create_engine

MSSQL.**create_engine**(*database_name=None*, *auth=None*, *password=None*)

Create a SQLAlchemy connectable engine for connecting to a SQL Server database.

This method generates and returns a *SQLAlchemy* engine configured to connect to a SQL Server database using the provided or default database name, authentication method and password. The returned engine can be used to execute SQL queries and interact with the database.

Parameters

- **database_name** (*str | None*) – Name of the database to connect to; defaults to the currently-connected database if database_name=None.
- **auth** (*str | None*) – Authentication method used to establish the connection; defaults to the current authentication method if auth=None.
- **password** (*str | int | None*) – User's password; if password=None (default), manual input of the correct password is required to establish the

connection.

Returns
A SQLAlchemy connectable engine.

Return type
sqlalchemy.engine.Engine

1. Use pyodbc (or pypyodbc):

```
connect_string = 'driver={...};server=...;database=...;uid=username;pwd=...'
conn = pyodbc.connect(connect_string)  # conn = pypyodbc.connect(connect_string)
```

2. Use SQLAlchemy:

```
conn_string = 'mssql+pyodbc:///?odbc_connect=%s' % quote_plus(connect_string)
engine = sqlalchemy.create_engine(conn_string)
conn = engine.connect()
```

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> db_engine = mssql.create_engine()
>>> db_engine.name
'mssql'
>>> db_engine.dispose()
```

MSSQL.create_schema

MSSQL.**create_schema**(*schema_name*, *verbose=False*, *raise_error=False*)

Create a schema.

Parameters

- **schema_name** (*str*) – Name of the schema to be created.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> test_schema_name = 'test_schema'
>>> testdb.create_schema(schema_name=test_schema_name, verbose=True)
Creating a schema: [test_schema] ... Done.
>>> testdb.schema_exists(schema_name=test_schema_name)
True
>>> testdb.drop_database(verbose=True)  # Delete the database [testdb]
To drop the database [testdb] from <server_name>@localhost:1433
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.create_table

MSSQL.**create_table**(*table_name*, *column_specs*, *schema_name=None*, *verbose=False*, *raise_error=False*)

Create a table with specified columns.

Parameters

- **table_name** (*str*) – Name of the table to be created.
- **column_specs** (*str*) – Specifications for each column of the table.
- **schema_name** (*str | None*) – Name of the schema where the table will be created; defaults to *DEFAULT_SCHEMA* (i.e. `'dbo'`) if `schema_name=None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> # Create a new table named 'test_table'
>>> tbl_name = 'test_table'
>>> col_spec = 'col_name_1 INT, col_name_2 varchar(255)'
>>> testdb.create_table(table_name=tbl_name, column_specs=col_spec, verbose=True)
Creating a table: [dbo].[test_table] ... Done.
>>> testdb.table_exists(table_name=tbl_name)
True
>>> testdb.get_column_names(table_name=tbl_name)
['col_name_1', 'col_name_2']
>>> test_tbl_col_info = testdb.get_column_info(table_name=tbl_name, as_dict=False)
>>> test_tbl_col_info.head()
                    column_0    column_1
TABLE_CATALOG         testdb      testdb
TABLE_SCHEMA             dbo         dbo
TABLE_NAME        test_table  test_table
COLUMN_NAME       col_name_1  col_name_2
ORDINAL_POSITION           1           2
>>> testdb.validate_column_names(table_name=tbl_name)
'"col_name_1", "col_name_2"'
>>> # Drop the table [dbo].[test_table]
>>> testdb.drop_table(table_name=tbl_name, verbose=True)
To drop the table [dbo].[test_table] from <server_name>@localhost:1433/testdb
? [No]|Yes: yes
Dropping [dbo].[test_table] ... Done.
>>> # Delete the database [testdb]
>>> testdb.drop_database(verbose=True)
To drop the database [testdb] from <server_name>@localhost:1433
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.database_exists

MSSQL.**database_exists**(*database_name=None*)

Check whether a database exists.

Parameters

database_name (*str | None*) – Name of the database to check; defaults to None.

Returns

Whether the database exists.

Return type

bool

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> # Check whether the database [testdb] exists now
>>> testdb.database_name
'testdb'
>>> testdb.database_exists(database_name='testdb')
True
>>> # Delete the database [testdb]
>>> testdb.drop_database(verbose=True)
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
>>> # Check again whether the database [testdb] still exists now
>>> testdb.database_exists(database_name='testdb')
False
>>> testdb.database_name
'master'
```

MSSQL.disconnect_database

MSSQL.**disconnect_database**(*database_name=None*, *verbose=False*, *raise_error=False*)

Disconnect from a database.

Parameters

- **database_name** (*str | None*) – Name of the database to disconnect from; if database_name=None (default), disconnects from the current database.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.disconnect_database()
>>> testdb.database_name
'master'
>>> # Delete the database [testdb]
>>> testdb.drop_database(database_name='testdb', verbose=True)
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
>>> testdb.drop_database(database_name='testdb', verbose=True)
The database [testdb] does not exist.
```

MSSQL.drop_database

MSSQL.**drop_database**(*database_name=None*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Delete/drop a database.

Parameters

- **database_name** (*str | None*) – Name of the database to be dropped; if database_name=None (default), drops the currently-connected database.
- **confirmation_required** (*bool*) – Whether to prompt a confirmation message before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.database_name
'testdb'
>>> testdb.drop_database(verbose=True)  # Delete the database [testdb]
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
>>> testdb.database_exists(database_name='testdb')
False
>>> testdb.drop_database(database_name='testdb', verbose=True)
The database [testdb] does not exist.
>>> testdb.database_name
'master'
```

MSSQL.drop_schema

MSSQL.**drop_schema**(*schema_names*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Delete/drop one or multiple schemas.

Parameters

- **schema_names** (*str | Iterable[str]*) – Name of a single schema or names of multiple schemas to be dropped.
- **confirmation_required** (*bool*) – Whether to prompt a confirmation message before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> new_schema_names = ['points', 'lines', 'polygons']
>>> for new_schema in new_schema_names:
...     testdb.create_schema(new_schema, verbose=True)
Creating a schema: [points] ... Done.
Creating a schema: [lines] ... Done.
Creating a schema: [polygons] ... Done.
>>> new_schema_names_ = ['test_schema']
>>> testdb.drop_schema(new_schema_names + new_schema_names_, verbose=True)
To drop the following schemas from <server_name>@localhost:1433/testdb:
  [points]
  [lines]
  [polygons]
  [test_schema]
? [No]|Yes: yes
Dropping ...
  "points" ... Done.
  "lines" ... Done.
  "polygons" ... Done.
  "test_schema" (does not exist.)
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.drop_table

MSSQL.**drop_table**(*table_name*, *schema_name=None*, *confirmation_required=True*, *verbose=False*, *raise_error=False*)

Delete/drop a table.

Parameters

- **table_name** (*str*) – Name of the table to be deleted.
- **schema_name** (*str | None*) – Name of the schema where the table resides; defaults to *DEFAULT_SCHEMA* (i.e. `'dbo'`) if `schema_name=None`.

- **confirmation_required** (*bool*) – Whether to prompt for confirmation before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to `False`.
- **raise_error** (*bool*) – Whether to raise the provided exception; if `raise_error=False` (default), the error will be suppressed.

See also

- Examples for the method *create_table()*.

MSSQL.get_column_info

MSSQL.**get_column_info**(*table_name*, *schema_name=None*, *as_dict=True*)

Retrieve information about columns of a table.

Parameters

- **table_name** (*str*) – Name of the table to retrieve column information from.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to *DEFAULT_SCHEMA* (i.e. `'master'`) if `schema_name=None`.
- **as_dict** (*bool*) – Whether to return the column information as a dictionary; defaults to `True`.

Returns

Information about all columns of the specified table in the currently-connected database.

Return type

pandas.DataFrame | dict

See also

- Examples for the method *create_table()*.

MSSQL.get_column_names

MSSQL.**get_column_names**(*table_name*, *schema_name=None*)

Retrieve column names of a table.

Parameters

- **table_name** (*str*) – Name of the table to retrieve column names from.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to `None`.

Returns

List of column names in the specified table in the currently-connected database.

Return type
list

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_table_names()
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor']}
>>> mssql.get_column_names(table_name='MSreplication_options')
['optname',
 'value',
 'major_version',
 'minor_version',
 'revision',
 'install_failures']
```

MSSQL.get_database_names

MSSQL.**get_database_names**(*names_only=True*)

Get names of all existing databases.

Parameters
names_only (*bool*) – Whether to return only the names of the databases; defaults to True.

Returns
Names of all existing databases.

Return type
list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_database_names()
['master', 'tempdb', 'model', 'msdb']
```

MSSQL.get_file_tables

MSSQL.**get_file_tables**(*names_only=True*)

Retrieve information about *FileTables* from the database (if available).

Parameters
names_only (*bool*) – Whether to return only the names of *FileTables*; defaults to True.

Returns
Information about *FileTables* (if available).

Return type

list | pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_file_tables()
[]
```

MSSQL.get_primary_keys

MSSQL.**get_primary_keys**(*table_name=None*, *schema_name=None*, *table_type='TABLE'*)

Retrieve the primary keys of table(s) from the currently-connected database.

Parameters

- **table_name** (*str | None*) – Name of a specific table to retrieve primary keys from; when `table_name=None` (default), retrieves primary keys for all tables.
- **schema_name** (*str | None*) – Name of the schema where the table(s) are located; defaults to *DEFAULT_SCHEMA* (i.e. `'master'`) if `schema_name=None`.
- **table_type** (*str*) – Type of table to consider; defaults to `'TABLE'`.

Returns

The primary keys for the specified table(s).

Return type

list

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_primary_keys()
{}
```

MSSQL.get_row_count

MSSQL.**get_row_count**(*table_name*, *schema_name=None*)

Get the row count of a table.

Parameters

- **table_name** (*str*) – Name of the table to get row count from.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to `None`.

Returns

Number of rows in the specified table in the currently-connected database.

Return type

int | None

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_table_names()
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor']}
>>> mssql.get_row_count(table_name='MSreplication_options')
3
```

MSSQL.get_schema_info

MSSQL.**get_schema_info**(*names_only=True*, *include_all=False*, *column_names=None*, *verbose=False*)

Retrieve information about existing schemas.

Parameters

- **names_only** (*bool*) – Whether to return only the names of the schemas; defaults to `True`.
- **include_all** (*bool*) – Whether to include all available schemas; defaults to `False`.
- **column_names** (*list | None*) – Column names of the returned dataframe if `names_only=False`; defaults to `['schema_name', 'schema_id', 'role']` if `column_names=None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

Names of schemas, a dataframe with schema information, or `None` if no information is retrieved.

Return type

list | pandas.DataFrame | None

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.get_schema_info()
['dbo']
>>> test_schema_name = 'test_schema'
>>> testdb.create_schema(schema_name=test_schema_name, verbose=True)
Creating a schema: [test_schema] ... Done.
>>> testdb.get_schema_info()
['dbo', 'test_schema']
>>> testdb.get_schema_info(names_only=False, include_all=True)
          schema_name        schema_owner  schema_id
0       db_accessadmin      db_accessadmin      16385
```

(continues on next page)

(continued from previous page)

```
1    db_backupoperator   db_backupoperator      16389
2        db_datareader       db_datareader      16390
3        db_datawriter       db_datawriter      16391
4          db_ddladmin         db_ddladmin      16387
5    db_denydatareader   db_denydatareader      16392
6    db_denydatawriter   db_denydatawriter      16393
7             db_owner            db_owner      16384
8     db_securityadmin    db_securityadmin      16386
9                 dbo                 dbo          1
10              guest               guest          2
11  INFORMATION_SCHEMA  INFORMATION_SCHEMA          3
12                 sys                 sys          4
13         test_schema                 dbo          5
>>> testdb.drop_schema(schema_names='test_schema', verbose=True)
To drop the schema "test_schema" from <server_name>@localhost:1433/testdb
? [No]|Yes: yes
Dropping "test_schema" ... Done.
>>> testdb.get_schema_info()  # None
['dbo']
>>> testdb.drop_database(verbose=True)  # Delete the database "testdb"
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.get_table_names

MSSQL.**get_table_names**(*schema_name=None*, *verbose=False*)

Get names of all tables stored in one or multiple schemas.

Parameters

- **schema_name** (*str | list | None*) – Name of a single schema, or names of multiple schemas; defaults to *DEFAULT_SCHEMA* when `schema_name=None`.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to `False`.

Returns

Dictionary containing table names grouped by schema(s) specified in `schema_name`.

Return type

dict

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_table_names()
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor',
```

(continues on next page)

(continued from previous page)

```
  'test_table']}
>>> mssql.get_table_names(schema_name=['dbo', 'sys'])
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor',
  'test_table'],
 'sys': []}
```

MSSQL.has_dtypes

MSSQL.**has_dtypes**(*table_name*, *dtypes*, *schema_name=None*)

Check whether a table contains columns of specified data types.

Parameters

- **table_name** (*str*) – Name of the table in the currently-connected database.
- **dtypes** (*str | list*) – Data type(s) to check for, e.g. `'geometry'`, `'hierarchyid'`, `'varbinary'`.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to *DEFAULT_SCHEMA* (i.e. `'master'`) if `schema_name=None`.

Returns

Data type, whether the table has this data type and the corresponding column names.

Return type

Generator[str, bool, list]

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_table_names()
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor']}
>>> res = mssql.has_dtypes(table_name='spt_monitor', dtypes='varbinary')
>>> list(res)
[('varbinary', False, [])]
>>> res = mssql.has_dtypes(table_name='spt_monitor', dtypes=['geometry', 'int'])
>>> list(res)
[('geometry', False, []),
 ('int',
  True,
  ['cpu_busy',
   'io_busy',
   'idle',
   'pack_received',
   'pack_sent',
```

(continues on next page)

(continued from previous page)

```
    'connections',
    'pack_errors',
    'total_read',
    'total_write',
    'total_errors'])]
```

MSSQL.import_data

MSSQL.**import_data**(*data*, *table_name*, *schema_name=None*, *if_exists='fail'*, *force_replace=False*, *chunk_size=None*, *dtype=None*, *method='multi'*, *index=False*, *geom_column_name=None*, *srid=None*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import tabular data into the database.

See also [DBMS-MS-ID-1].

Parameters

- **data** (*pandas.DataFrame | pandas.io.parsers.TextFileReader | list | tuple*) – Tabular data to be imported into the database. It can be a dataframe, a `TextFileReader` or a list/tuple of tuples.
- **table_name** (*str*) – Name of the table where the data will be imported.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to *DEFAULT_SCHEMA* (i.e. `'master'`) if `schema_name=None`.
- **if_exists** (*str*) – Action to take if the table already exists:
  - `'replace'`: Drop the table before inserting new data.
  - `'append'`: Insert new data to the existing table.
  - `'fail'`: Raise a ValueError if the table already exists (default).
- **force_replace** (*bool*) – Whether to force replace the existing table; defaults to `False`.
- **chunk_size** (*int | None*) – Number of rows to insert at a time; defaults to `None` (all at once).
- **dtype** (*dict | None*) – Dictionary specifying column data types; defaults to `None`.
- **method** (*str | None | Callable*) – Method for SQL insertion clause:
  - `None`: Uses standard SQL `INSERT` clause (one per row).
  - `'multi'`: Passes multiple values in a single `INSERT` clause (default).
- **index** (*bool*) – Whether to include the DataFrame index as a column in the database table.
- **geom_column_name** (*str | None*) – Name of the geometry column if importing spatial data; defaults to `None`.
- **srid** (*int | None*) – Spatial Reference Identifier (SRID) associated with the coordinate system, tolerance and resolution; defaults to `None`.

- **confirmation_required** (*bool*) – Whether to prompt a confirmation message before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print detailed information during the import process; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the method pandas.DataFrame.to_sql().

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> from pyhelpers._cache import example_dataframe

>>> testdb = MSSQL(database_name='testdb', verbose=True)
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.

>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418

>>> test_table_name = 'example_df'
>>> testdb.import_data(example_df, test_table_name, index=True, verbose=2)
To import data into [dbo].[example_df] at <server_name>@localhost:1433/testdb
? [No]|Yes: yes
Importing the data into the table [dbo].[example_df] ... Done.
```

The imported example data can also be viewed using Microsoft SQL Server Management Studio (as illustrated in Figure 20 below):

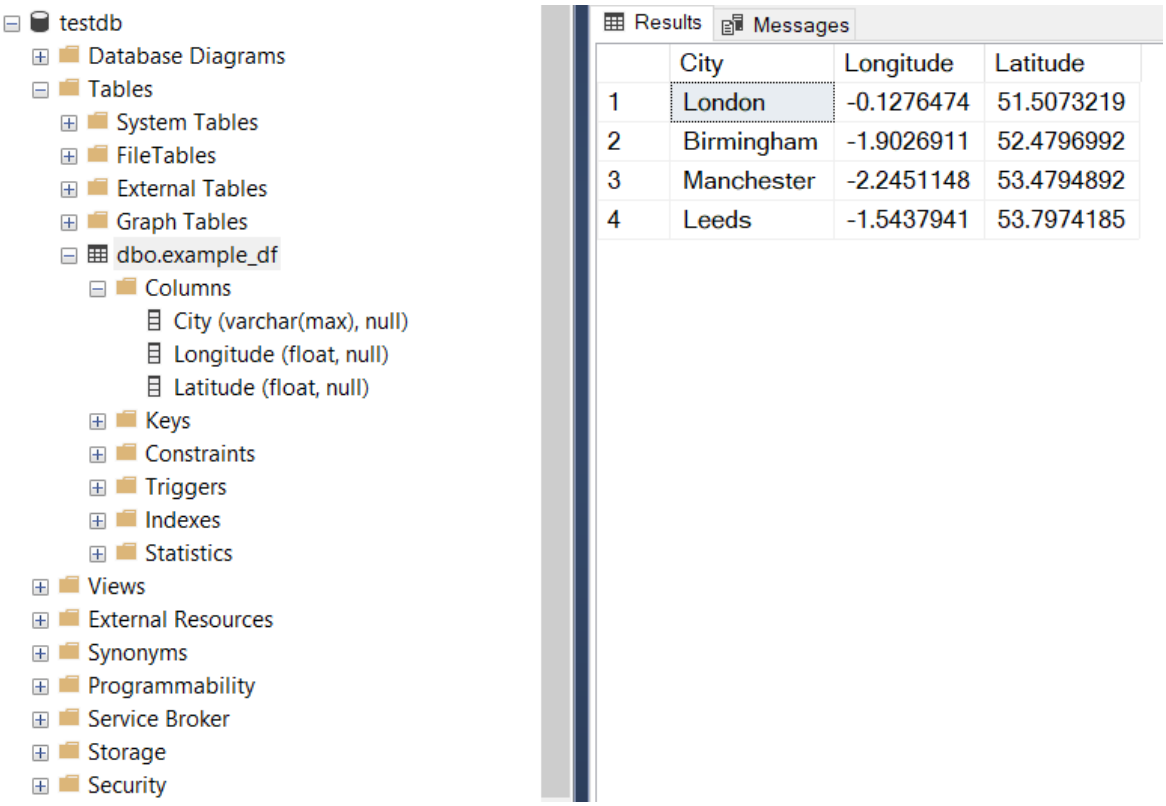

Figure 20: The table *[dbo].[example_df]* in the database *[testdb]*.

```
>>> # Drop/delete the table [dbo].[example_df]
>>> testdb.drop_table(table_name=test_table_name, verbose=True)
To drop the table [dbo].[example_df] from <server_name>@localhost:1433/testdb
? [No]|Yes: yes
Dropping [dbo].[example_df] ... Done.

>>> # Drop/delete the database [testdb]
>>> testdb.drop_database(verbose=True)
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

See also

- Examples for the method `read_table()`.

MSSQL.read_columns

MSSQL.**read_columns**(*table_name*, *column_names*, *dtype=None*, *schema_name=None*, *chunk_size=None*, ***kwargs*)

Read data of specific columns of a table.

Parameters

- **table_name** (*str*) – Name of a table in the currently-connected database.

- **column_names** (*list | tuple*) – Column name(s) of the specified table.
- **dtype** (*str | None*) – data type; options are {'hierarchyid', 'varbinary', 'geometry'}; defaults to None.
- **schema_name** (*str | None*) – Name of a schema, defaults to *DEFAULT_SCHEMA* when schema_name=None.
- **chunk_size** (*int | None*) – Number of rows to include in each chunk (if specified); defaults to None
- **kwargs** – [Optional] Additional parameters for the function pandas.read_sql().

Returns

Data of specific columns of the queried table.

Return type

pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> from pyhelpers._cache import example_dataframe
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_table_names()
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor']}
>>> mssql.read_columns('MSreplication_options', column_names=['optname', 'value'])
          optname  value
0   transactional   True
1           merge   True
2  security_model   True
```

See also

- Examples for the method *read_table()*.

MSSQL.read_sql_query

MSSQL.**read_sql_query**(*sql_query*, *method='tempfile'*, *max_size_spooled=1*, *delimiter=','*, *tempfile_kwargs=None*, *stringio_kwargs=None*, ***kwargs*)

Executes a SQL query and read the result into a DataFrame.

Parameters

- **sql_query** (*str*) – SQL query to execute.
- **method** (*str*) – Method for reading the query result.
- **max_size_spooled** (*int*) – Maximum size in bytes before spooling to disk.
- **delimiter** (*str*) – Delimiter to use for reading the query result.

- **tempfile_kwargs** (*dict*) – Additional keyword arguments for `tempfile`.
- **stringio_kwargs** (*dict*) – Additional keyword arguments for `StringIO`.
- **kwargs** – [Optional] Additional arguments passed to the reading method.

MSSQL.read_table

MSSQL.**read_table**(*table_name*, *schema_name=None*, *column_names=None*, *conditions=None*, *chunk_size=None*, *save_as=None*, *data_dir=None*, *save_args=None*, *verbose=False*, ***kwargs*)

Read data from a specified table.

Parameters

- **table_name** (*str*) – Name of the table to read data from in the currently-connected database.
- **schema_name** (*str | None*) – Name of the schema where the table resides; defaults to *DEFAULT_SCHEMA* when `schema_name=None`.
- **column_names** (*list | tuple | None*) – Names of columns to retrieve data from; defaults to all columns when `column_names=None`.
- **conditions** (*str | None*) – Conditions to apply in the SQL query statement; defaults to `None`.
- **chunk_size** (*int | None*) – Number of rows to retrieve in each chunk (if specified); defaults to `None`.
- **save_as** (*str | None*) – File extension (if specified) for saving table data locally; defaults to `None`.
- **data_dir** (*str | None*) – Directory path where the table data should be saved; defaults to `None`.
- **save_args** (*dict | None*) – Optional parameters for the function *pyhelpers.store.save_data()*; defaults to `None`.
- **verbose** (*bool | int*) – Whether to print relevant information in the console; defaults to `False`.
- **kwargs** – [Optional] Additional parameters for the function pandas.read_sql().

Returns

Data of the queried table from the currently-connected database.

Return type

pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> from pyhelpers._cache import example_dataframe
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.get_table_names()
```

(continues on next page)

(continued from previous page)

```
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor']}
>>> mssql.read_table(table_name='MSreplication_options')
          optname  value  ...  revision  install_failures
0   transactional   True  ...         0                 0
1           merge   True  ...         0                 0
2  security_model   True  ...         0                 0
[3 rows x 6 columns]
>>> mssql.read_table(table_name='MSreplication_options', column_names=['optname'])
          optname
0   transactional
1           merge
2  security_model
>>> # Create a new database for testing
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> example_df = example_dataframe()
>>> example_df
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> test_table_name = 'example_df'
>>> testdb.import_data(example_df, test_table_name, index=True, verbose=2)
To import data into [dbo].[example_df] at <server_name>@localhost:1433/testdb
? [No]|Yes: yes
Importing the data into the table [dbo].[example_df] ... Done.
>>> # Retrieve the imported data
>>> example_df_ret = testdb.read_table(test_table_name, index_col='City')
>>> example_df_ret
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> # Drop/Delete the testing database [testdb]
>>> testdb.drop_database(verbose=True)
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

See also

- Examples for the method *import_data()*.

MSSQL.schema_exists

MSSQL.**schema_exists**(*schema_name*)

Check whether a schema exists.

Parameters

schema_name (*str*) – Name of the schema to check.

Returns

Whether the schema exists.

Return type

bool

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> testdb = MSSQL(database_name='testdb')
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> testdb.schema_exists('dbo')
True
>>> testdb.schema_exists('test_schema')  # (if the schema [test_schema] does not exist)
False
>>> testdb.drop_database(verbose=True)  # Delete the database [testdb]
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.specify_conn_str

MSSQL.**specify_conn_str**(*database_name=None*, *auth=None*, *password=None*)

Specify the connection string for establishing a connection to a database.

Parameters

- **database_name** (*str | None*) – Name of the database to connect to; defaults to the currently-connected database if `database_name=None`.
- **auth** (*str | None*) – Authentication method used for the connection; defaults to the current authentication method if `auth=None`.
- **password** (*str | int | None*) – User's password; if `password=None` (default), manual input of the correct password is required to establish the connection.

Returns

Connection string formatted for establishing a connection.

Return type

str

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> conn_str = mssql.specify_conn_str()
>>> conn_str
'DRIVER={ODBC Driver 17 for SQL Server};SERVER={localhost};DATABASE={master};...
```

MSSQL.table_exists

MSSQL.**table_exists**(*table_name*, *schema_name=None*)

Check whether a table exists.

Parameters

- **table_name** (*str*) – Name of the table to check.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to *DEFAULT_SCHEMA* (i.e. `'dbo'`) if `schema_name=None`.

Returns

Whether the table exists in the currently-connected database.

Return type

bool

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> mssql = MSSQL(verbose=True)
Connecting <server_name>@localhost:1433/master ... Successfully.
>>> mssql.table_exists(table_name='test_table')
False
>>> mssql.get_table_names()
{'dbo': ['MSreplication_options',
  'spt_fallback_db',
  'spt_fallback_dev',
  'spt_fallback_usg',
  'spt_monitor']}
>>> mssql.table_exists(table_name='MSreplication_options')
True
```

MSSQL.validate_column_names

MSSQL.**validate_column_names**(*table_name*, *schema_name=None*, *column_names=None*)

Validate column names for use in a SQL query statement.

Parameters

- **table_name** (*str*) – Name of the table.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to `None`.
- **column_names** (*str | list | tuple | None*) – Column name(s) to validate for a dataframe.

Returns

Validated column names formatted for a SQL query statement.

Return type
str

See also

- Examples for the method *create_table()*.

MSSQL.varchar_to_geometry_dtype

MSSQL.**varchar_to_geometry_dtype**(*table_name*, *geom_column_name=None*, *srid=None*, *schema_name=None*, *verbose=True*, *raise_error=False*)

Alter a VARCHAR column to a GEOMETRY type (MSSQL Specific).

Parameters

- **table_name** (*str*) – Name of the table where the column exists.
- **geom_column_name** (*str | None*) – Name of the VARCHAR column to convert to geometry data type.
- **srid** (*int | None*) – Spatial Reference Identifier (SRID) associated with the coordinate system, tolerance and resolution; defaults to None.
- **schema_name** (*str | None*) – Name of the schema where the table is located; defaults to *DEFAULT_SCHEMA* (i.e. 'master') if schema_name=None.
- **verbose** (*bool | int*) – Whether to print error information if any. Defaults to True.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.

See also

- Reference: [DBMS-MS-4].
- Examples for *import_data()*.

4.7.2 Database tools/utilities

1anti-flashwhitewhite

make_database_address(host, port, username)	Generates a string representing a database address.
get_default_database_address(db_cls)	Retrieves the database address of a database class given its default parameters.
add_sql_query_condition(sql_query[, ...])	Adds a condition to a given SQL query statement.
get_adaptive_index_dtypes(data[, index, ...])	Generates SQLAlchemy types for MSSQL compatibility, specifically for indices.

continues on next page

Table 36 – continued from previous page

import_data(db_instance, data, schema_name, ...)	Imports data into the project database.
read_data(db_instance, schema_name, table_name)	Reads data from a database using either a table name or custom SQL.
mssql_to_postgresql(mssql, postgres[, ...])	Copies tables of a database from a Microsoft SQL server to a PostgreSQL server.

make_database_address

pyhelpers.dbms.utils.**make_database_address**(*host*, *port*, *username*, *database_name=''*)

Generates a string representing a database address.

Parameters

- **host** (*str | None*) – Host name or IP address of the database server.
- **port** (*int | str | None*) – Port number of the database server.
- **username** (*str | None*) – Username for the connection.
- **database_name** (*str | None*) – Name of the database; defaults to an empty string.

Returns

Formatted address as `'<username>:***@<host>:<port>[/<database_name>]'`.

Return type

str

Examples:

```
>>> from pyhelpers.dbms.utils import make_database_address
>>> make_database_address('localhost', 5432, 'postgres', 'postgres')
'postgres:***@localhost:5432/postgres'
>>> make_database_address('localhost', 5432, 'admin', 'my_database')
'admin:***@localhost:5432/my_database'
>>> make_database_address('127.0.0.1', '5432', 'user', '')
'user:***@127.0.0.1:5432'
```

get_default_database_address

pyhelpers.dbms.utils.**get_default_database_address**(*db_cls*)

Retrieves the database address of a database class given its default parameters.

Parameters

db_cls (*object*) – Class representing a database.

Returns

Address of the database given its default parameters in the format '<username>@<host>:<port>/<database_name>'.

Return type

str

Examples:

```
>>> from pyhelpers.dbms.utils import get_default_database_address
>>> from pyhelpers.dbms import PostgreSQL
>>> db_addr = get_default_database_address(db_cls=PostgreSQL)
>>> db_addr
'None:***@None:None'
```

add_sql_query_condition

pyhelpers.dbms.utils.**add_sql_query_condition**(*sql_query*, *add_table_name=None*, ***kwargs*)

Adds a condition to a given SQL query statement.

Parameters

- **sql_query** (*str*) – SQL query statement to which the condition will be added.
- **add_table_name** (*str | None*) – [Optional] table name to add to each column name in the condition; defaults to None.

Returns

Updated SQL query statement with the specified conditions.

Return type

str

Examples:

```
>>> from pyhelpers.dbms.utils import add_sql_query_condition
>>> query = 'SELECT * FROM a_table'
>>> query
'SELECT * FROM a_table'
>>> add_sql_query_condition(query)
'SELECT * FROM a_table'
>>> add_sql_query_condition(query, COL_NAME_1='A')
'SELECT * FROM a_table WHERE "COL_NAME_1"='A''
>>> add_sql_query_condition(query, COL_NAME_1='A', COL_NAME_2=['B', 'C'])
'SELECT * FROM a_table WHERE "COL_NAME_1"='A' AND "COL_NAME_2" IN ('B', 'C')'
>>> add_sql_query_condition(query, COL_NAME_1='A', add_table_name='t1')
'SELECT * FROM a_table WHERE t1."COL_NAME_1"='A''
```

get_adaptive_index_dtypes

pyhelpers.dbms.utils.**get_adaptive_index_dtypes**(*data*, *index=False*, *dtype=None*, *max_len=255*, *verbose=False*)

Generates SQLAlchemy types for MSSQL compatibility, specifically for indices.

This function prevents MSSQL error 42000, which occurs when attempting to use an NVARCHAR(MAX) column as an index key. It maps string-based indices to fixed-length NVARCHAR types.

Parameters

- **data** (*pandas.DataFrame | pandas.io.parsers.TextFileReader | list | tuple*) – Tabular data to be inspected.
- **index** (*bool*) – Whether the index will be included in the database import. Defaults to False.

- **dtype** (*dict | None*) – Existing dictionary mapping column names to SQLAlchemy types. Defaults to None.
- **max_len** (*int*) – Maximum character length for the index column; defaults to 255. Note: For MSSQL, this should not exceed 450 for Unicode (NVARCHAR) columns.
- **verbose** (*bool | int*) – Whether to print adaptive mapping information. Defaults to False.

Returns
A dictionary of column/index names mapped to SQLAlchemy types.

Return type
dict

Examples:

```
>>> from pyhelpers.dbms.utils import get_adaptive_index_dtypes
>>> from pyhelpers._cache import example_dataframe
>>> df = example_dataframe()
>>> get_adaptive_index_dtypes(df, index=True)
{'City': Unicode(length=255)}
```

import_data

pyhelpers.dbms.utils.**import_data**(*db_instance*, *data*, *schema_name*, *table_name*, *data_name='data'*, *prefix=''*, *suffix=''*, *confirmation_required=True*, *verbose=False*, *raise_error=False*, ***kwargs*)

Imports data into the project database.

Parameters

- **db_instance** (*Any*) – A class instance for handling the database.
- **data** (*pandas.DataFrame*) – Data to import (from a local directory).
- **schema_name** (*str*) – Name of the schema where the table resides.
- **table_name** (*str*) – Name of the table where data will be imported.
- **data_name** (*str*) – Name identifier for the data; defaults to "data".
- **prefix** (*str*) – Prefix to prepend to data_name; defaults to ''.
- **suffix** (*str*) – Suffix to append to data_name; defaults to ''.
- **confirmation_required** (*bool*) – Whether to request confirmation before proceeding; defaults to True.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the method *PostgreSQL.import_data()* or *MSSQL.import_data()*.

See also

- Examples for the method *read_data()*.

read_data

pyhelpers.dbms.utils.**read_data**(*db_instance*, *schema_name*, *table_name*, *sql_query=None*, *data_name='data'*, *prefix=''*, *suffix=''*, *verbose=False*, *raise_error=False*, ***kwargs*)

Reads data from a database using either a table name or custom SQL.

Parameters

- **db_instance** (*pyhelpers.dbms.PostgreSQL | pyhelpers.dbms.MSSQL*) – A class instance for handling the database.
- **schema_name** (*str*) – Name of the schema from which to load data.
- **table_name** (*str*) – Name of the table from which to load data.
- **sql_query** (*str | None*) – SQL query statement for custom data retrieval; defaults to None.
- **data_name** (*str*) – Name identifier for the loaded data; defaults to "data".
- **prefix** (*str*) – Prefix to prepend to data_name; defaults to ''.
- **suffix** (*str*) – Suffix to append to data_name; defaults to ''.
- **verbose** (*bool | int*) – Whether to print relevant information to the console; defaults to False.
- **raise_error** (*bool*) – Whether to raise the provided exception; if raise_error=False (default), the error will be suppressed.
- **kwargs** – [Optional] Additional parameters for the method *PostgreSQL.read_sql_query()*, *PostgreSQL.read_table()* or *MSSQL.read_table()*.

Returns

Queried data as a dataframe.

Return type

pandas.DataFrame | None

Examples:

```
>>> from pyhelpers.dbms.utils import import_data, read_data
>>> from pyhelpers.dbms import PostgreSQL
>>> from pyhelpers._cache import example_dataframe
>>> testdb_name = 'testdb'
>>> testdb = PostgreSQL(database_name=testdb_name, verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # Import an example dataframe into a table named "points"."England"
>>> example_df = example_dataframe()
```

(continues on next page)

(continued from previous page)

```
>>> test_schema_name = 'points'
>>> test_table_name = 'England'
>>> test_data_name = 'the data of "England points"'
>>> import_data(
...     testdb, example_df, test_schema_name, test_table_name, test_data_name, index=True,
...     verbose=True)
To import the data of "England points" into "points"."England"?
 [No]|Yes: yes
Importing the data ... Done.
>>> # Retrieve the data
>>> example_df_ = read_data(
...     testdb, test_schema_name, test_table_name, data_name=test_data_name,
...     index_col='City', verbose=True)
Reading the data of "England points" from "points"."England" ... Done.
>>> # Check whether the retrieved data is the same as the original example dataframe
>>> example_df_.equals(example_df)
True
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

mssql_to_postgresql

pyhelpers.dbms.utils.**mssql_to_postgresql**(*mssql*, *postgres*, *mssql_schema=None*, *postgres_schema=None*, *chunk_size=None*, *excluded_tables=None*, *file_tables=False*, *memory_threshold=2.0*, *update=False*, *confirmation_required=True*, *verbose=True*)

Copies tables of a database from a Microsoft SQL server to a PostgreSQL server.

Parameters

- **mssql** (pyhelpers.dbms.MSSQL) – Name of the Microsoft SQL (source) database.
- **postgres** (pyhelpers.dbms.PostgreSQL) – Name of the PostgreSQL (destination) database.
- **mssql_schema** (*str | None*) – Name of the schema to be migrated from the SQL Server.
- **postgres_schema** (*str | None*) – Name of the schema to store the migrated data in the PostgreSQL server.
- **chunk_size** (*int | None*) – Number of rows in each batch to be read/written at a time; defaults to None.
- **excluded_tables** (*list | None*) – Names of tables excluded from data migration.
- **file_tables** (*bool*) – Whether to include FileTables; defaults to False.
- **memory_threshold** (*float | int*) – Threshold (in GiB) beyond which data is migrated by partitions; defaults to 2..

- **update** (*bool*) – Whether to redo the transfer between database servers; defaults to `False`.
- **confirmation_required** (*bool*) – Whether to request confirmation before proceeding; defaults to `True`.
- **verbose** (*bool | int*) – Whether to print relevant information; defaults to `True`.

Examples:

```
>>> from pyhelpers.dbms.utils import mssql_to_postgresql
>>> from pyhelpers.dbms import PostgreSQL, MSSQL
>>> from pyhelpers._cache import example_dataframe
>>> # Connect/create a PostgreSQL database, which is named [testdb]
>>> mssql_testdb = MSSQL(database_name='testdb', verbose=True)
Creating a database: [testdb] ... Done.
Connecting <server_name>@localhost:1433/testdb ... Successfully.
>>> mssql_testdb.database_name
'testdb'
>>> example_df = example_dataframe()
>>> example_df
           Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418
>>> test_table_name = 'example_df'
>>> # Import the example dataframe into a table named [example_df]
>>> mssql_testdb.import_data(example_df, table_name=test_table_name, index=True, verbose=2)
To import data into [dbo].[example_df] at <server_name>@localhost:1433/testdb
? [No]|Yes: yes
Importing the data into the table [dbo].[example_df] ... Done.
>>> mssql_testdb.get_column_names(table_name=test_table_name)
['City', 'Longitude', 'Latitude']
```

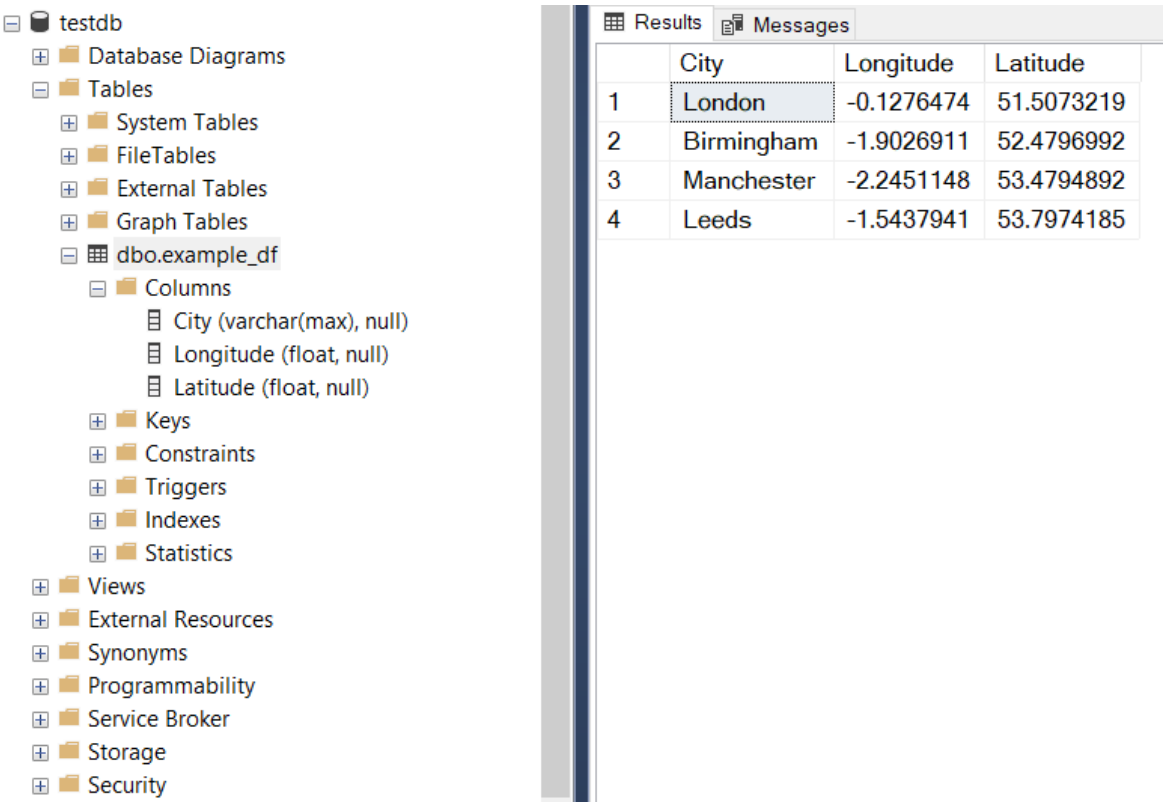

Figure 21: The table *[dbo].[example_df]* in the Microsoft SQL Server database *[testdb]*.

```
>>> # Create an instance for a PostgreSQL database, which is also named "testdb"
>>> postgres_testdb = PostgreSQL(database_name='testdb', verbose=True)
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> # For now, the newly-created database doesn't contain any tables
>>> postgres_testdb.get_table_names()
{'public': []}
>>> # Copy the example data from the SQL Server to the PostgreSQL "testdb" (under "public")
>>> mssql_to_postgresql(mssql=mssql_testdb, postgres=postgres_testdb)
To copy tables from [testdb] (MSSQL) to "testdb" (PostgreSQL)
? [No]|Yes: yes
Processing tables ...
    (1/1) Copying [dbo].[example_df] to "public"."example_df" ... Done.
Completed.
>>> postgres_testdb.get_table_names()
{'public': ['example_df']}
```

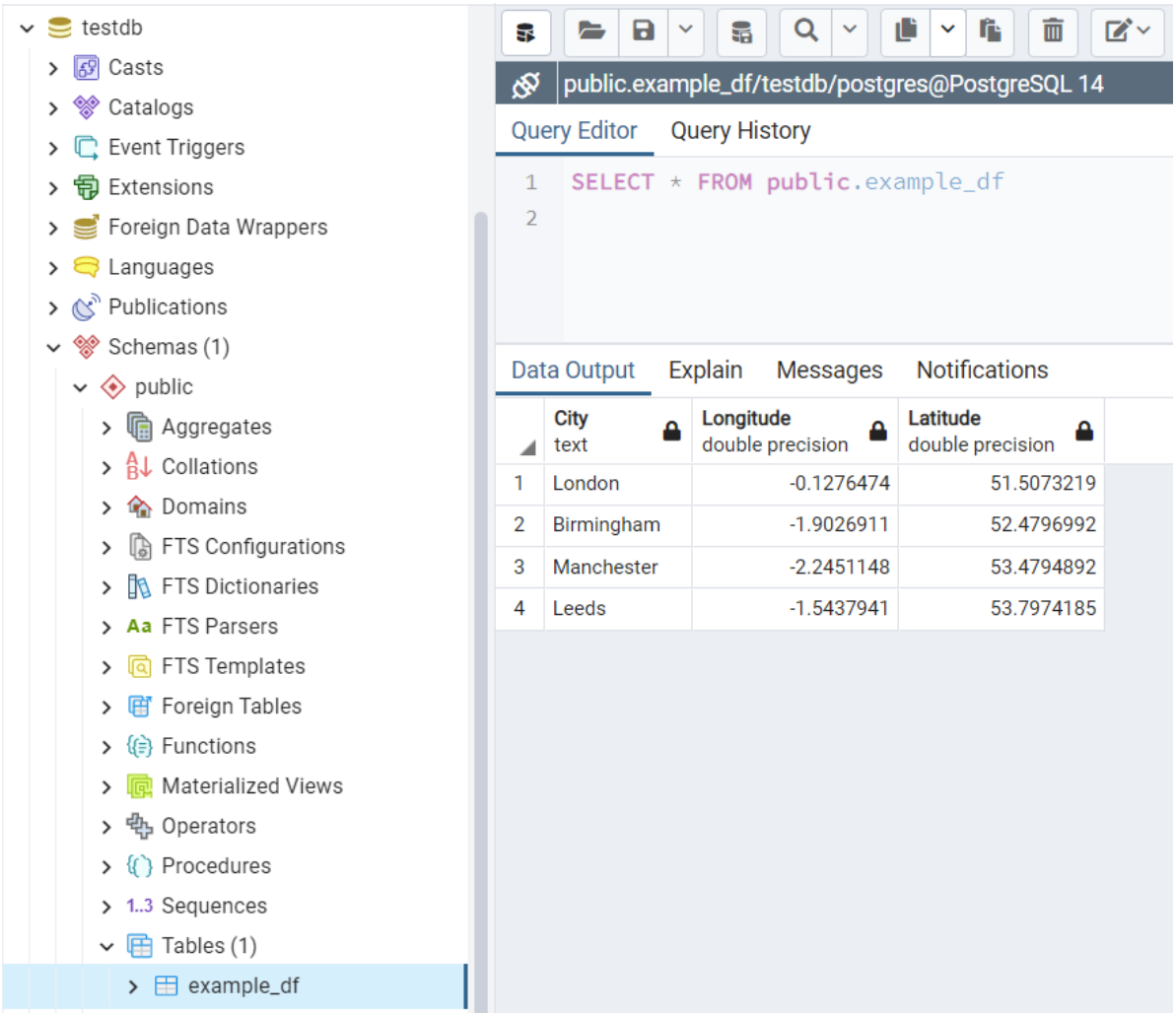

Figure 22: The table *"dbo"."example_df"* in the PostgreSQL database *"testdb"*.

```
>>> # Drop/delete the created databases
>>> mssql_testdb.drop_database(verbose=True)
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
>>> postgres_testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

4.8 viz

Visualization utilities for geospatial and map-related operations.

This subpackage provides tools to enhance and simplify visualization workflows (for interactive maps and geospatial data rendering).

4.8.1 General utilities

1anti-flashwhitewhite

cmap_discretization(cmap, n_colors)	Creates a discrete colormap based on the input.
color_bar_index(cmap, n_colors[, labels])	Creates a color bar with correctly aligned labels.

cmap_discretization

pyhelpers.viz.**cmap_discretization**(*cmap*, *n_colors*)

Creates a discrete colormap based on the input.

Parameters

- **cmap** (*matplotlib.colors.ListedColormap | matplotlib.colors.LinearSegmentedColormap | matplotlib.colors.Colormap | str*) – A colormap instance, e.g. built-in colormaps available via matplotlib.pyplot.get_cmap.
- **n_colors** (*int*) – Number of colors to discretize the colormap.

Returns

A discrete colormap derived from the continuous cmap.

Return type

matplotlib.colors.LinearSegmentedColormap

Examples:

```
>>> from pyhelpers.viz import cmap_discretization
>>> from pyhelpers.settings import mpl_preferences
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> mpl_preferences()
>>> cm_accent = cmap_discretization(cmap=plt.colormaps['Accent'], n_colors=5)
>>> cm_accent.name
'Accent_5'
>>> cm_accent = cmap_discretization(cmap='Accent', n_colors=5)
>>> cm_accent.name
'Accent_5'
>>> fig = plt.figure(figsize=(10, 2), constrained_layout=True)
>>> ax = fig.add_subplot()
>>> ax.imshow(np.resize(range(100), (5, 100)), cmap=cm_accent, interpolation='nearest')
>>> ax.axis('off')
>>> fig.show()
>>> # from pyhelpers.store import save_figure
>>> # path_to_fig_ = "docs/source/_images/viz-cmap_discretization-demo"
>>> # save_figure(fig, f"{path_to_fig_}.svg", verbose=True)
>>> # save_figure(fig, f"{path_to_fig_}.pdf", verbose=True)
```

The example is illustrated in Figure 23:

Figure 23: An example of discrete color ramp, created by the function *cmap_discretization()*.

color_bar_index

pyhelpers.viz.**color_bar_index**(*cmap*, *n_colors*, *labels=None*, ***kwargs*)

Creates a color bar with correctly aligned labels.

> **Note**
>
> - To avoid off-by-one errors, this function takes a standard colormap, discretizes it, and then draws a color bar with labels aligned correctly.

Parameters

- **cmap** (*matplotlib.colors.ListedColormap | matplotlib.colors.Colormap*) – A colormap instance, e.g. built-in colormaps accessible via matplotlib.pyplot.get_cmap().
- **n_colors** (*int*) – Number of discrete colors to use in the color bar.
- **labels** (*list | None*) – Optional list of labels for the color bar; defaults to None.
- **kwargs** – [Optional] Additional optional parameters for the matplotlib.pyplot.colorbar() function.

Returns

A color bar object.

Return type

matplotlib.colorbar.Colorbar

Examples:

```
>>> from pyhelpers.viz import color_bar_index
>>> from pyhelpers.settings import mpl_preferences
>>> import matplotlib.pyplot as plt
>>> mpl_preferences()
>>> fig1 = plt.figure(figsize=(2, 6), constrained_layout=True)
>>> ax1 = fig1.add_subplot()
>>> cbar1 = color_bar_index(cmap=plt.colormaps['Accent'], n_colors=5)
>>> ax1.tick_params(axis='both', which='major', labelsize=14)
>>> cbar1.ax.tick_params(labelsize=14)
>>> # ax.axis('off')
>>> fig1.show()
>>> # from pyhelpers.store import save_figure
>>> # path_to_fig1_ = "docs/source/_images/viz-color_bar_index-demo-1"
>>> # save_figure(fig1, f"{path_to_fig1_}.svg", verbose=True)
>>> # save_figure(fig1, f"{path_to_fig1_}.pdf", verbose=True)
```

The above example is illustrated in Figure 24:

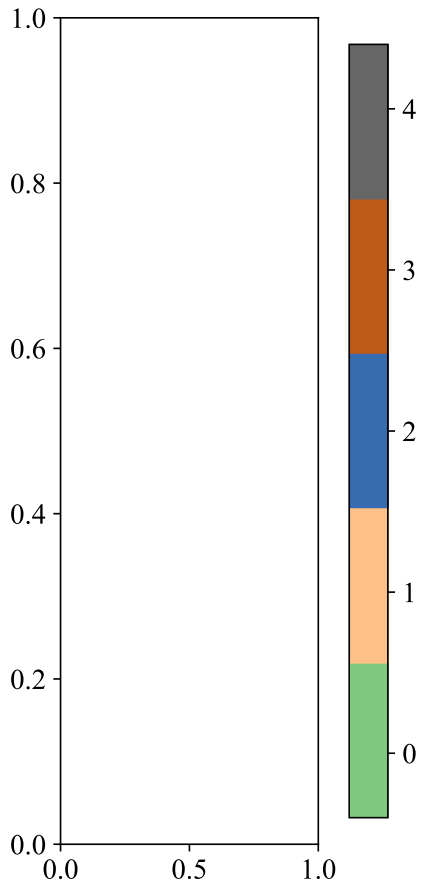

Figure 24: An example of color bar with numerical index.

```
>>> fig2 = plt.figure(figsize=(2, 6), constrained_layout=True)
>>> ax2 = fig2.add_subplot()
>>> labels_ = list('abcde')
>>> cbar2 = color_bar_index(cmap=plt.colormaps['Accent'], n_colors=5, labels=labels_)
>>> ax2.tick_params(axis='both', which='major', labelsize=14)
>>> cbar2.ax.tick_params(labelsize=14)
>>> # ax.axis('off')
>>> fig2.show()
>>> # path_to_fig2_ = "docs/source/_images/viz-color_bar_index-demo-2"
>>> # save_figure(fig2, f"{path_to_fig2_}.svg", verbose=True)
>>> # save_figure(fig2, f"{path_to_fig2_}.pdf", verbose=True)
```

This second example is illustrated in Figure 25:

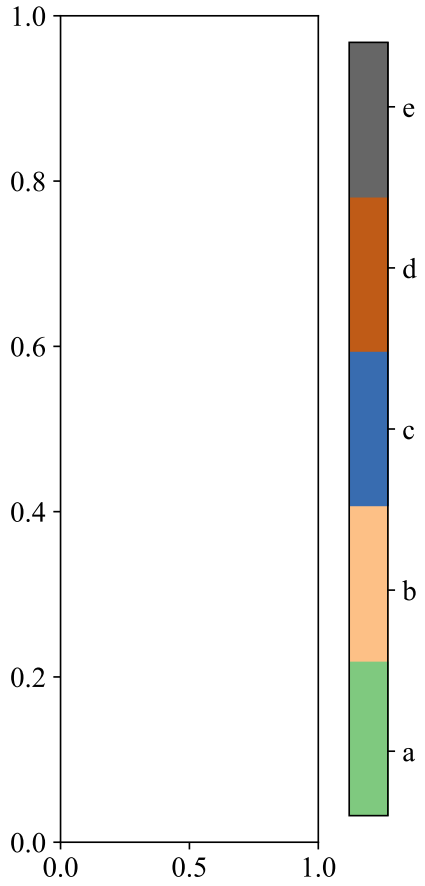

Figure 25: An example of color bar with textual index.

4.8.2 Utilities for maps

1anti-flashwhitewhite

get_base_map_center(gdf[, center_method])	Calculates geographical center and bounding box for a GeoDataFrame.
create_base_folium_map(gdf[, center_method, ...])	Initializes a Folium map centered on the extent of a GeoDataFrame.
BindColormap(layer, colormap[, display])	Binds a colormap to a Folium layer with visibility synchronization.

get_base_map_center

class pyhelpers.viz.**get_base_map_center**(*gdf*, *center_method='bounds'*)

Calculates geographical center and bounding box for a GeoDataFrame.

This function prepares data for web mapping by re-projecting to EPSG:4326 and calculating coordinates based on the chosen geometric method. It is primarily used to determine the initial central point for creating a base map using libraries such as Folium or Leaflet.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – Input GeoDataFrame containing

geographical features.

- **center_method** (*str*) – Method to calculate the center:
  - `'bounds'`: Calculates the center based on the arithmetic mean of the bounding box corners (quickest method).
  - `'centroid'`: Calculates the true geometric centroid of the merged (union) geometry (most accurate, requires projection).
  - `'convex_hull'`: Calculates the centroid of the convex hull of all geometries (quicker than 'centroid' for complex data).

Returns

A tuple containing the re-projected GeoDataFrame and the center coordinates. The center coordinates are returned as a list *[latitude, longitude]*.

Return type

tuple[list[float], list[float], list[float], geopandas.GeoDataFrame]

Examples:

```
>>> from pyhelpers.viz import get_base_map_center
>>> import geopandas as gpd
>>> from shapely.geometry import Point
>>> gdf = gpd.GeoDataFrame({'col': [1, 2]}, geometry=[Point(0, 0), Point(2, 2)], crs=4326)
>>> center_lat_lon, ll_lat_lon, ur_lat_lon, gdf_proj = get_base_map_center(gdf)
>>> center_lat_lon
[1.0, 1.0]
>>> center_lat_lon, ll_lat_lon, ur_lat_lon, gdf_proj = get_base_map_center(
...     gdf, center_method='centroid')
>>> center_lat_lon
[1.0001523435165862, 0.9999999999999998]
```

create_base_folium_map

class pyhelpers.viz.**create_base_folium_map**(*gdf*, *center_method='bounds'*, *fit_bounds=True*, *tiles=None*, *initial_tile_name=None*, *add_mini_map=True*, ***kwargs*)

Initializes a Folium map centered on the extent of a GeoDataFrame.

This function first calls *get_base_map_center()* to calculate the map center coordinates and re-project the data to EPSG:4326 (WGS84), which is required by Folium.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – Input GeoDataFrame containing the geographical features.
- **center_method** (*str*) – The method used to calculate the map center.
  - `'bounds'`: Calculates the center based on the arithmetic mean of the bounding box corners (quickest method).
  - `'centroid'`: Calculates the true geometric centroid of the merged (union) geometry (most accurate, requires projection).

  - `'convex_hull'`: Calculates the centroid of the convex hull of all geometries (quicker than 'centroid' for complex data).

- **fit_bounds** (*bool*) – Whether to automatically adjust the map view to fit the full extent of the data bounds. Defaults to `True`.
- **tiles** (*str | list[str] | None*) – The tile layer(s) to use. Can be a string (e.g. `'CartoDB positron'`) or a list of strings for multiple tile layers.
- **initial_tile_name** (*str | None*) – Optional custom name to display for the first (default) tile layer in the map's Layer Control. If None, the default tile name is used.
- **add_mini_map** (*bool*) – If `True`, a small inset map is added to the bottom-right corner using `folium.plugins.MiniMap`.
- **kwargs** (*dict*) – Additional arguments passed to the `folium.Map` constructor.

Returns

Tuple of (re-projected GeoDataFrame, Folium Map object).

Return type

tuple[folium.Map, geopandas.GeoDataFrame]

Examples:

```
>>> from pyhelpers.viz import create_base_folium_map
>>> from pyhelpers._cache import example_dataframe
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> df = example_dataframe()
>>> df['geometry'] = df.apply(lambda x: Point([x.Longitude, x.Latitude]), axis=1)
>>> gdf = gpd.GeoDataFrame(df, geometry='geometry', crs=4326)
>>> m, gdf_proj = create_base_folium_map(gdf, fit_bounds=False, zoom_start=7)
>>> m.show_in_browser()
```

BindColormap

class pyhelpers.viz.**BindColormap**(*layer*, *colormap*, *display=True*)

Binds a colormap to a Folium layer with visibility synchronization.

This class creates a connection between a Folium map layer and a color scale, showing/hiding the colormap legend when the layer visibility changes.

Parameters

- **layer** (*folium.FeatureGroup | folium.GeoJson*) – Folium layer object (`FeatureGroup`, `GeoJson`, etc.) to bind with.
- **colormap** (*branca.colormap.ColorMap*) – Color scale to display and synchronize with layer visibility.
- **display** (*bool*) – Initial visibility state of the colormap legend; when `display=True` (default), the legend will be visible by default; valid options are `'block'` (visible) or `'none'` (hidden).

Note

Requires Folium's `LayerControl` to be enabled for automatic visibility toggling.

Examples:

```
>>> from pyhelpers.viz import BindColormap
>>> import folium
>>> import branca
>>> m = folium.Map()
>>> test_feature_group = folium.FeatureGroup(name='Test')
>>> test_feature_cm = branca.colormap.LinearColormap(
...     colors=branca.colormap.linear.RdYlGn_06.colors[::-1],
...     vmin=0.0,
...     vmax=1.0,
...     caption="Test",
... )
>>> m.add_child(BindColormap(test_feature_group, test_feature_cm))
>>> m
```

See also

- Example usages: Exploration of UK's Food Hygiene Rating data
- Reference: [VIZ-BC-1].

License

- PyHelpers (since version 2.0.0) is licensed under the MIT License.
- Versions 1.5.2 and earlier are licensed under the GPLv3+ License.

Contributors

- Qian Fu
- Xiangyong Luo

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Z